Eskimon
Glenn
otanoshimini
bgaultier

Geste de savoir

La Fabrication numérique

24 octobre 2023

Table des matières

Introduction 5

I. Arduino 9

I.1. Semaine 1 : Introduction 10

Introduction . . . 10
I.1.1. Organisation générale du MOOC . . . 11
I.1.2. Qu'est-ce que la Fabrication Numérique? . . . 12
I.1.3. Pourquoi Arduino? . . . 14
I.1.4. Liste du matériel nécessaire . . . 15
I.1.5. Fournisseurs Arduino et Composants électroniques . . . 17
Conclusion . . . 17

I.2. Semaine 2 : Les outils 18

Introduction . . . 18
I.2.1. Installation du logiciel Arduino . . . 18
I.2.2. Utilisation du simulateur Arduino . . . 20
I.2.3. Arduino et platine de prototypage . . . 24
I.2.4. Lois simples d'électricité . . . 26
I.2.5. Premier programme: Blink . . . 27
I.2.6. Travaux pratiques . . . 30
I.2.7. Quelques outils de base . . . 30
I.2.8. Des outils spécifiques à l'électronique . . . 33

I.3. Semaine 3 : Capteurs numériques 35

Introduction . . . 35
I.3.1. Corrigé du TP 1 . . . 35
I.3.2. Les capteurs numériques . . . 37
I.3.3. Prototypage et éléctricité . . . 40
I.3.4. Travaux pratiques . . . 42
I.3.5. Détection d'un front . . . 42

I.4. Semaine 4 : Capteurs analogiques 44

Introduction . . . 44
I.4.1. Corrigé du TP 2 . . . 44
I.4.2. Capteurs analogiques . . . 47
I.4.3. Prototypage et électricité . . . 53
I.4.4. Travaux pratiques . . . 55
I.4.5. L'étalonnage, principe et application . . . 56

Contenu masqué . . . 61

I.5. Semaine 5 : Bibliothèques et sorties **64**
Introduction . . . 64
I.5.1. Corrigé du TP 3 . . . 64
I.5.2. Bibliothèques et sorties . . . 67
I.5.3. Complément de cours . . . 70
I.5.4. Les condensateurs . . . 73
I.5.5. Travaux pratiques . . . 80
I.5.6. Environnements de développement . . . 82
I.5.7. Séparer en fichiers . . . 85

II. Modélisation **90**

II.1. Semaine 6 : Modélisation 2D **91**
Introduction . . . 91
II.1.1. Corrigé du TP 4 . . . 91
II.1.2. Introduction à la modélisation 2D . . . 92
II.1.3. Les Diodes (suite) . . . 93
II.1.4. Arduinomètre: Thermomètre avec diode pour sonde . . . 97
II.1.4.1. Première partie . . . 97
II.1.5. Travaux pratiques . . . 102
Contenu masqué . . . 104

II.2. Semaine 7 : Les découpeuses Laser **108**
Introduction . . . 108
II.2.1. Les découpeuses Laser . . . 108
II.2.1.1. La découpe laser par Dimitri . . . 108
II.2.2. Fonctions et boucles . . . 109
II.2.3. Les transistors . . . 111
II.2.3.1. Les Transistors (enfin) . . . 111
II.2.4. Arduinomètre, deuxième partie . . . 116
Contenu masqué . . . 124

II.3. Semaine 8 : Modélisation 3D **125**
Introduction . . . 125
II.3.1. Introduction à la modélisation 3D . . . 125
II.3.2. Initiation à Blender . . . 125
II.3.3. Introduction à la modélisation 3D - Annexe . . . 126
II.3.4. Transistors (suite) . . . 128
II.3.4.1. Transistors à Effet de Champ (FET) . . . 130
II.3.5. Arduinomètre, troisième partie . . . 132
II.3.5.1. Arduino thermomètre avec diode pour sonde . . . 132
II.3.6. Travaux pratiques . . . 140

II.4. Semaine 9 : Les imprimantes 3D **141**
Introduction . . . 141

II.4.1. L'impression 3D . . . 141
II.4.1.1. PRINCIPES DE L'IMPRIMANTE 3D . . . 141
II.4.1.2. PASSER D'UN MODÈLE 3D À DU LANGAGE MACHINE . . . 142
II.4.1.3. ANNEXE: L'IMPRESSION 3D . . . 142
II.4.1.4. L'IMPRIMANTE 3D POUR TOUS - FUTUREMAG - ARTE . . . 142
II.4.2. Les interruptions sur Arduino . . . 142
II.4.2.1. Les interruptions . . . 142
II.4.3. Travaux pratiques . . . 147

II.5. Semaine 10 : Les fraiseuses numériques **150**
Introduction . . . 150
II.5.1. Corrigé du TP: Feu bicolore, barrière et LCD . . . 150
II.5.2. Les fraiseuses numériques . . . 155
II.5.3. Arduino et I²C . . . 156
II.5.3.1. Le bus I²C (Inter Integrated Circuit) . . . 156
II.5.4. Travaux pratiques . . . 163
Contenu masqué . . . 166

II.6. Semaine 11 : Design d'objets **169**
Introduction . . . 169
II.6.1. Modélisation 3D pour l'impression 3D . . . 169
II.6.2. Corrigé du TP Arduino et Processing . . . 169
II.6.3. Design d'objets . . . 170
II.6.4. Les suites de design électroniques . . . 171
II.6.5. Fabrication Numérique . . . 174
II.6.5.1. La Fabrication Numérique . . . 174
Contenu masqué . . . 181

III. Internet of Things **184**

III.1. Semaine 12 : Internet des Objets (1/2) **185**
Introduction . . . 185
III.1.1. Arduino sans l'Arduino . . . 185
III.1.2. Internet des objets . . . 198
III.1.3. Travaux pratiques . . . 198

III.2. Semaine 13 : Internet des Objets (2/2) **200**
Introduction . . . 200
III.2.1. Corrigé du TP 10: Pilotage de LED par Internet . . . 200
III.2.2. Internet des objets . . . 201
III.2.3. Qu'est ce qu'une API? . . . 202
III.2.4. Exemples d'objets connectés . . . 209
III.2.5. Electronique et Arduino . . . 210
III.2.5.1. Arduinomètre connecté . . . 210
III.2.6. Gravure CNC avec Inkscape . . . 218
Contenu masqué . . . 222

IV. Annexe et compléments **226**

Introduction **227**

IV.1. Bases électronique **228**
IV.1.1. Résistances (suite) . . . 228
IV.1.2. Diodes et condensateurs (suite) . . . 231
IV.1.3. Diodes et condensateurs (suite et fin) . . . 233
IV.1.4. Branchements LED . . . 237
IV.1.5. Pont de diodes . . . 237
IV.1.6. Bouton poussoir . . . 240
IV.1.7. Photorésistance (et simulateur) . . . 242
IV.1.8. Caractéristiques des composants . . . 243
IV.1.9. Mesures de tension et d'intensité . . . 244
IV.1.10. Zéro, un, euh... (calcul et notation binaire) . . . 245

IV.2. Éléments de programmation **249**
IV.2.1. Les commentaires . . . 249
IV.2.2. Programmation structurée (programmation avancée) . . . 255
IV.2.2.1. Programmation structurée . . . 255
IV.2.3. Arduino Thérémine et Processing (programmation avancée) . . . 260
IV.2.4. L'indentation . . . 262
IV.2.5. Les tableaux . . . 264
IV.2.6. Les pointeurs (programmation avancée) . . . 270

IV.3. Arduino (documentation et exemples) **275**
IV.3.1. Trucs et astuces . . . 275
IV.3.1.1. Commandes compilateur . . . 276
IV.3.1.2. Mise en forme du texte . . . 277
IV.3.2. Le bouton rotatif . . . 280
IV.3.3. Simulation du bouton rotatif . . . 284
IV.3.4. Quelques liens et autres documents utiles . . . 285
IV.3.4.1. Zeste de Savoir . . . 285
IV.3.4.2. Le site mon-club-elec (à voir vraiment) . . . 285
IV.3.4.3. Le site flossmanuals (les livres FabLabs et Arduino) . . . 286
IV.3.5. Logiciels autour d'Arduino . . . 286
IV.3.6. Lexique électronique et informatique anglais → français . . . 287

IV.4. Arduinomètre **292**
Introduction . . . 292
IV.4.1. Arduinomètre avec sonde LM35 . . . 292
IV.4.1.1. Arduinomètre avec sonde LM35 . . . 292
IV.4.2. Arduinomètre avec thermistance . . . 294

Conclusion **297**

Introduction

Ce cours va vous permettre de vous approprier les outils et les techniques issus des FabLabs: Électronique, Arduino, Design, Internet des objets, modélisation 2D/3D, Imprimantes 3D…

Chaque semaine[1], une vidéo courte vous présentera un nouveau concept de la Fabrication Numérique. Cette vidéo sera accompagnée de sa transcription et éventuellement d'un cours complémentaire.

L'objectif du cours est d'acquérir les compétences de base permettant ensuite aux apprenants de créer à peu près n'importe quoi!

Ce cours a initialement été rédigé via la participation de l'institut "Mines Télécom" de Rennes et publié sur la plateforme gouvernementale FUN .

À qui s'adresse ce cours ?

Ce cours s'adresse aux curieux et aux passionnés du numérique souhaitant découvrir les technologies que l'on trouve dans les FabLabs.

Pré-requis

Une première expérience dans le développement informatique (C, Python, Java) est recommandée (il y a d'autres très bons MOOCs pour se former au développement).

Conditions d'utilisation du contenu

Certains extraits du cours sont sous licence CC-BY-**SA**. Cette dernière sera alors rappelée quand le cas s'applique. Si rien n'est précisé pour un extrait, c'est alors la licence CC-BY qui s'applique.

Auteurs de ce cours

Photo	Présentation

/opt/zds/data/contents-public/la-fabrication-numerique__building/extra_cor	**Ingénieur passionné par les FabLabs** (@galouf), Baptiste a débuté sa carrière en 2009 à l'Institut Mines Télécom en tant qu'ingénieur de recherche. Il s'est intéressé à l'informatique embarquée dans les véhicules autonomes pour rejoindre ensuite une équipe spécialisée dans l'internet des objets. Baptiste est un passionné d'open-source et d'open-hardware; il utilise aujourd'hui les technologies de la Fabrication Numérique pour ses projets professionnels et personnels. Il enseigne depuis 2012 ces thématiques au sein de l'École des Beaux-Arts de Rennes, à l'école d'ingénieur Télécom Bretagne mais aussi dans un des premiers FabLabs français, le LabFab .
/opt/zds/data/contents-public/la-fabrication-numerique__building/extra_cor	**Ingénieur, bidouilleur, animateur et inventeur**, avec plus d'une vingtaine d'années d'expérience dans l'industrie en tant qu'ingénieur en électronique et informatique industrielle, Glenn apporte son savoir-faire et son enthousiasme avec l'envie de partager et bidouiller ensemble. Glenn est aussi porteur d'un projet de "open space" chez lui dans la région Midi-Pyrénées.
/opt/zds/data/contents-public/la-fabrication-numerique__building/extra_cor	**Bidouilleur curieux** (@otanoshimini), Laurent est curieux! Il a passé sa carrière dans des domaines variés (post-production, design, informatique, services généraux, web…). Il prône la transversalité et l'esprit d'exploration. Il a toujours aimé créer, bricoler, et il continue aujourd'hui avec tous les outils qu'il trouve!
/opt/zds/data/contents-public/la-fabrication-numerique__building/extra_cor	**Dresseur de robots** (@Eskimon_fr), Simon est un passionné de robotique et de l'embarqué depuis ses études. Après plusieurs participations en coupe de robotique et plein de choses apprises, il a décidé de transmettre sa passion et son esprit de partage en écrivant des tutoriels, son plus gros ouvrage étant un tutoriel sur Arduino .

/opt/zds/data/contents-public/la-fabrication-numerique__building/extra_cor	**Évangéliste suprême** (@JohnDaYoung), Hacker de toujours, John baigne dans la Fabrication Numérique depuis que celle-ci existe (peut être même avant). Pédagogue de génie, John a initié un nombre colossal de personnes à la soudure, au développement web, à Arduino et aux machines CNC sur tous les continents et dans toutes les conditions. Figure emblématique de la culture DIY, c'est tout naturellement que le LabFab a recruté John en tant que facilitateur. Poste qu'il occupe toujours aujourd'hui.

/opt/zds/data/contents-public/la-fabrication-n

FIGURE .0.1. – La fabrication numérique

Si vous souhaitez poser des questions sur le forum, n'hésitez pas à ouvrir un sujet dans la rubrique "Systèmes et Matériels ". Cependant, pour que tout reste clair, merci de respecter la convention de nommage suivante: `[Tuto fablab][Semaine x] Titre-du-sujet`

Et voici sans plus attendre la vidéo bande-annonce de ce cours!

ÉLÉMENT EXTERNE (VIDEO) —
Consultez cet élément à l'adresse https://geo.dailymotion.com/player.html?video=x15c9xk&.

1. Ce cours a initialement été publié sous forme d'un MOOC sur la plateforme gouvernementale FUN. C'est pourquoi il a été et reste divisé en semaines.

Première partie

Arduino

I.1. Semaine 1 : Introduction

Introduction

Découverte des Fablabs

Cette semaine est consacrée à la découverte des FabLabs au travers de la vidéo "Qu'est-ce que la Fabrication Numérique? ". Nous vous invitons aussi à visiter le FabLab le plus proche de chez vous et à participer à un des ateliers organisés dans ces lieux de fabrication (carte des FabLabs). Le nombre de FabLabs dans le monde ne cesse d'augmenter et plus de 100 nouveaux FabLabs sont apparus dans le monde depuis la fin de la première session du MOOC.

Pour compléter cette vidéo, nous vous invitons à lire le livre Fablab, Hackerspace, les lieux de fabrication numérique collaboratifs . Vous pouvez aussi le télécharger en PDF ici .

Découverte de la plateforme Arduino

Vous allez aussi découvrir ce qu'est la plateforme Arduino, plateforme intéressante lorsque l'on souhaite prototyper des objets. Vous allez ainsi comprendre pourquoi nous l'avons sélectionnée comme plateforme de référence pour ce MOOC.

À ce sujet, vous pouvez aussi lire le livre sur Arduino .

Outils

Annexes Au fil du temps, une annexe assez conséquente a vu le jour. N'hésitez pas à aller y chercher des informations supplémentaires (et essayez d'y faire un tour pour trouver réponse à vos questions).

Forum Vous pourrez poser toutes vos questions sur le forum . Nous vous souhaitons une très bonne semaine 1!

L'équipe du MOOC.

I.1.1. Organisation générale du MOOC

Organisation générale

Le cours a été initialement publié sur la plateforme gouvernementale FUN. Chaque semaine commençait le mardi à 10h, heure de Paris. Vous pourrez désormais le suivre à la vitesse qui vous plaira, puisque le cours est dorénavant disponible intégralement et sans interruption.

Organisation d'une semaine type

Chaque semaine illustre un concept lié à la Fabrication Numérique. À la fin de la vidéo, nous vous proposons la transcription de la vidéo en `.pdf`, que vous pourrez imprimer si vous le souhaitez.

À chaque fin de cours, vous aurez à répondre soit à des QCM, soit des travaux pratiques.

Si vous avez des questions autour de l'organisation générale du MOOC, n'hésitez pas à nous en parler sur le Forum.

Planning

Le MOOC s'étend sur 13 semaines en trois modules:

- Module 1 - **Prototypage électronique avec Arduino**
  - Semaine 1: **Introduction**
  - Semaine 2: **Installation des outils**
  - Semaine 3: **Les capteurs numériques**
  - Semaine 4: **Les capteurs analogiques**
  - Semaine 5: **Librairies et sorties**
- Module 2 - **Modélisation et machines à commandes numériques**
  - Semaine 6: **Modélisation 2D**
  - Semaine 7: **Les découpeuses Laser**
  - Semaine 8: **Modélisation 3D**
  - Semaine 9: **Les imprimantes 3D**
  - Semaine 10: **Les fraiseuses numériques**
  - Semaine 11: **Design d'objets**
- Module 4 - **Internet des objets**
  - Semaine 12: **Internet des objets 1/2**
  - Semaine 13: **Internet des objets 2/2**

Pour souder la communauté, nous vous invitons à collaborer au travers du **forum** pour échanger, partager et poser vos questions. Ce MOOC est vivant, ne l'oubliez pas! Utilisez le tag `[Tuto fablab]` pour rassembler le maximum de monde.

Wiki:

Le wiki est aussi un outil de partage et nous comptons sur vous pour le consulter, mais aussi pour compléter les différents documents proposés. Nous vous invitons à remplir les pages avec vos idées, suggestions, remarques ou critiques. Nous avons ainsi ouvert des pages pour:

Le wiki a été exporté en annexe de ce cours, si vous désirez y ajouter des éléments, n'hésitez pas à en parler sur le forum!

— **L'Agenda** (Si vous avez connaissance d'un évènement autour des FabLabs, n'hésitez pas à en faire part à toute la communauté)
— **Liste du matériel** (Là aussi, vous avez certainement des idées de matériel supplémentaire)
— **Liste des fournisseurs** (Nous avons établi une liste de fournisseurs, mais cette liste a vocation à être complétée)
— **Arduino (documentation et exemples).** (Vous souhaitez ajouter des informations supplémentaires sur le sujet? Ne vous gênez pas!

Ainsi, nous remercions à l'avance tous les futurs contributeurs.

L'équipe du MOOC.

I.1.2. Qu'est-ce que la Fabrication Numérique ?

Élément externe (Video) —
Consultez cet élément à l'adresse https://geo.dailymotion.com/player.html?video=x2kh01c&.

Partout dans le monde, dans plus de 100 pays, des lieux aux objectifs et dimensions variables, hackerspaces, fablabs, fabshops, investissent des registres autrefois réservés aux professionnels de la conception, du prototypage et même de la distribution des objets. Leur point commun? Un alphabet technique et des méthodes hybridant les manières de faire et créer ensemble dans les mondes physique et numérique.

La connaissance des briques de base de cet alphabet permet à la fois de concevoir ensemble, physiquement ou non, des objets nouveaux, notamment interactifs et communicants, mais aussi de les partager, de les améliorer, et de les transmettre sans déplacer un gramme de matière.

Cela est rendu possible par la mise au point à la fois d'outils et de méthodes pédagogiques ouverts non aux spécialistes, mais à tous et toutes, et permettant de manière de concevoir, tester et diffuser à peu près n'importe quoi en utilisant la puissance combinée du lien social et de l'internet global.

Tout comme l'existence d'un standard ouvert pour les pages web permet de lire des contenus sur toute marque d'ordinateur et sur tout navigateur, la fabrication numérique repose sur des fondamentaux et standards techniques universels et ouverts capables d'être utilisés par quasiment tous les ordinateurs, machines-outils et composants programmables.

La démocratisation de ces outils et la transmission des méthodes collaboratives provoque une révolution comparable à celle du web 2.0, mais dans le registre des objets itérables, voire viraux, qui peuvent être désormais conçus pour régler le problème d'une personne unique, de petits groupes, ou de millions de personnes.

Pour ce qui concerne les laboratoires de fabrication numériques ou Fablabs, leur philosophie globale consiste à donner aux personnes plus de pouvoir pour exprimer leur créativité, intelligence et imagination et l'appliquer immédiatement et concrètement à la résolution de problèmes sur le terrain. Les méthodes et outils privilégient l'intelligence collective autour de viviers de partages d'objets, de code et de savoir-faire. Ce qui est également intéressant, c'est que dans les fablabs, il n'y a pas de théorie sans pratique: la réalisation prime avant tout, et l'erreur fait partie du jeu pour progresser.

Les briques de base transmises et éprouvées par le faire dans les laboratoires de fabrication ou Fablabs sont au nombre de quatre:

La première consiste à pouvoir récupérer, créer, modifier, fabriquer et transmettre des objets interactifs et communicants. En clair, cela signifie comprendre et créer des comportements programmables dans tout objet ou réseau d'objets. Cela n'est désormais plus l'apanage des ingénieurs mais peut être réalisé par tous à l'aide d'outils à la fois pédagogiques et efficaces sur le terrain, comme la carte électronique programmable arduino. Le format `.ino` est d'ailleurs l'un des formats de référence permettant mondialement la réutilisation concrète de l'interaction inventée ailleurs sur la planète. Un téléchargement de plan de montage et de code actif se transforme ainsi en objet dont la dimension n'est plus seulement matérielle, mais aussi numérique. Arduino n'est pas la seule voie pour vulgariser et partager ces possibles, mais permet de comprendre et pratiquer immédiatement l'algorithmie, autrement dit le fameux «CODE,» et de le démystifier.

La deuxième consiste à pouvoir récupérer, créer, modifier, fabriquer et transmettre des formes par l'utilisation combinée de la modélisation ou du scan 3D et de l'impression 3D. Le format de référence est le `.stl` utilisable sur toute la planète. Cette brique permet la transmission de formes reconstituées par impression 3D et donc le déplacement d'objets sans bouger un gramme de matière quel que soit le modèle de machine.

La troisième consiste à pouvoir récupérer, créer, modifier, fabriquer des formes par enlèvement de matière (gravure, découpe) en se basant sur des outils de conception bi-dimensionnels. Le format de référence est le `.dxf` et les machines utilisées sont surtout des fraiseuses et découpeuses. Cette technique permet notamment de pouvoir transmettre des mobiliers et objets de grande taille découpés dans des matières très variées, papier, plastiques, bois, et même métal.

La quatrième brique est fondamentale. Sans elle il n'y aurait pas de progression rapide ni de réutilisation ou transmission possible des objets. Il s'agit de la documentation des objets créés. Cette documentation doit être comprise comme la condition de l'intelligence collective et de l'ouverture. Il ne s'agit pas de recréer des micro-communautés de spécialistes, mais bien d'élargir les possibles au niveau planétaire et de permettre l'amélioration par tous, dans une dynamique comparable à celle de wikipédia.

Contrairement aux idées reçues, il n'est nul besoin d'être ingénieur pour comprendre et utiliser ces briques de base. Chacun et chacune les éprouvera d'abord pour son projet d'objet futile, utile ou artistique. mais une fois acquises, elles ouvrent un immense champ de possibles tant au niveau de l'amélioration, de la réutilisation, que de la collaboration. Les effets en sont tout de suite mis à l'épreuve des faits, car les objets sont tangibles et opérationnels. Le pouvoir de faire

et celui de comprendre se combinent ici, tout comme celui du groupe bricolant ensemble et du réseau planétaire, mais avec des outils puissants capables d'investir tous les registres.

Alors à vous d'apprendre et surtout, à vous de fabriquer!

Hugues Aubin

Chargé de mission TIC à la ville de Rennes et co-fondateur du LabFab

I.1.3. Pourquoi Arduino ?

Élément externe (Video) —
Consultez cet élément à l'adresse https://geo.dailymotion.com/player.html?video=x2kh00f&.

Merci d'avoir regardé cette vidéo, c'est la toute première que nous ayons tournée en février 2013 pour la première édition du MOOC. Elle n'est donc pas exempte de défauts, mais nous nous sommes améliorés dans les suivantes.

Nous espérons qu'elle vous a convaincu qu'Arduino est une plateforme intéressante lorsque l'on souhaite prototyper des objets. Aussi, nous espérons qu'elle deviendra votre couteau suisse pour mettre de l'intelligence dans vos futurs projets.

Figure I.1.1. – Une Arduino Uno

Nous avons choisi Arduino car celle-ci est:

— **ouverte**: le code source et les schémas électroniques du logiciel et du matériel Arduino sont open-source. Ce qui signifie que l'on peut comprendre comment fonctionne un Arduino, le cloner, le modifier, l'adapter pour l'intégrer ensuite dans toute sorte de projet.
— bien **documentée**: l'utilisation de toutes les instructions dont on peut tirer parti avec Arduino est détaillée dans une référence accessible en ligne.
— **peu couteuse**: le prix d'un Arduino officiel classique (appelé Arduino UNO) est d'environ 20€ TTC.
— adoptée par une **large communauté**: c'est probablement le plus important. Le nombre croissant de personnes qui utilisent cette plateforme est un atout indéniable. En effet, grâce à la puissance d'Internet, il est possible de trouver de la documentation, des tutoriels qui traitent de toutes les thématiques dans lesquels on peut trouver un Arduino:

```
-   Apprentissage de l'électronique
-   Domotique
-   Robotique
-   Vêtements intelligents
-   Installations artistiques
-   Machines à commande numérique
```

Pour toutes ces raisons, nous avons choisi de sélectionner Arduino comme **la** plateforme de ce MOOC.

Sachez également qu'il existe des alternatives à Arduino.

Juste pour en citer quelques-unes: Raspberry Pi , BeagleBoard , MSP430 LaunchPad , mbed ... sont des plateformes particulièrement intéressantes pour mettre de l'intelligence dans des objets. Ces plateformes seront comparées à Arduino à la fin du module 1 de ce MOOC.

La question que vous vous posez peut-être maintenant est:

? Est ce qu'il faut que j'achète un Arduino pour suivre ce MOOC?

La réponse est non, l'achat d'un Arduino est optionnel. En effet, vous pourrez suivre la quasi totalité des modules consacrés à Arduino grâce à un simulateur en ligne qui sera détaillé la semaine prochaine. C'est moins cool, mais ça fonctionne!

Par contre, si vous souhaitez prototyper avec un vrai Arduino, différents choix s'offrent à vous:

— soit vous rendre dans votre FabLab le plus proche et participer à un des ateliers organisés dans ces lieux de fabrication (carte des FabLabs)
— soit acquérir un kit ou compléter votre matériel existant grâce à la liste de matériel présente ici .

I.1.4. Liste du matériel nécessaire

I.1.4.0.1. Liste du matériel nécessaire

Vous souhaitez prototyper sur du vrai matériel et investir dans du matériel?

Alors vous êtes sur la bonne page! Le matériel ci-dessous est nécessaire si vous souhaitez réaliser les montages présentés tout au long du MOOC. Les composants décrits ci-dessous sont vendus à l'unité ou sous forme de **kit de démarrage** (appelé souvent *Starter Kit* sur les boutiques en ligne présentes dans la sous-partie suivante).

Nous n'avons pas souhaité mettre en avant une boutique ou un kit particulier sur ce site afin que vous puissiez **comparer** les offres des différents sites marchands (énumérés dans la sous-partie suivante). Nous vous recommandons de faire attention à la **qualité** des kits ou composants que vous allez acquérir, à la documentation et au sérieux des revendeurs.

Enfin, nous vous encourageons (beaucoup) à venir **échanger** sur notre forum ou sur Facebook **avant** de vous lancer dans un achat.

I.1.4.0.1.1. Kit de démarrage : l'essentiel

- Arduino UNO R3 avec son câble USB ×1 (ou équivalent)
- Platine de prototypage × 1 (aux moins 500 trous)
- Kit de câbles de prototypages × 1
- LED de différentes couleurs (au moins 2 × vert, 2 × rouge, 2 × orange ou jaune)
- Résistances de différentes valeurs:
  - 10kΩ × 5
  - 4,7kΩ × 5
  - 1kΩ × 5
  - 220Ω × 5
  - 150Ω × 5
- Condensateurs céramiques de différentes valeurs:
  - 100nF × 2
  - 10nF × 2
- Condensateurs chimiques de différentes valeurs:
  - 10uF × 2
  - 47uF × 2
  - 470uF × 2
- Diodes 1N4148 × 2 (ou autres diodes)
- Transistor NPN (TO92) × 2 (ex. BC337, BC546, 2N2222, 2N3904 …)
- Transistor PNP (TO92) × 2 (ex. BC327, BC556, 2N2907, 2N3906 …)
- Photorésistance × 1
- Boutons poussoirs × 5
- Potentiomètre 10k × 1
- Potentiomètre 10k ou 50k avec bouton × 1
- Piezzo buzzer × 1

I.1.4.0.1.2. Matériel optionnel

- Module d'extension (shield ou module) écran LCD 16×2
- LED RVB × 1
- Bouton codeur avec contact × 1
- Thermistance (100k ou 50k) × 1
- LM35 (sonde de température) × 1
- Transistor MOSFET-N "Logic level" × 1 (ex. IRL530)
- Transistor MOSFET-P "Logic level" × 1 (ex. IRF9530)
- Module Relais pilotable en 5V pour commuter jusqu'à 230V × 1
- Module d'extension (shield ou breakout board) Ethernet x 1 (choisir un des deux options):

```
-   basé sur chip Wiznet W5100  ( plus cher mais plus facile
    à utiliser
avec l'Arduino UNO )
-   basé sur chip ENC28J60  (moins cher mais gourmand en
    mémoire)
```

— Mini Servo Moteur × 1 (choisir le plus grand angle d'ouverture possible)
— Boîtier plastique pour contenir les composants et l'Arduino × 1 (vraiment optionnel 🍊

I.1.5. Fournisseurs Arduino et Composants électroniques

I.1.5.0.1. Fournisseurs Arduino et composants electroniques

Si vous souhaitez acheter un kit de démarrage Arduino ou simplement compléter le matériel dont vous disposez déjà, vous trouverez ci-dessous la liste des fournisseurs Arduino Français.

I.1.5.0.1.1. Liste des fournisseurs (par ordre alphabétique)

— AlphaCrusis
— Arobose
— Bosc Technologie Services (afficheur LCD 2 x 16 seulement)
— Conrad
— Elecdif.com
— Evola
— Gotronic
— HackSpark
— Letmeknow
— Lextronic
— L'Impulsion
— Matlog
— Mouser
— RS
— Selectronic
— Semageek (Kit de démarrage préparé spécialement pour le MOOC)
— Snootlab (Kit de démarrage préparé spécialement pour le MOOC)
— Zartronic

Aussi, nous tenons à remercier tous les fournisseurs présents sur cette page, qui ont pris le temps de répondre à nos questions et donner leur accord pour apparaître sur ce site.

Notes:

— Les cartes Arduino UNO officielles sont documentées, fabriquées en Italie et ont une empreinte carbone nulle. Ce qui signifie que l'open-source, le droit du travail européen et le respect de l'environnement sont au coeur des préoccupations de l'équipe Arduino. Nous partageons ces valeurs et vous invitons à vous méfier des cartes "low cost".

Conclusion

Suite à la proposition de correction de Corentin, je change le lien vers la carte des labs

I.2. Semaine 2 : Les outils

Introduction

Toujours connaître ses outils Et c'est parti pour la semaine 2! Cette semaine, nous allons découvrir les outils à notre disposition pour réaliser les montages et exercices des prochaines semaines consacrées au *Prototypage électronique avec Arduino*.

Comme nous l'avons indiqué la semaine passée, deux choix s'offrent à vous pour réaliser des montages avec Arduino:

— Utiliser **le logiciel Arduino** installé sur votre ordinateur. Ce logiciel va nous permettre d'envoyer des ordres à un Arduino pour qu'il les exécute. Si vous comptez utiliser un Arduino (le vôtre, celui d'un(e) amie ou celui de votre FabLab);
— Utiliser le **simulateur 123D Circuits** directement dans votre navigateur internet. Ce logiciel permettra de simuler le fonctionnement d'un Arduino.

Forum Sur le forum, nous vous conseillons de poser vos questions grâce aux mots-clefs `[Tuto fablab][Semaine 2]`. Voici quelques exemples:

— [Tuto fablab] Questions sur le simulateur
— [Tuto fablab] Questions sur les bases de l'électricité
— [Tuto fablab] TP feu tricolore

Bonne semaine,

L'équipe du MOOC.

I.2.1. Installation du logiciel Arduino

Élément externe (Video) —
Consultez cet élément à l'adresse https://geo.dailymotion.com/player.html?video=x2kgzt1&.

Voici donc notre premier programme (celui qui est décrit dans la vidéo). Cette suite d'instructions va faire clignoter une LED branchée sur la broche 13 de l'Arduino toutes les secondes. Lorsque

vous utilisez le logiciel Arduino, il peut être trouvé en cliquant sur Fichier → Exemples → 01.Basics → Blink. Vous pouvez également copier le code suivant dans votre éditeur:

I.2.1.0.0.1. Programme

```
/*
  Clignotement
  Allume la LED pendant 1 seconde,
  puis l'éteint pendant 0,5 seconde.
*/

// Numéro de la broche à laquelle est
// connectée la LED
int led = 13;

// le code dans cette fonction est exécuté une fois au début
void setup() {
  // indique que la broche de la LED une sortie :
  // on va modifier sa tension
  pinMode(led, OUTPUT);
}

// le code dans cette fonction est exécuté en boucle
void loop() {
  digitalWrite(led, HIGH);   // allumer la LED (tension 5V sur la
    broche)
  delay(1000);               // attendre 1000ms = 1s
  digitalWrite(led, LOW);    // éteindre la LED (tension 0V sur la
    broche)
  delay(1000);               // attendre à nouveau 1seconde
}
```

Listing 1 – Le clignotement

Pour comprendre tout ça, nous vous invitons à consulter la section consacrée au code de notre premier programme ici .

I.2.1.0.0.2. Références

- Référence mini du langage Arduino par Xavier Hinault
- Référence officielle du langage Arduino (anglais) par l'équipe d'Arduino
- Compléments sur la programmation Arduino par Eskimon

I.2.1.0.0.3. Licence

FIGURE I.2.1. – CC-BY-SA

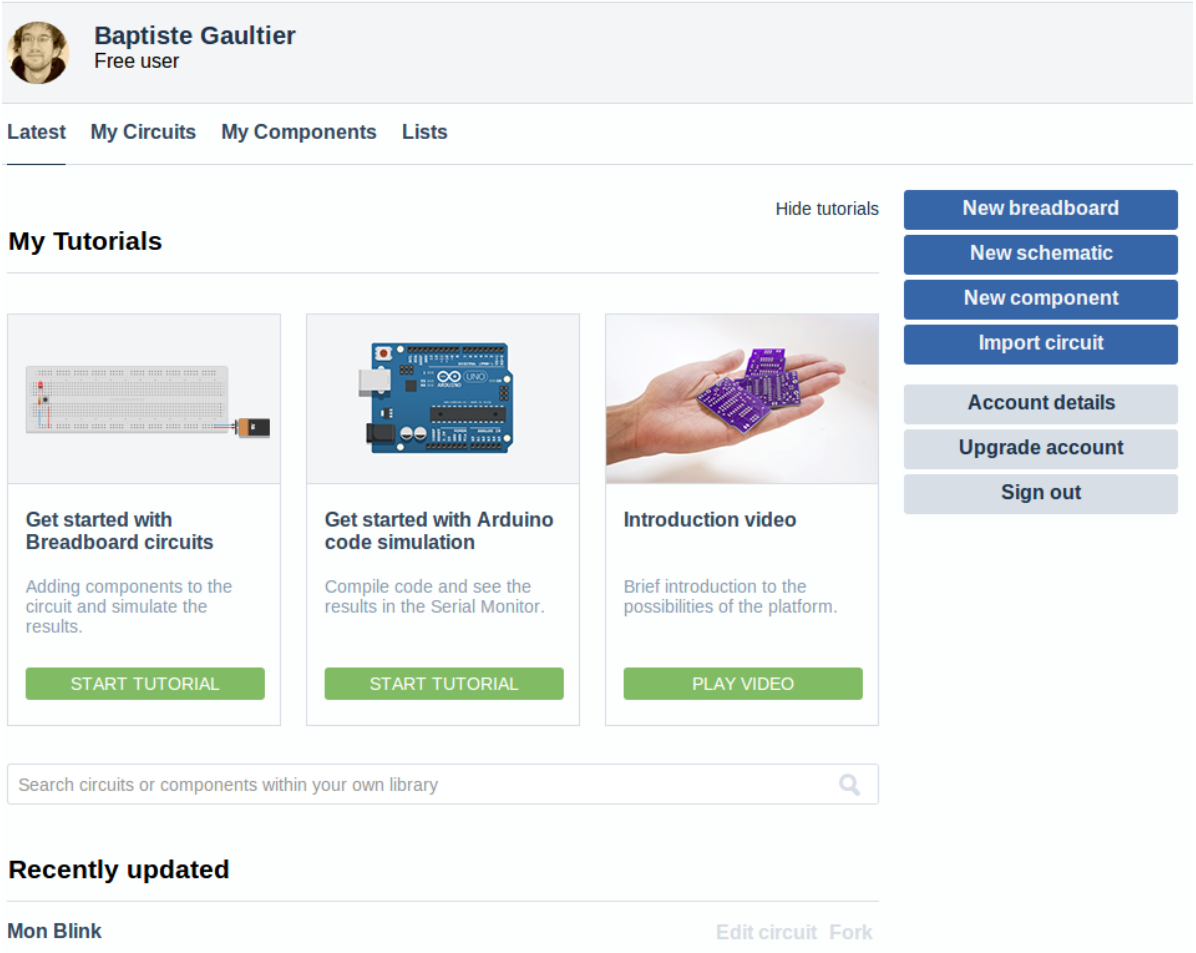
I.2.2. Utilisation du simulateur Arduino

I.2.2.0.1. Utilisation du simulateur Arduino

Avant de vous lancer dans l'achat d'un Arduino, il est possible de réaliser des montages Arduino basiques grâce à un simulateur en ligne gratuit (mais propriétaire).
Ce simulateur, appelé **123D Circuits** , est proposé par la société ©Autodesk. Si, vous ne souhaitez pas utiliser un vrai Arduino pour l'instant, suivez le tutoriel ci-dessous:

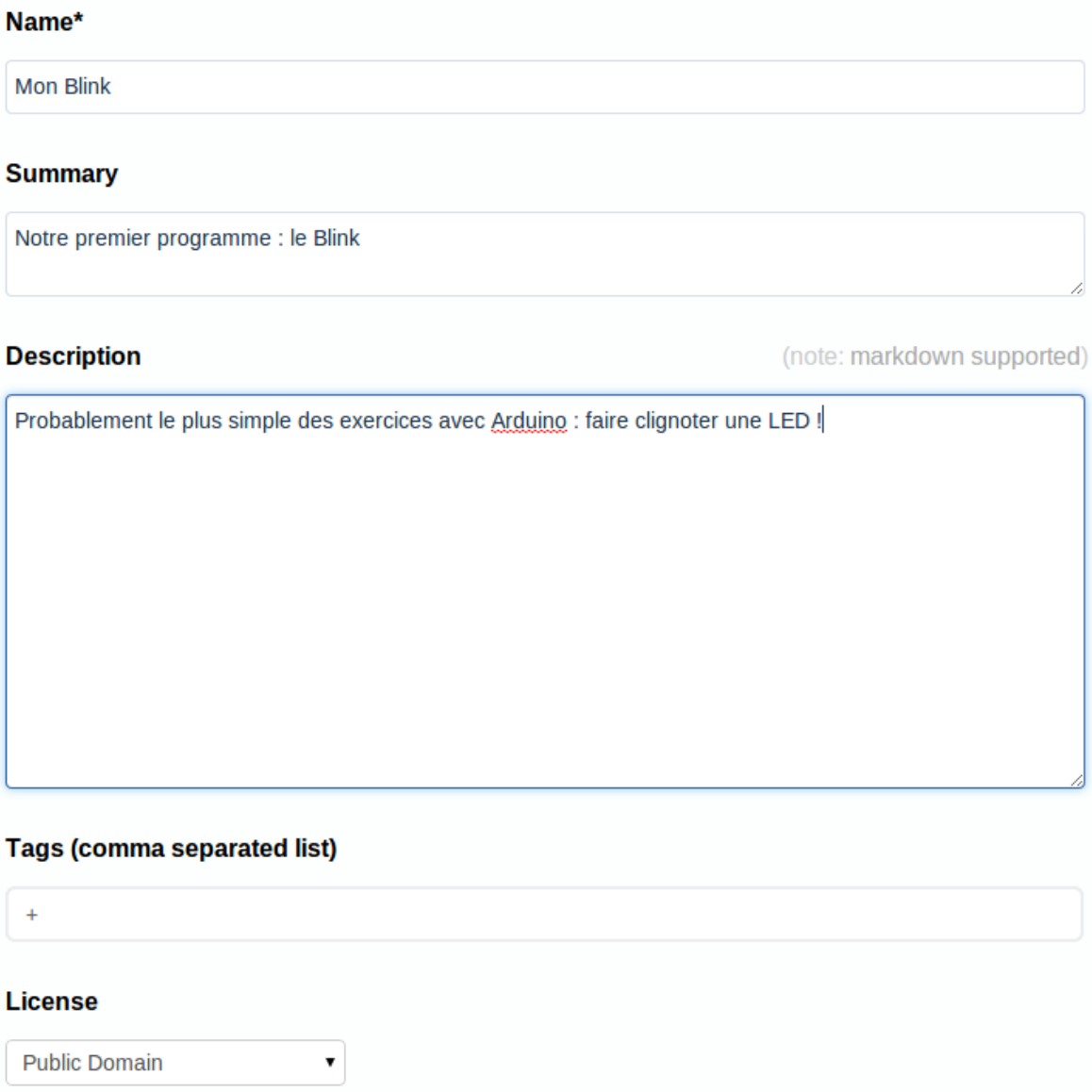
I.2.2.0.1.1. Tutoriel

1. Créez tout d'abord un compte sur le site 123d.circuits.io . Nous vous recommandons d'utiliser le navigateur Chrome qui est requis pour utiliser pleinement toutes les fonctionnalités du simulateur.
2. Une fois votre compte créé, vous devriez accéder à cette page:

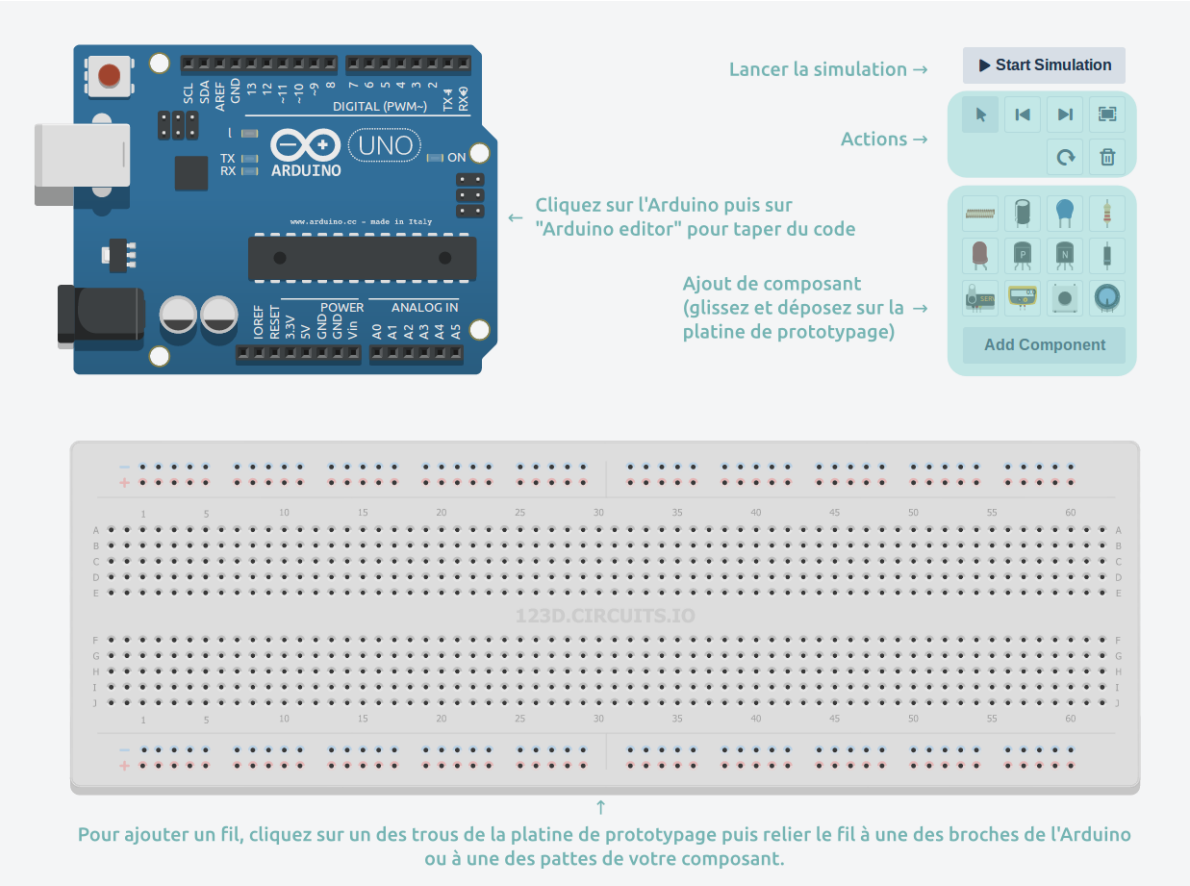


3. Pour créer un nouveau montage, veuillez cliquer sur ce bouton:

4. Remplissez les informations nécessaires dans la page qui s'ouvre puis cliquez sur *Create New!*:



5. Une fois le simulateur chargé, voici les différentes actions possibles dans le simulateur:



6. Pour envoyer des instructions à Arduino, cliquez sur celui-ci puis sur *Arduino Editor*.
7. Copiez le code suivant:

```
/*
  Clignotement
  Allume la LED pendant 1 seconde,
  puis l'éteint pendant 1 seconde.
*/

// Numéro de la broche à laquelle est
// connectée la LED
int led = 13;

// le code dans cette fonction est exécuté une fois au début
void setup() {
  // indique que la broche de la LED une sortie :
  // on va modifier sa tension
  pinMode(led, OUTPUT);
}

// le code dans cette fonction est exécuté en boucle
void loop() {
  digitalWrite(led, HIGH);   // allumer la LED (tension 5V
    sur la broche)
  delay(1000);               // attendre 1000ms = 1s
```

```
22    digitalWrite(led, LOW);    // éteindre la LED (tension 0V
        sur la broche)
23    delay(1000);               // attendre à nouveau 1seconde
24  }
```

et collez le dans l'éditeur, vous devriez avoir ceci:

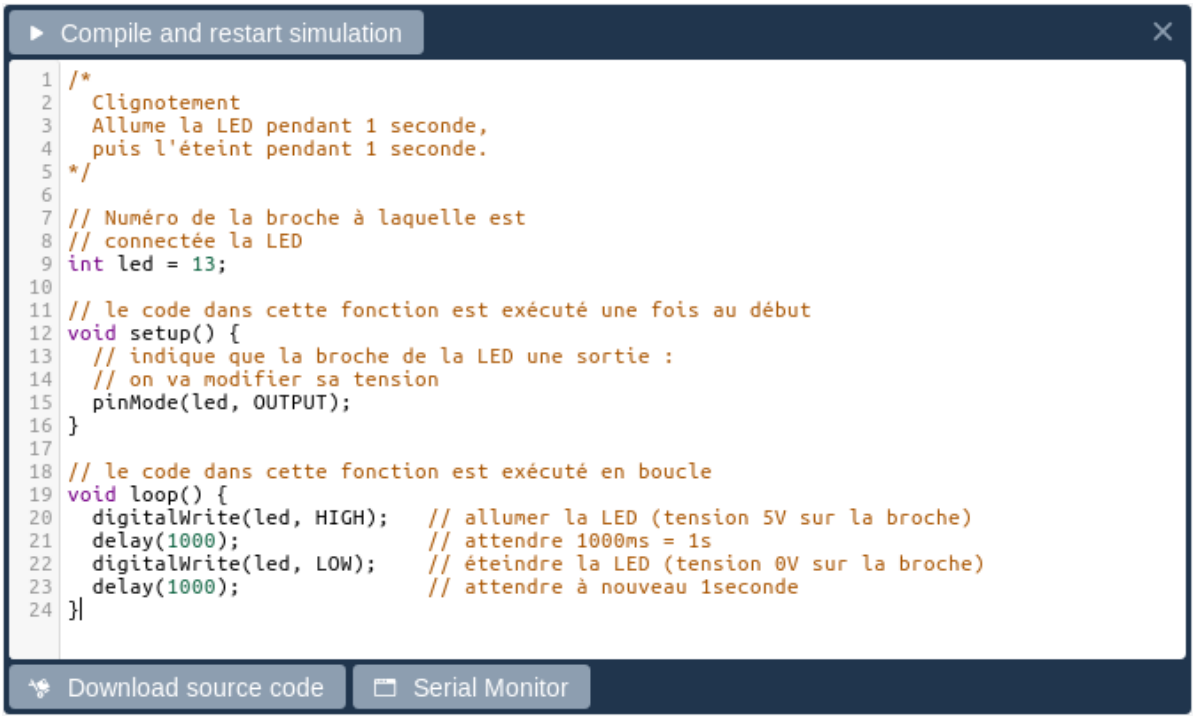
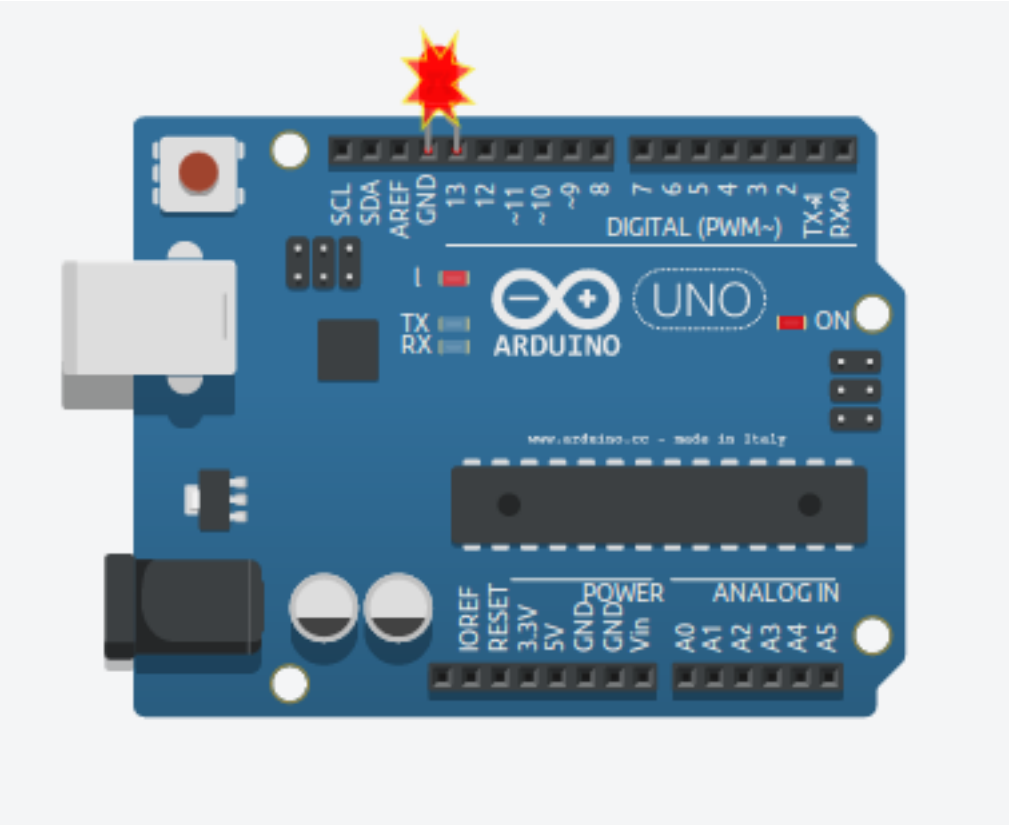


FIGURE I.2.2. – editor

Cliquez ensuite suite *Compile and restart simulation*

8. Enfin, ajoutez une LED sur la broche 13 en la faisant glisser depuis la droite directement sur les broches `13` et `GND` de l'Arduino:



9. Joie et jubilation, notre LED virtuelle clignote! Passez vite à la suite pour comprendre le code que vous avez tapé!

I.2.2.0.1.2. Licence

/opt/zds/data/contents-public/la-fabrication-n

Figure I.2.3. – CC-BY-SA

I.2.3. Arduino et platine de prototypage

I.2.3.0.0.1. Arduino

Petits rappels sur les broches d'Arduino:

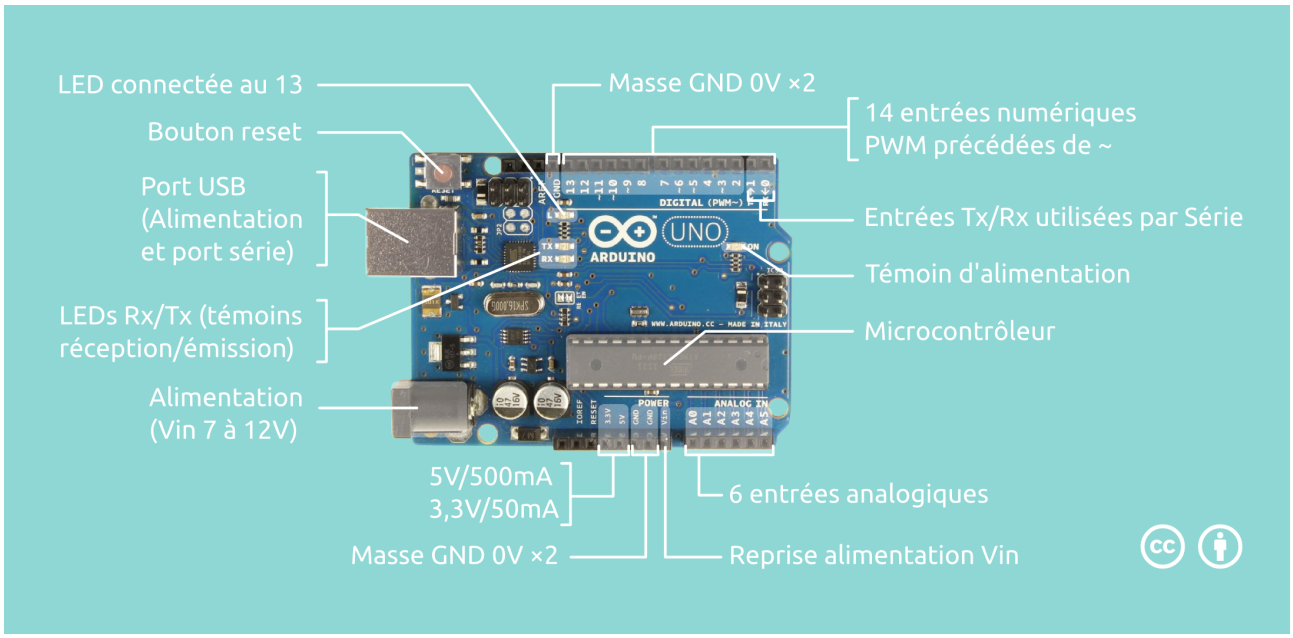


FIGURE I.2.4. – Les broches d'Arduino

Ensuite, pour éviter de tout brancher directement sur l'Arduino, nous allons utiliser une platine d'essai ou **breadboard**.
Pour nous le montrer, Eskimon nous explique ceci :
"Je vais maintenant vous présenter un outil très pratique lorsque l'on fait ses débuts en électronique ou lorsque l'on veut tester rapidement/facilement un montage. Cet accessoire s'appelle une breadboard *(littéralement: planche à pain, techniquement: plaque d'essais sans soudure). Pour faire simple, c'est une plaque pleine de trous !"*

I.2.3.0.0.2. Principe de la breadboard Certes, la plaque est pleine de trous, mais pas de manière innocente! En effet, la plupart d'entre eux sont reliés. Voici un petit schéma rapide qui va aider à la compréhension.

FIGURE I.2.5. – Une breadboard

Comme vous pouvez le voir sur l'image, j'ai dessiné des zones. Les zones rouges et noires correspondent à l'alimentation. Souvent, on retrouve deux lignes comme celles-ci permettant de relier vos composants aux alimentations nécessaires. Par convention, le **noir** représente la masse et le **rouge,** l'alimentation (+5V, +12V, -5V… ce que vous voulez y amener). Habituellement, tous les trous d'une même **ligne** sont reliés sur cette zone. Ainsi, vous avez une ligne d'alimentation parcourant le long de la carte.
Ensuite, on peut voir des zones en bleu. Ces zones sont reliées entre elles par **colonne**. Ainsi, tous les trous sur une même colonne sont reliés entre eux. En revanche, chaque colonne est distincte. En faisant chevaucher des composants sur plusieurs colonnes, vous pouvez les connecter entre eux.

Dernier point, vous pouvez remarquer un espace coupant la carte en deux de manière symétrique. Cette espace coupe aussi la liaison des colonnes. Ainsi, sur le dessin ci-dessus, on peut voir que chaque colonne possède 5 trous reliés entre eux. Cet espace au milieu est normalisé et doit faire la largeur des circuits intégrés standards. En posant un circuit intégré à cheval au milieu, chaque patte de ce dernier se retrouve donc sur une colonne, isolée de la précédente et de la suivante.
Si vous voulez voir plus concrètement ce fonctionnement, je vous conseille d'essayer le logiciel Fritzing , qui permet de faire des circuits de manière assez simple et intuitive. Vous verrez ainsi comment les colonnes sont séparées les unes des autres.

I.2.3.0.0.3. Références

— Compléments sur la programmation Arduino par Eskimon

I.2.4. Lois simples d'électricité

I.2.4.0.1. Lois simples d'électricité

Avant de réaliser notre premier montage et de brancher notre première LED, il nous a semblé important de rappeler quelques bases en électricité. Glenn nous explique ça dans la section suivante:

I.2.4.0.1.1. Tension et intensité À partir du circuit électrique le plus simple, jusqu'aux circuits les plus complexes, nous avons affaire à un courant électrique qui circule. Pour parler de la magnitude de ce courant électrique, on parle de l'**intensité**. Ce courant ou intensité est mesuré et exprimé en Ampères (A).
Pour faire circuler un courant électrique, il faut une "force motrice". En électricité, cette force est appelée **tension**. La tension est mesurée et exprimée en Volts (V).
Pour visualiser la différence entre **Intensité** et **Tension,** on peut faire un parallèle avec l'eau. L'intensité dans un circuit électrique est comme le débit de l'eau dans un tuyau. Et la tension électrique est comme la pression de l'eau. Pour un tuyau donné, plus on augmente la pression, plus le débit d'eau augmente. En électricité c'est pareil: dans un circuit donné, plus on augmente la tension, plus l'intensité augmente.

I.2.4.0.1.2. Résistance Toujours avec notre tuyau d'arrosage, on peut visualiser une autre manifestation électrique: la **résistance**. La friction à l'intérieur du tuyau résiste au passage de l'eau. Plus le diamètre du tuyau est petit, plus la résistance est importante. Plus le tuyau est long, plus la résistance est forte. Et si, pour augmenter le débit, on augmente trop la pression – le tuyau se rompt.
La résistance électrique est très semblable. La résistance est mesurée et exprimée en Ohms (Ω). Un simple fil électrique en cuivre montre une résistance au courant. Plus le diamètre du fil est petit, plus il y a de résistance. Plus le fil est long, plus il y a de résistance. Vous avez peut-être déjà expérimenté: si on essaie de passer trop de courant dans un petit fil électrique – le fil fond (c'est le principe d'un fusible).

I.2.4.0.1.3. Loi d'Ohm Bon, nous avons déjà trois phénomènes électriques à maîtriser: la **tension** (volts), **l'intensité** (ampères) et la **résistance** (ohms). Ces trois phénomènes sont intimement liés par une loi d'électricité dite loi d'Ohm. La loi s'exprime dans une formule très simple:

$$U = R \times I$$

avec:

- U: Tension
- I: Intensité
- R: Résistance

La même loi peut être exprimée aussi en fonction de l'intensité et la résistance respectivement: $I = \frac{U}{R}$ et $R = \frac{U}{I}$

Pour terminer, sachez qu'il vous faudra protéger vos LED avec une résistance d'une valeur comprise entre 200Ω et 1kΩ. Nous verrons dès la semaine prochaine pourquoi!

Merci à Glenn Smith pour ce cours!

I.2.5. Premier programme : Blink

I.2.5.0.1. Le Blink

Voici donc notre premier programme! Cette suite d'instructions va faire clignoter une LED branchée sur la broche 13 de l'Arduino toutes les secondes. Lorsque vous utilisez le logiciel Arduino, il peut être trouvé en cliquant sur *Fichier→Exemples→01.Basics→Blink*.

```
/*
  Clignotement
  Allume la LED pendant 1 seconde,
  puis l'éteint pendant 1 seconde.
*/

// Numéro de la broche à laquelle est
// connectée la LED
int led = 13;

// le code dans cette fonction est exécuté une fois au début
void setup() {
  // indique que la broche de la LED une sortie :
  // on va modifier sa tension
  pinMode(led, OUTPUT);
}

// le code dans cette fonction est exécuté en boucle
void loop() {
  digitalWrite(led, HIGH);   // allumer la LED (tension 5V sur la
    broche)
  delay(1000);               // attendre 1000ms = 1s
  digitalWrite(led, LOW);    // éteindre la LED (tension 0V sur la
    broche)
  delay(1000);               // attendre à nouveau 1seconde
}
```

Listing 2 – Notre premier code, le blink

I.2.5.0.1.1. Instructions À la fin de chaque programme, nous détaillerons les nouvelles briques logicielles utilisées. Comme c'est notre premier programme, nous avons beaucoup de choses à voir (n'hésitez pas à cliquer sur les liens ci-dessous afin d'arriver sur la référence Arduino).
Dans ce programme, nous avons:
Des commentaires : qui sont des lignes de texte incluses dans le programme et qui ont pour but de vous aider à comprendre (ou à vous rappeler) comment votre programme fonctionne ou d'en informer les autres. Ces lignes ne sont pas envoyées à Arduino. Il y a deux façons de créer des lignes de commentaires :

```
/*
  Voici des
  commentaires sur
  plusieurs ligne
*/
```

```
// Ceci est également un commentaire
```

Des instructions:
- **Déclaration** d'une **variable** : on vient avec cette ligne stocker la valeur à droite du signe égal dans la variable à gauche du signe égal.

```
int led = 13;
```

Dans notre cas, cela signifie que la **variable** appelée `led` qui sera un nombre (puisqu'elle est précédée du mot-clé int) viendra prendre la valeur 13.
- **Les blocs d'instructions**:
  - `setup` regroupe toutes les instructions qui seront exécutées au démarrage du programme. La fonction `setup` n'est exécutée qu'une seule fois, après chaque mise sous tension ou reset (réinitialisation) de la carte Arduino.
  - `loop` (boucle en anglais) contient les instructions que l'on souhaite voir exécutées encore et encore tant que l'Arduino est branché.

```
void setup() {
}

void loop() {
}
```

Listing 3 – Le programme minimal

Les **fonctions**: sont des instructions qui permettent d'exécuter une ou plusieurs actions. Les fonctions sont définies avec:
- Un **nom**: ce qu'on devra taper pour appeler la fonction.
- Une ou des **entrées**: ce sont des variables passées à la fonction appelées **paramètres** ou **arguments**. Ces arguments sont placés entre parenthèses.
- **Une sortie**: le résultat de la fonction qui peut être stocké dans une variable.

Prenons l'exemple de la fonction suivante:

```
1 digitalWrite(led, HIGH);
```

Listing 4 – La fonction digitalWrite

Dans ce cas, le nom de la fonction est `digitalWrite` . Nous passons deux paramètres à la fonction: `led` et `HIGH`. La fonction `digitalWrite` n'a pas de sortie. Avec cette fonction, nous allumons la broche située sur la broche passée avec le premier paramètre (qui peut être un nombre ou une variable). Lorsque le second arguement est placé à `HIGH`, on vient allumer la LED. Tandis qu'on éteindra la LED si le second argument passé est `LOW`.
Les autres fonctions présentes dans le programme *Blink* sont:

— `pinMode` configure la broche spécifiée dans le premier paramètre pour qu'elle se comporte soit en entrée (`INPUT`), soit en sortie (`OUTPUT`) passée avec le second paramètre:

```
1 pinMode(led, OUTPUT);
```

— `delay` fait une pause dans l'exécution du programme pour la durée (en millisecondes) passée en paramètre :\

```
1 delay(1000);
```

Voilà, c'est tout pour le code de cette semaine, la suite c'est les QCM et le premier devoir.

I.2.5.0.1.2. Références

— Référence mini du langage Arduino par Xavier Hinault
— Référence officielle du langage Arduino (anglais) par l'équipe d'Arduino
— Compléments sur la programmation Arduino par Eskimon

I.2.5.0.1.3. Licence

FIGURE I.2.6. – CC-BY-SA

Nous profitons de cette section pour remercier Xavier Hinault qui met à disposition sur son site www.mon-club-elec.fr une documentation détaillée et libre sur lequel s'appuie ce texte.

I.2.6. Travaux pratiques

I.2.6.0.1. Exercice à faire pour la semaine 3

On vous montre à quoi ça ressemble.

FIGURE I.2.7. – Animation du résultat attendu

La correction (montage et code) sera donnée la semaine prochaine!
Bon courage!

I.2.7. Quelques outils de base

Parmi la liste des outils incontournables du bricolage, certains sont réellement faciles à trouver dans plein de magasins. En voici une petite liste…

I.2.7.0.0.1. Tournevis Les tournevis! Vous en avez tous déjà vus et sûrement manipulés. Ils sont indispensables et portent bien leurs noms. Aucune excuse pour ne pas en avoir une paire toujours à portée de main. Ils vous seront très pratiques dès qu'il s'agira de démonter quelque chose ou de resserrer un boîtier par exemple.
On en trouve de très nombreux dans les matériaux, les tailles et les formes. Pour faire de l'électronique, je vous conseille d'avoir au moins plusieurs tailles de tournevis plats et quelques tournevis cruciformes.
Si vous faites des montages avec des petits potentiomètres de réglages, vous serez sûrement amenés à utiliser des tournevis en plastique. Leur gros avantage est bien sûr qu'ils ne conduisent pas l'électricité, donc si vous ripez en faisant votre réglage vous ne risquez pas de créer un court-circuit si ce dernier est sous tension!
Il existe aussi des tournevis possédant un indicateur lumineux dans le manche. Ces derniers sont utilisés pour tester la présence d'une tension dans un circuit.
Enfin, il existe aussi des tournevis à embout échangeable. Vous pouvez alors facilement changer de forme/taille d'embout pour vous adapter à diverses situations.

I.2.7.0.0.2. Clés Passons maintenant aux clés. Bien sur je ne parle pas des clefs pour rentrer dans votre atelier, mais bien des clés pour pouvoir serrer les boulons et autres écrous.
Parmi la grande famille de ces dernières, voici quelques unes que l'on retrouve très souvent.

I.2.7.0.0.3. La clé Allen (ou clé 6 pans) Facile à distinguer, c'est une pièce de métal coudée en L qui possède… 6 faces (d'où le nom de clé 6 pans). Elle est utilisée pour serrer des vis possédant une tête creuse de la même forme. Le nom de clé "Allen" vient du nom de la marque éponyme qui a mis en œuvre cette dernière.

I.2.7.0.0.4. La clé Torx (ou clé étoile) Comme pour la clé 6 pans, c'est la caractéristique mécanique qui lui donne son nom. Elle possède une forme en étoile et s'adapte là encore aux vis possédant une tête creuse de la même forme. Cette clé fut inventée par le constructeur automobile Renault.

I.2.7.0.0.5. La clé plate Les clés plates font partie des incontournables. On en trouve de toutes tailles et elles servent à serrer des écrous ou des boulons. Prenez en quelques unes, elles ne coûtent pas une fortune et sont toujours pratiques!

I.2.7.0.0.6. La clé à pipe/douille/cliquet La clé à pipe est similaire à la clé plate à ceci prêt qu'elle a la forme d'une pipe. Sa prise en main facilite le mouvement et donc donne plus de force pour agir sur le boulon/écrou.
La clé à douille est similaire, elle utilise des "douilles" qui ne sont rien d'autres que des embouts interchangeables pour agir sur la visserie.
Enfin, la clé à cliquet est une clé à douille mais possédant un mécanisme particulier lui permettant de faire des mouvements de vissage/dévissage rapide puisque la douille n'a pas à être retirée du boulon/écrou entre chaque mouvement.

FIGURE I.2.8. – Un jeu de clé 6 pans (source: http://upload.wikimedia.org/wikipedia/commons/thumb/c/c3/Allen_keys.jpg/207px-Allen_keys.jpg)

FIGURE I.2.9. – Une clé plate et à œil (source: http://upload.wikimedia.org/wikipedia/commons/thumb/e/e9/Beta_combinata_42.jpg/320px-Beta_combinata_42.jpg)

I.2.7.0.0.7. Pinces Là encore, comme pour les autres outils, il en existe de toutes les formes et toutes les couleurs… Seules certaines d'entre elles vont nous intéresser.

I.2.7.0.0.8. Les pinces plates Son nom l'indique, cette pince est… plate. Elle est souvent utilisée pour serrer deux choses entre elles ou pour maintenir une pièce le temps d'une opération. Par exemple pour refermer un bout de métal ou pour maintenir un fil en train d'être soudé (et ainsi ne pas se brûler).

I.2.7.0.0.9. La pince coupante Là encore le nom est évocateur. Elle va nous servir tout simplement à couper des fils. Cependant n'allez pas couper des fils avec une section (un diamètre) trop grosse sous peine d'endommager l'outil.

I.2.7.0.0.10. La pince à dénuder Toujours avec un nom très clair, celle-ci sert à dénuder vos fils. Autrement dit, elle coupera **proprement** l'isolant plastique sur le bout de votre fil qui pourra ensuite être simplement étamé, soudé ou vissé sur un connecteur. Elle est plutôt facultative dans le sens où le dénudage peut être fait avec une pince coupante, des ciseaux ou un couteau, mais ce sera moins rapide et propre qu'avec cet outil dédié.

I.2.7.0.0.11. La pince à sertir Allant souvent de paire avec la pince à dénuder, cet outil vous permettra de **sertir** vos fils dénudés sur vos connecteurs. Cela vous garantit que votre fil sera bien fixé et maintenu sur sa cosse et ainsi qu'il ne risque pas de s'arracher du connecteur final dès que vous allez tirer un peu dessus. Là encore elle est facultative, cette opération pouvant être réalisée (de manière moins triviale) avec une pince plate et de la patience. Son coût peut aussi être un frein à l'achat…

I.2.7.0.0.12. La pince brucelles Cette dernière est toute simple et a une application bien particulière. Elle ressemble un peu à une pince à épiler à la différence qu'elle se termine en pointe et ne coupe pas. On s'en sert pour manipuler des composants montés en surface (CMS) qui sont difficiles à prendre à la main et à positionner correctement.

FIGURE I.2.10. – Divers pinces (à bec, coupante, plate…) (source: http://commons.wikimedia.org/wiki/File:Work_shop_clamps.JPG)

I.2.7.0.0.13. D'autres outils pratiques Une fois votre caisse à outils équipée avec les outils minimum de tout bon bricoleur (le kit de survie), nous pouvons commencer à en rajouter quelques uns, pour du confort ou de la sécurité.

I.2.7.0.0.14. Sécurité Commençons par la sécurité puisque c'est important!
Si vous commencez à fabriquer vos propres cartes électroniques, vous allez manipuler des produits toxiques et corrosifs. Le minimum est donc de prendre une paire de gants afin d'éviter les éventuelles éclaboussures avec la peau (qui pourraient vous laisser des tâches par exemple, voire bien pire).
Qui dit gant, dit aussi lunettes (car les éclaboussures ne sont pas difficiles, elles se contenteront de vous attaquer là où c'est possible). 🍊
Les lunettes sont aussi très pratiques quand il s'agit de faire des perçages (un copeau dans l'œil est vite arrivé).
D'une manière générale: Protégez-vous! N'oubliez pas ce que nous dit la loi de Murphy (ou loi de l'emmerdement maximal): c'est la seule fois où vous ne porterez pas de protection que l'accident arrivera!

I.2.7.0.0.15. Confort et outils++

I.2.7.0.0.16. Se simplifier la vie Il arrive souvent que l'on ait des tâches un peu fastidieuses à faire à la main. Par exemple étamer un simple fil (action qui consiste à recouvrir d'étain le

cuivre d'un fil). Pour cela il vous faut tenir le fil, tenir l'étain et manipuler le fer à souder pour déposer l'étain sur le fil. Nul doute qu'une troisième main serait fort pratique! C'est justement le nom de l'outil suivant. La troisième main se présente sous la forme d'un support avec une pince ou deux et éventuellement une loupe. Le support est leste pour bien rester en place lorsque vous manipulez. Grâce aux pinces vous pouvez facilement placer des composants dans l'orientation qui vous arrange le plus pour faire vos bricoles!
Dans le même ordre d'idée on retrouve l'étau. Cet ustensile est plus dédié au serrage ou au maintien bien en place pendant un usinage. Placez une pièce entre les mâchoires, serrez, c'est maintenu! Vous pouvez alors laisser le tout en place (pour coller deux pièces par exemple) ou commencer à percer/limer/etc. de manière stable, propre et sécurisée. Il existe différents moyens de fixer l'étau à votre établi. Certains sont par un système de ventouse, d'autres se fixent par une vis de serrage.

I.2.7.0.0.17. Outils d'usinage Pour faire des ajustements mécaniques sur des pièces, il est utile d'avoir de quoi usiner. Parmi ces outils on retrouve la perceuse (portative ou colonne) qui vous permettra de trouer à volonté, la fraiseuse pour faire des ajustements de surface ou encore une bonne vieille lime pour ébavurer des découpes ou arrondir des angles un peu trop francs et dangereux.

Rappel sécurité, lorsque vous utilisez des outils électro-portatifs (comme une perceuse), veillez à toujours bien bloquer la pièce qui va recevoir le traitement. Vous ne souhaitez pas la voir commencer à tourner et risquer de partir dans le décor. De même, des lunettes pour éviter les copeaux dans les yeux sont les bienvenues!

I.2.8. Des outils spécifiques à l'électronique

I.2.8.0.1. Faire son montage et le tester

Pour faire un montage rapide ou de prototypage pour tester quelque chose, nous l'avons vu la *breadboard* reste un choix idéal. Il est facile d'y enficher des composants pour faire des tests.
Afin d'optimiser le tout, une bobine de fil mono-brin est souvent plus simple que le multibrins pour rentrer dans les trous de la *breadboard*. Il existe des kits de fils pour *breadboard* de tailles et couleurs différentes pour faire des montages propres. Les plus bricoleurs d'entre vous en ont cependant sûrement déjà vus, c'est le même type de fil que l'on a dans nos murs pour transporter le signal téléphonique!
Une fois votre montage fait vous allez devoir le tester pour trouver où l'électronique ne fonctionne pas. L'utilisation d'un multimètre permettant de mesurer la tension, l'intensité ou encore possédant des fonctions de testeur de diodes et de mesure de résistances est rapidement indispensable. Ils ne coûtent pas cher et se trouvent facilement un peu partout sans avoir besoin de rentrer dans des magasins spécialisés.
Enfin, lorsque votre montage est terminé et fonctionnel vous pourriez avoir envie de l'embarquer sur une carte électronique pour fixer les composants. Si vous ne pouvez pas réaliser vos propres cartes et/ou ne souhaitez pas dépenser des fortunes pour les réaliser, vous pouvez toujours souder vos composants sur des "plaques à trous" (aussi appelées *veroboard*).

I.2.8.0.2. La soudure

Puisqu'on parle de soudure voyons un peu tout ce qui y touche… Bien entendu, tout commence par le fer. Pour des travaux d'électronique, un fer d'une trentaine de watts sera suffisant. Il doit être capable de chauffer entre 300 et 400°C si vous voulez que l'étain fonde. Bien entendu, il en existe pour tous les prix, allant du simple fer à 10€ à la station réglable qui en coûtera plus d'une centaine. Choisissez en fonction de vos moyens et de vos prétentions mais gardez en tête que les moins chers vont chauffer lentement ou atteindre difficilement une température suffisante. De manière générale, essayez d'éviter les fers de type "pistolets".
Le fer à souder ne sert pas à grand chose s'il est seul, il vous faudra aussi investir dans une bobine d'étain (fil à souder) qui peut contenir ou non du plomb (plus simple à souder mais à utiliser dans un endroit aéré car les vapeurs ne sont pas très bonnes pour l'organisme).
De la tresse à dessouder ou une pompe à dessouder peuvent aussi s'avérer utiles.

I.3. Semaine 3 : Capteurs numériques

Introduction

Arduino et les capteurs La **semaine 3** sera l'occasion de poursuivre notre découverte d'Arduino en lui branchant des capteurs afin qu'il sache ce qui se passe autour de lui. Nous prendrons l'exemple d'un bouton poussoir pour illustrer ce qu'est un capteur numérique avant de nous attaquer la semaine prochaine aux capteurs analogiques.
Nous continuons aussi le cours sur le Prototypage et l'électricité avec Glenn et pour aller plus loin, Eskimon nous a concocté un cours sur la détection d'un front!
Supports pdf: Certaines personnes nous ont demandé sur le forum de mettre à disposition la transcription de la vidéo. C'est fait! Vous pouvez le télécharger ici , sinon, il est disponible après la vidéo.
Quiz: Comme la semaine dernière, vous aurez la possibilité de répondre à un quiz et de réaliser un TP.
Forum: Sur le forum, nous vous conseillons de poser vos questions grâce aux mots clef `[Tuto fablab][Semaine 3]`. Voici quelques exemples:

- [Tuto fablab] Questions sur l'organisation
- [Tuto fablab] Les capteurs
- [Tuto fablab] Questions sur les résistances
- [Tuto fablab] TP feux piétons

Bonne semaine,
L'équipe du MOOC

I.3.1. Corrigé du TP 1

I.3.1.0.1. Corrigé du TP Feu Tricolore

Voici la correction du TP de la semaine dernière et on commence avec le code:

I.3.1.0.1.1. Code

```
/*
  Feu tricolore

 TP de la semaine 2 du MOOC "La Fabrication Numerique"
 Allume trois LED comme suit :
 Orange allumee pendant 1 seconde
 Rouge allumee pendant 3 secondes
 Verte allumee pendant 3 secondes

 Le montage :
```

```
 * Une LED rouge sur la broche 2 en serie avec une resistance de 1K
 * Une LED orange sur la broche 3 en serie avec une resistance de 1K
 * Une LED verte sur la broche 4 en serie avec une resistance de 1K

 cree le 24 Mars 2014
 par Baptiste Gaultier

 Ce code est en CC0 1.0 Universal

 https://www.france-universite-numerique-mooc.fr/courses/MinesTelecom/04002/Trimestre_1_2014/about
 */

int rouge = 2;
int orange = 3;
int verte = 4;

// le code dans cette fonction est exécuté une fois au début
void setup() {
  // indique que les broches des LED
  // sont des sorties :
  pinMode(rouge, OUTPUT);
  pinMode(orange, OUTPUT);
  pinMode(verte, OUTPUT);
}

// le code dans cette fonction est exécuté en boucle
void loop() {
  digitalWrite(orange, HIGH);
  delay(1000);
  digitalWrite(orange, LOW);

  digitalWrite(rouge, HIGH);
  delay(3000);
  digitalWrite(rouge, LOW);

  digitalWrite(verte, HIGH);
  delay(3000);
  digitalWrite(verte, LOW);
}
```

Vous pouvez également télécharger le programme (appelé également *croquis* dans le logiciel Arduino) en cliquant ici et en l'ouvrant avec le simulateur ou le logiciel Arduino.

I.3.1.0.1.2. Montage électronique

FIGURE I.3.1. – Montage correction TP feu tricolore

Pour réaliser ce montage, vous avez besoin de:
- Un Arduino
- Une platine de prototypage
- Un câble USB
- Trois résitances de 220Ω
- Des fils de prototypage
- Une LED verte
- Une LED orange
- Une LED rouge
- Un peu de votre temps

C'est tout pour ce premier TP! On vous invite à poser vos questions sur le forum et à commencer à réfléchir sur le TP de cette semaine.

I.3.2. Les capteurs numériques

ÉLÉMENT EXTERNE (VIDEO) —
Consultez cet élément à l'adresse https://geo.dailymotion.com/player.html?video=x2kgzqn&.

Vous pouvez télécharger le pdf de la vidéo ici .

I.3.2.0.1. Le button

Merci d'avoir regardé cette vidéo! Voici quelques informations pour mieux la comprendre:

I.3.2.0.1.1. Montage

FIGURE I.3.2. – Montage bouton

Pour le réaliser, vous aurez besoin de:

— Un Arduino
— Un câble USB
— Deux résitances de 1KΩ
— Des fils de prototypage
— Une platine de prototypage
— Un bouton poussoir
— Une LED de votre couleur préférée

I.3.2.0.1.2. Code Cette suite d'instructions va allumer une LED branchée sur la broche 13 lorsque le bouton branché sur la broche 2 est appuyé. Lorsque vous utilisez le logiciel Arduino, le code peut être trouvé en cliquant sur *Fichier→Exemples→02.Digital→Button*.

```
1  /*
2    Bouton
3
4   Allume une LED branchée sur la broche 13 lorsque le bouton
5   branché sur la broche 2 est appuyé.
6   */
7
8  // Initialisation des constantes :
9  const int buttonPin = 2;     // Numéro de la broche à laquelle est
       connecté le bouton poussoir
10 const int ledPin =  13;      // Numéro de la broche à laquelle est
       connectée la LED
11
12 // Déclaration des variables :
13 int buttonState = 0;         // variable qui sera utilisée pour
       stocker l'état du bouton
14
15 // le code dans cette fonction est exécuté une fois au début
16 void setup() {
17   // indique que la broche ledPin est une sortie :
18   pinMode(ledPin, OUTPUT);
19   // indique que la broche ledPin est une entrée :
20   pinMode(buttonPin, INPUT);
21 }
22
23 // le code dans cette fonction est exécuté en boucle
24 void loop(){
25   // lit l'état du bouton et stocke le résultat  // dans
       buttonState :
26   buttonState = digitalRead(buttonPin);
27
28   // Si buttonState est à 5V (HIGH→bouton appuyé)
29   if (buttonState == HIGH) {
30     // on allume la LED
31     digitalWrite(ledPin, HIGH);
32   }
33   else {
```

```
34      // sinon on éteint
35      digitalWrite(ledPin, LOW);
36    }
37  }
```

Remarques: Copiez-collez ce code dans le simulateur pour ne pas avoir à tout retaper. Saviez vous que vous pouvez accéder à la documentation d'une fonction en cliquant avec le bouton droit sur celle-ci puis en cliquant sur *Trouvez dans la référence*.

I.3.2.0.1.3. Instructions Comme la semaine dernière, voici une description des nouvelles fonctions utilisées (n'hésitez pas à cliquer sur les liens ci-dessous afin d'arriver sur la référence Arduino).

— **Déclaration d'une constante:** comme pour une variable , on vient avec cette ligne stocker la valeur à droite du signe égal dans `led`.

```
1  const int led = 13;
```

Listing 5 – déclaration d'une constante

Le mot clé `const` indique que l'on ne souhiate pas que la valeur de led puisse être modifiée dans le programme.

— **Les nouvelles instructions**:

— `digitalRead` lit l'état d'une broche et renvoie la valeur `HIGH` si la broche est à la tension de l'alimentation ou `LOW` si la broche est à 0V.

```
1  digitalRead(buttonPin);
```

Listing 6 – lecture numérique

La valeur de retour de `digitalRead` peut être stockée dans une variable comme ceci:

```
1  buttonState = digitalRead(buttonPin);
```

— `if` permet de tester si une expression située entre parenthèse est vraie. Dans *Button*, nous cherchons à savoir si le bouton est appuyé, nous allons donc comparer buttonState à `HIGH` comme ceci:

```
1  if(buttonState == HIGH)
```

Listing 7 – Votre première condition

— `else` : le bloc situé après cet mot clé viendra être executé si le test précédent échoue. Dans *Button*, si le bouton **n'est pas** appuyé alors on viendra éteindre la LED.

I.3.2.0.1.4. Références

— Référence mini du langage Arduino par Xavier Hinault

— Référence officielle du langage Arduino (anglais) par l'équipe d'Arduino

— Compléments sur la programmation Arduino par Eskimon et olyte

I.3.2.0.1.5. Licence

FIGURE I.3.3. – CC-BY-SA

Nous profitons de cette section pour remercier Xavier Hinault qui met à disposition sur son site www.mon-club-elec.fr une documentation détaillée et libre sur lequel s'appuie ce texte.

I.3.3. Prototypage et éléctricité

La semaine passée, nous indiquions ici qu'il fallait protéger vos LED avec une résistance comprise entre 200Ω et $1k\Omega$.
Cette semaine, on s'intéresse au pourquoi avec un montage:

I.3.3.0.0.1. Montage

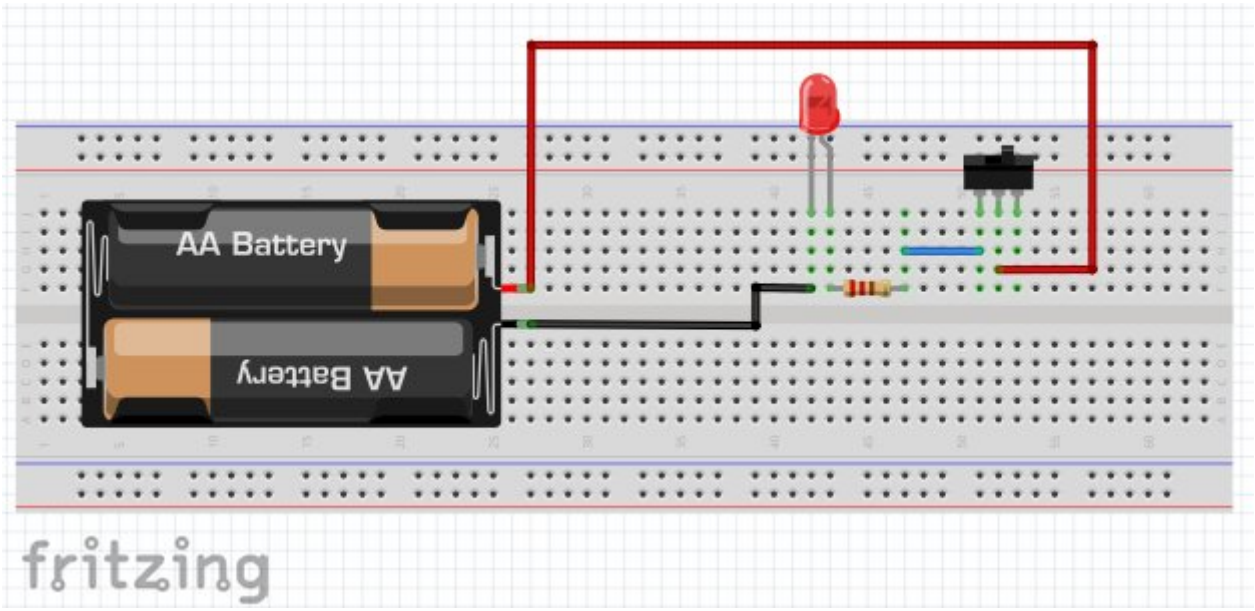


FIGURE I.3.4. – Le montage

Le montage suivant utilise une diode électroluminescente ou LED, un interrupteur et une résistance. Quand l'interrupteur est fermé, le courant passe à travers la résistance et la diode - et la diode va s'allumer. Ce même montage peut être dessiné en tant que schéma électronique comme suit (interrupteur dessiné en position fermée):

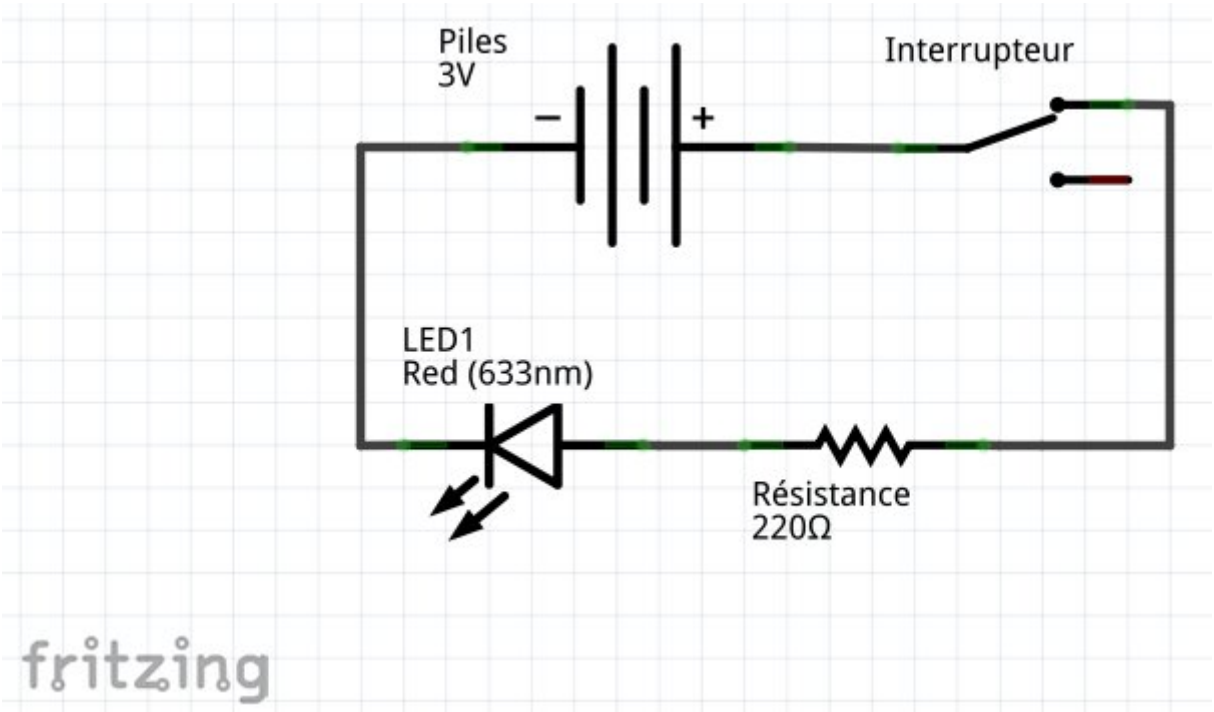


FIGURE I.3.5. – Schéma du circuit

Si on regarde la documentation pour les LED standards, on voit qu'une LED a besoin d'une tension entre 1,5v et 3,5v pour fonctionner (cela est fonction de la couleur de la LED, ainsi que de son fabricant), mais il ne faut pas dépasser quelques dizaines de milliampères (écrits mA: millième d'un ampère) sinon la LED peut être endommagée. C'est pour cela que la résistance est mise dans ce circuit – pour limiter l'intensité à travers la LED.
Pour calculer la bonne valeur de résistance pour que la LED s'allume au maximum, sans être endommagée, on va se servir de la formule :

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

dont nous parlions ici .
Dans la suite de cette discussion, je vais prendre 2,2V comme chute de tension (valeur choisie arbitrairement).
La tension de ce circuit fournie par les deux piles, donnent 3V. La chute de tension à travers la résistance est donc $3V–2.2V = 0.8V$. Souvent on limite l'intensité à travers une LED à 20mA (= 0,02 A). Cela donne, donc $R = \frac{0.8V}{0.02}$. Le résultat est qu'il est possible de réduire la résistance jusqu'à 40Ω sans endommager la LED.
Dans le cas d'une LED branchée sur une broche de l'Arduino, nous avons du 5V à la sortie de la broche. La chute de tension au travers de la résistance est de $5V–2.2V = 2.8V$. Ça nous donne donc $R = \frac{2.8V}{0.02A} = 140\Omega$.

I.3.3.0.0.2. Valeurs de résistances Les résistances sont fabriquées et triées dans plusieurs gammes de valeurs normalisées. On ne trouve pas, par exemple, une résistance de 40Ω dans le commerce. La valeur la plus proche est, en fait, 39Ω. En plus, la valeur d'une résistance n'est que très rarement exactement la valeur indiquée: il y a toujours une tolérance (normalement 5%). Une tolérance de 5% sur 39Ω donne de 37,05Ω à 40,95Ω! Si vous cherchez sur internet vous trouverez les infos sur les gammes de valeurs, et comment lire les valeurs (code de couleurs) des résistances.

I.3.3.0.0.3. Pour plus d'infos... Vous pouvez lire les articles dans notre wiki sur:
- les mesures de tension et intensité
- les résistances
- le branchement des LEDs

I.3.3.0.0.4. Références
- Le Wiki des Petits Débrouillards où l'on retrouve plein d'infos (nottament sur l'utilisation des LED).

Merci à Glenn Smith pour ce cours!

I.3.4. Travaux pratiques

Le montage à réaliser pour le TP ci-dessous:

FIGURE I.3.6. – TP de la semaine 3

Seules les notions abordées dans les cours de cette semaine et des semaines passées sont nécessaires pour réaliser ce TP
Bon courage!

I.3.5. Détection d'un front

Comme nous l'avons vu cette semaine, détecter l'appui d'un bouton se fait avec la fonction `digitalRead()`. Cependant, si nous mettons juste cette lecture d'état dans la boucle principale, nous pouvons avoir des surprises.
En effet, prenons le code suivant:

```
1  int monBouton = 2; // bouton en broche 2
2  int compteur = 0;  // un compteur
3  int etatBouton; // L'état du bouton
4
5  void setup() {
6    // le bouton en entrée
7    pinMode(monBouton, INPUT);
8  }
9
10 void loop() {
11   // si on appuie sur le bouton
12   etatBouton = digitalRead(monBouton);
13   if(etatBouton == HIGH) {
14     // alors on incrémente le compteur
15     compteur = compteur+1;
```

```
  }
}
```

Dans ce code, nous incrémentons un compteur de 1 à chaque fois que l'on appuie sur un bouton. Tout du moins c'est ce que l'on aimerait… En effet, il y a un problème! La fonction loop fonctionne très vite, tellement vite que votre compteur va s'incrémenter plein de fois par seconde, aussi longtemps que votre doigt reste appuyé sur le bouton, là ou nous souhaiterions qu'il ne s'incrémente qu'une seule fois par appui…
Ce problème possède une solution: **La détection de front**.
Un front peut aussi être vu comme un changement d'état. Lorsque l'on appuie, on fera passer le niveau électrique de 0 à 1 ou le contraire. C'est ça un front. Le but est maintenant de le détecter afin d'incrémenter notre compteur en sa présence et non pas uniquement sur un état. Pour cela nous allons devoir utiliser une variable nous servant de mémoire et qui va retranscrire l'état du bouton au moment *t-1* (lors du précédent passage dans la boucle si vous préférez).
Ensuite il nous suffira de comparer cette mémoire avec l'état actuel du bouton pour voir si ce dernier a changé entre les deux passages dans la boucle. Si effectivement la mémoire est différente de l'état actuel, alors le bouton a changé d'état, il y a un front. Sinon, cela signifie que le bouton est dans le même état, il n'y a pas eu de front.
Voici ce que cela pourrait donner sous la forme d'un code:

```
int monBouton = 2; // bouton en broche 2
int compteur = 0;  // un compteur
int etatBouton; // L'état du bouton
int memoire = LOW; // La mémoire de l'état du bouton

void setup() {
  // le bouton en entrée
  pinMode(monBouton, INPUT);
}

void loop()
{
    // lecture de l'état du bouton
    etatBouton = digitalRead(monBouton);

    // Si le bouton a un état différent de celui enregistré ET
    // que cet état est "haut"
    if((etatBouton != memoire) && (etatBouton == HIGH))
    {
        // on incrémente le compteur
        compteur++;
    }

    // on enregistre l'état du bouton pour le tour suivant
    memoire = etatBouton;
}
```

I.4. Semaine 4 : Capteurs analogiques

Introduction

Parce que nous vivons dans un monde analogique Même si notre monde est de plus en plus numérique, les grandeurs physiques qui nous entourent sont très majoritairement analogiques. En effet, lorsque vous regardez par la fenêtre (ce que nous vous encourageons à faire entre deux exercices) vous pouvez dire s'il fait jour ou nuit.
Mais vous pouvez également dire si le ciel est très lumineux, légèrement sombre ou si c'est carrément nuit noire.
L'intérêt des capteurs analogiques est là: pouvoir distinguer tout un panel d'états et les convertir en une valeur numérique exploitable dans nos programmes.
Pour illustrer cela, nous allons nous intéresser dans cette **semaine 4** au capteur de luminosité (aussi appelé photorésistance) qui permettra à Arduino de connaître la luminosité autour de lui.
Nous verrons également comment Arduino peut dialoguer avec l'ordinateur grâce au moniteur série.
Enfin, nous continuons aussi le cours sur le Prototypage et l'électricité avec Glenn.
Supports PDF: Pour ceux qui ont besoin du support écrit de la vidéo, vous pouvez le télécharger ici .
Forum: Sur le forum, nous vous conseillons de poser vos questions grâce aux mots clef `[Tuto fablab][Semaine 4]`.
Voici quelques exemples:

- [Tuto fablab] Les capteurs analogiques
- [Tuto fablab] Questions sur les LED
- [Tuto fablab] TP Thérémine lumineux

Bonne semaine,
L'équipe du MOOC

I.4.1. Corrigé du TP 2

I.4.1.0.1. Corrigé du TP Feu Tricolore+Feu piéton

Voici la correction du TP de la semaine dernière qui reprend des éléments du TP 1: Feu tricolore ainsi que des éléments du cours 2 sur les capteurs numériques.
Nous profitons de cette page pour remercier les personnes qui nous ont alerté que l'intitulé de ce TP laissait planer quelques doutes. Merci beaucoup!
Pour les prochains travaux pratiques et exercices où les consignes vous semblent floues, n'hésitez pas à appliquer les conseils suivants:

1. Avertir la communauté et l'équipe pédagogique sur le forum dans le fil de discussion dédié.
2. Prenez toujours les hypothèses qui vous arrangent. 🍊

3. Seules les notions abordées dans le cours et les annexes sont nécessaires pour mener à bien les TPs.
4. Il n'y a pas une mais plusieurs solutions à chaque problème. La meilleure est celle que vous comprenez!

I.4.1.0.1.1. Code Voici une des solutions possibles pour répondre au problème dans l'état actuel de nos connaissances.
Sachez cependant que ce code n'est pas parfait car l'appui sur le bouton n'est détecté que si l'on appuie pendant tout le feu vert voiture. Dans quelques semaines, nous verrons comment être plus réactif.

Contenu masqué n°1

Vous pouvez également télécharger le programme (appelé également *croquis* dans le logiciel Arduino) en cliquant ici et en l'ouvrant avec le simulateur ou le logiciel Arduino.

I.4.1.0.1.2. Montage électronique

FIGURE I.4.1. – Montage du TP "Feu tricolore"

Pour réaliser ce montage, vous avez besoin de:
— Un Arduino
— Une platine de prototypage
— Un câble USB
— Cinq résitances de 220Ω
— Une résistance de 10kΩ
— Des fils de prototypage
— Deux LED verte
— Une LED orange
— Deux LED rouge
— Un bouton poussoir
— Du temps

Notre correction s'arrête là. Pour des éclaircissements, nous vous invitons à poser vos questions sur le forum et à reparcourir le cours.
On vous laisse avec un montage réalisé par un des membres, ProPhil, qui nous a épaté avec cette photo :

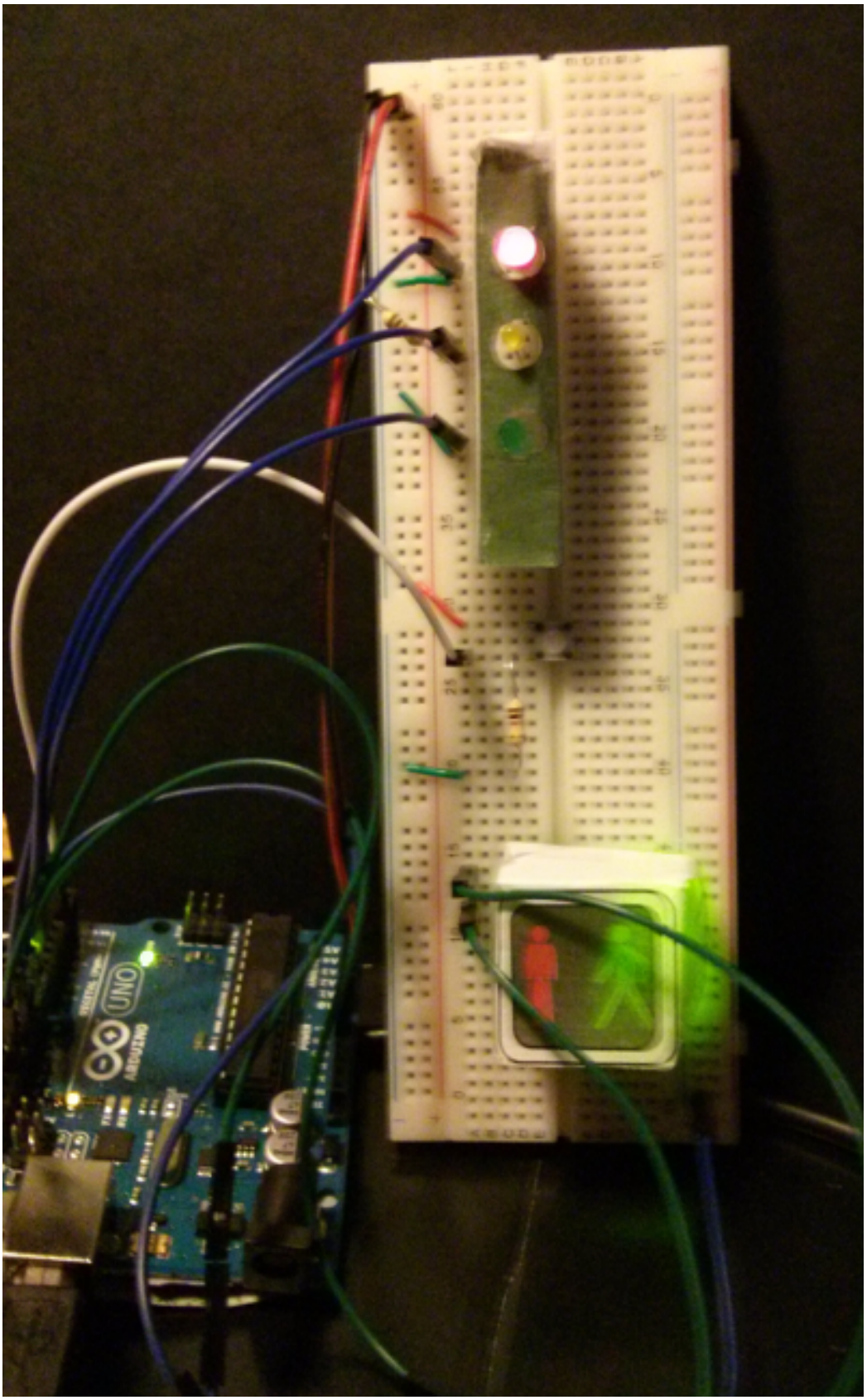


FIGURE I.4.2. – Exemple de réalisation

I.4.1.0.1.3. Schéma électronique Comme on nous l'a demandé sur le forum, voici le schéma électronique créé avec Fritzing :

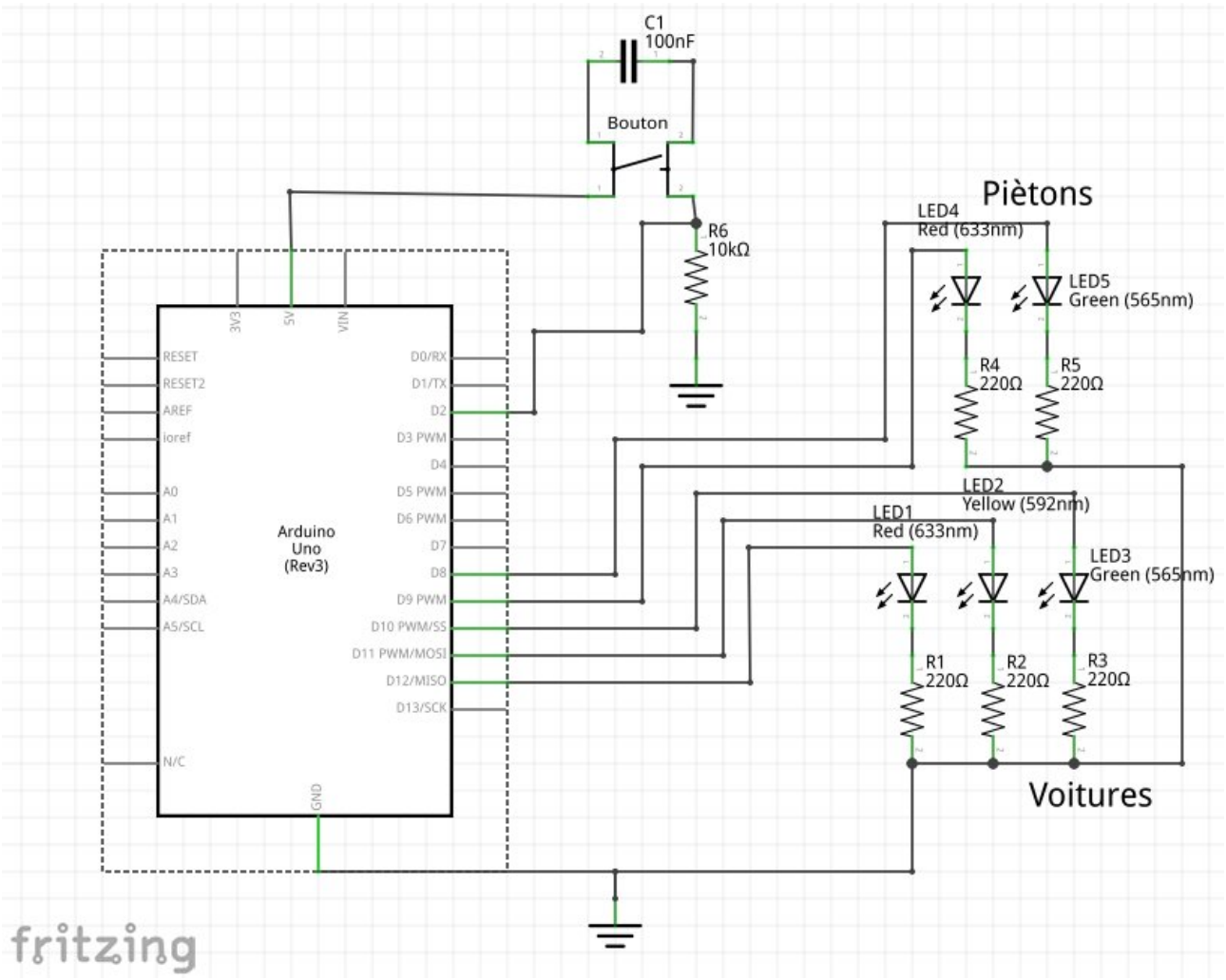


FIGURE I.4.3. – Schéma de Fritzing

I.4.2. Capteurs analogiques

ÉLÉMENT EXTERNE (VIDEO) —
Consultez cet élément à l'adresse https://geo.dailymotion.com/player.html?video=x2kgzq8&.

Vous pouvez télécharger le transcript de la vidéo ici .

I.4.2.0.1. analogInOutSerial

Merci d'avoir regardé cette vidéo! Voici quelques éléments pour mieux la comprendre:

I.4.2.0.1.1. Montage

FIGURE I.4.4. – Montage de la photorésistance

I.4.2.0.1.2. Schéma

FIGURE I.4.5. – Schéma avec la photorésistance

Pour le réaliser, vous aurez besoin de:
- Un Arduino
- Un câble USB
- Deux résitances de 1kΩ
- Des fils de prototypage
- Une platine de prototypage
- Une photorésistance
- Une LED de votre couleur préférée

I.4.2.0.1.3. Code Cette suite d'instructions va allumer une LED branchée sur la broche `9`. L'intensité lumineuse de la LED sera proportionnelle à la luminosité captée par la photorésistance branchée sur la broche `A0` (notez bien le `A0`, le **A** qui précède le 0 signifie que c'est une entrée **A**nalogique).
Lorsque vous utilisez le logiciel Arduino, le code peut être trouvé en cliquant sur *Fichier→Exemples→03.Analog→AnalogInOutSerial*.

```
/*
  Entrée et sortie analogiques + communications série

 Ce programme va allumer une LED branchée sur la broche 9.
 L'intensité lumineuse de la LED sera proportionnelle à la luminosité
 captée par la photorésistance branchée sur la broche A0.

 */

// Initialisation des constantes :
// Numéro de la broche à laquelle est connecté la photorésistance
const int analogInPin = A0;
// Numéro de la broche à laquelle est connectée la LED
```

```
14 const int analogOutPin = 9;
15
16 // Valeur lue sur la photorésistance
17 int sensorValue = 0;
18 // Valeur envoyée à la LED
19 int outputValue = 0;
20
21 void setup() {
22   // Initialise la communication avec l'ordinateur
23   Serial.begin(9600);
24
25   // Indique que la broche analogOutPin est une sortie :
26   pinMode(analogOutPin, OUTPUT);
27   // Indique que la broche analogInPin est une entrée :
28   pinMode(analogInPin, INPUT);
29 }
30
31 void loop() {
32   // lit la valeur de la photorésistance et
33   // stocke le résultat dans sensorValue :
34   sensorValue = analogRead(analogInPin);
35   // change sensorValue vers une intervalle de 0 à 255
36   // et stocke le résultat dans outputValue :
37   outputValue = map(sensorValue, 0, 1023, 0, 255);
38   // envoie de cette nouvelle valeur sur la LED
39   analogWrite(analogOutPin, outputValue);
40
41   // envoie tout ça vers l'ordinateur
42   Serial.print("sensor = " );
43   Serial.print(sensorValue);
44   Serial.print("\t output = ");
45   Serial.println(outputValue);
46 }
```

Listing 8 – Lecture de la photorésistance

Remarques:

— Copiez-collez ce code dans le simulateur pour ne pas avoir à tout retaper. Saviez vous que vous pouvez accéder à la documentation d'une fonction en cliquant avec le bouton droit sur celle-ci puis en cliquant sur *Trouvez dans la référence*.

— Aussi, il faut bien attendre la fin du téléversement avant d'ouvrir le moniteur série dont on parle en dessous.

I.4.2.0.1.4. Instructions Comme les semaines passées, voici une description des nouvelles fonctions utilisées (n'hésitez pas à cliquer sur les liens ci-dessous afin d'arriver sur la référence Arduino).

Les nouvelles instructions:

— `analogRead()` permet de lire l'état d'une broche analogique et de renvoyer une valeur numérique proportionnelle à la tension reçue. La carte Arduino comporte 6 voies, `A0` à `A5`, connectées à un convertisseur analogique-numérique 10 bits. Cela signifie qu'il

est possible de transformer la tension d'entrée entre 0 et 5V en une valeur numérique entière comprise entre 0 et 1023.

```
1 analogRead(analogInPin);
```

La valeur de retour de `analogRead()` peut être stockée dans une variable entière (c'est pour cela que nous devons déclarer une variable de type `int` pour stocker le résultat):

```
1 sensorValue = analogRead(analogInPin);
```

— **`Serial`** est une librairie (un ensemble de fonctions) utilisée pour les communications par le port série entre la carte Arduino et un ordinateur ou d'autres composants. Ce port série permet l'envoie et la réception de suites de caractères sur les broches 0 (RX) et 1 (TX) avec l'ordinateur via le port USB. C'est pourquoi, si vous utilisez cette fonctionnalité, vous ne pouvez utiliser les broches 0 et 1 en tant qu'entrées ou sorties numériques. Si vous souhaitez visualiser le texte envoyé depuis l'Arduino vers votre ordinateur, vous pouvez utiliser le terminal série intégré à l'environnement Arduino. Il suffit pour cela de cliquer sur le bouton du moniteur série dans la barre d'outils (**Rappel:** il faut bien attendre la fin du téléversement avant d'ouvrir le moniteur série):

FIGURE I.4.6. – Bouton du moniteur série

— Dans la fenêtre qui s'ouvre, vérifier bien que vous êtes au même débit de communication que celui utilisé dans l'appel de la fonction `Serial.begin`. Par défaut le débit est de 9600:

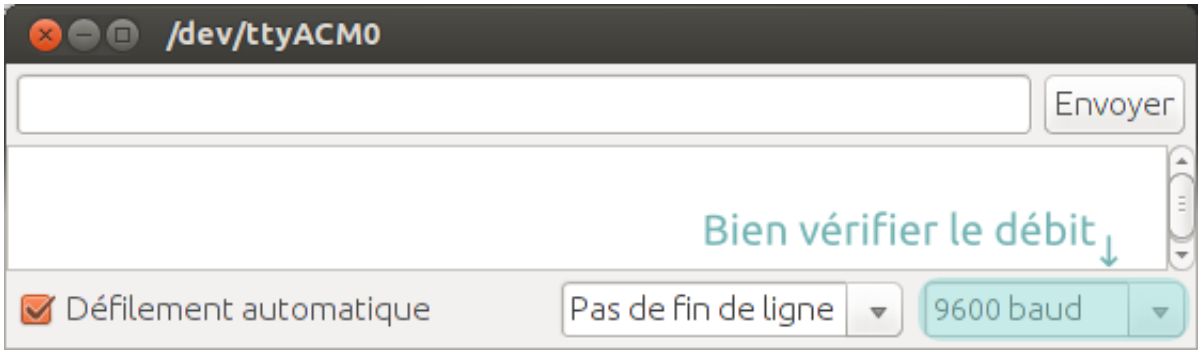


FIGURE I.4.7. – Sélecteur du débit de la communication

Les fonctions de la librairie **`Serial`** sont:

— `Serial.begin` permet d'initialiser la communication entre Arduino et votre ordinateur. Cette fonction doit être placée dans le bloc **`setup`**, elle doit être suivi d'un seul paramètre qui correpond au débit de communication en nombre de caractères échangés par seconde (l'unité est le baud) pour la communication série. De manière classique, nous choisirons de communiquer à 9600 bauds. Pour en savoir plus sur l'utilisation du port série et des débits possibles, cliquez ici. Sans le `Serial.begin` dans le bloc **`setup`**, on ne peut utiliser les autres fonctions `Serial.print` et `Serial.println`.

```
Serial.begin(9600);
```

— `Serial.print` Cette fonction permet d'envoyer sur le port série une suite de caractères indiqué en paramètre. De la même façon, `Serial.println` envoie une suite de caractères suivi d'un retour à la ligne:

```
Serial.print("Texte envoye vers l'ordinateur"); // sans retour à
    la ligne
Serial.println("Texte avec retour a la ligne"); // avec retour à
    la ligne
```

— `Serial.print` nous sera particulièrement utile pour une tâche que vous avez déjà faite sans vous en rendre compte: **le débogage**. Il n'est pas rare que vous téléversiez votre code sur Arduino sans problème, mais une fois sur Arduino le comportement de votre programme mais pas celui attendu. Des petites erreurs viendront souvent se glisser dans vos programmes car nous ne sommes malheureusement pas parfait et qu'un oubli est vite arrivé. Grâce à **`Serial`** , il nous sera possible d'indiquer quand nous allumons une LED ou lorsque nous faisons un test… Il nous sera ainsi possible de suivre le déroulement de notre programme! Mais ce n'est pas tout, `Serial.print` et `Serial.println` peuvent également afficher la valeur de variables si nous indiquons un nom de variable en paramètre:

```
Serial.print("sensor = " );
Serial.print(sensorValue);
```

— `map()` permet de faire passer une valeur située dans une intervalle vers un autre. Les paramètres de ces fonctions sont les suivants:

```
-   variable qui se trouve dans l'intervalle initiale
-   début de l'intervalle initiale
-   fin de l'intervalle initiale
-   début de l'intervalle visée
```

Grâce à cette fonction, nous allons donc nous retrouver avec une valeur proportion

```
map(sensorValue, 0, 1023, 0, 255); // sensorValue passe de
    l'intervalle 0→1023 vers 0→255
```

— `analogWrite()` va être utilisé dans ce programme pour moduler l'intensité lumineuse d'une LED branché sur la broche spécifiée avec le premier paramètre. Dans les semaines passées, nos LED étaient allumées ou éteintes. L'intérêt de `analogWrite` avec est de pouvoir régler l'intensité lumineuse en spécifiant un nombre (compris entre 0 et 255) dans le second paramètre de la fonction:

```
analogWrite(11, 0);  // éteint complètement la LED branché sur la
    broche 11
analogWrite(11, 90); // allume un tout petit peu la LED branché
    sur la broche 11
analogWrite(11, 255);// allume complètement la LED branché sur la
    broche 11
```

I.4.2.0.1.5. Simulateur Pur ceux qui souhaiteraient utiliser le simulateur, la photorésistance est disponible en cliquant sur *Add component* puis sur *LDR*. Cependant, l'interaction avec la photorésistance est limité et se limite à cliquer dessus pour changer la valeur de sa résistance. Nous vous conseillons de le remplacer par un potentiomètre comme ci-dessous:

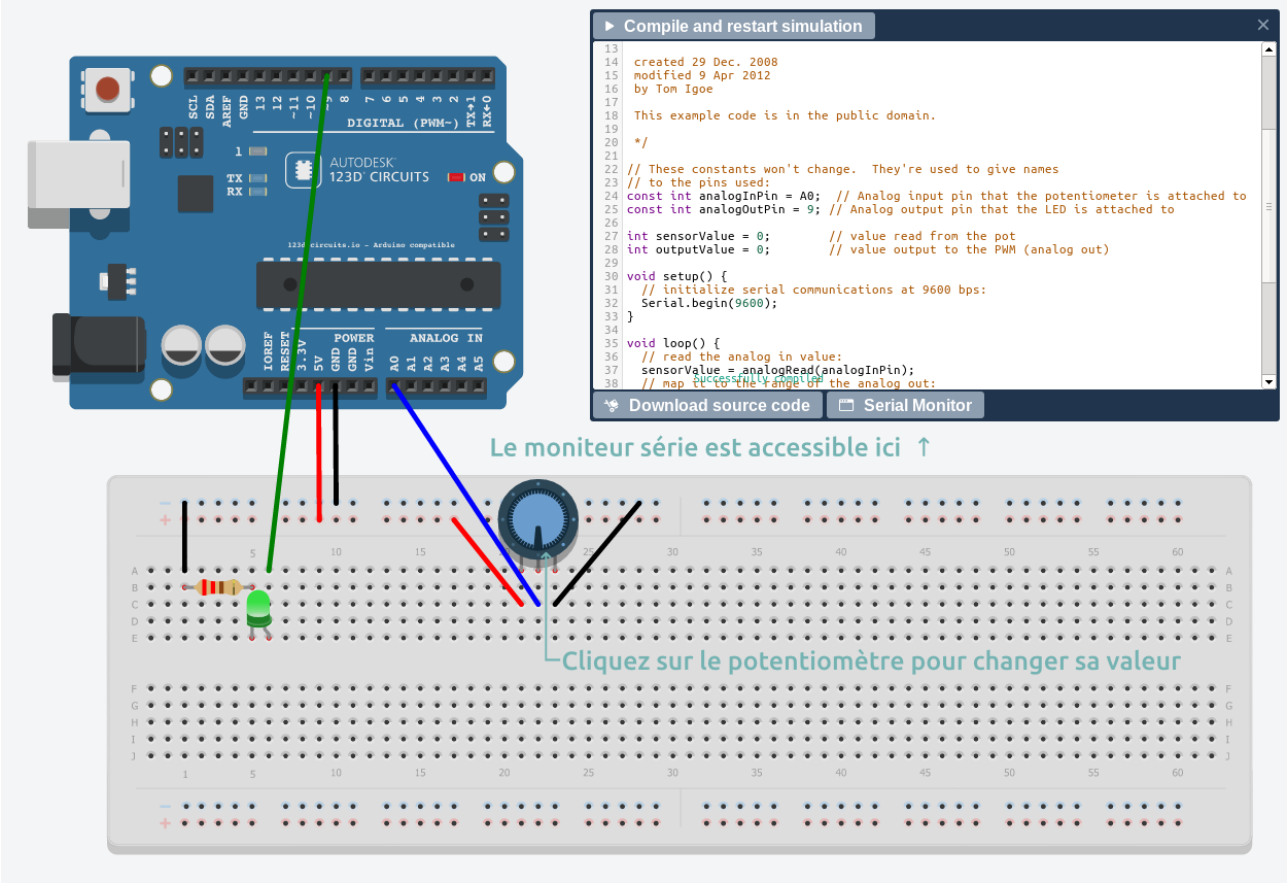


FIGURE I.4.8. – Remplacement de la résistance par un potentiomètre

I.4.2.0.1.6. Références

— Référence standard du langage Arduino par Xavier Hinault
— Chapitre dédié aux capteurs du manuel Arduino de chez FlossManuals
— Compléments sur les capteurs sur le wiki des petits débrouillards

Les références sont là pour vous aider et aller plus loin dans la compréhension des notions abordées dans ce cours donc on n'hésite pas à les consulter

I.4.2.0.1.7. Licence

FIGURE I.4.9. – CC-BY-SA

Nous profitons de cette section pour remercier Xavier Hinault qui met à disposition sur son site www.mon-club-elec.fr une documentation détaillée et libre sur lequel s'appuie ce texte.

I.4.3. Prototypage et électricité

I.4.3.0.1. Composants passifs

Dans les semaines passées, nous avons pas mal discuté des résistances, composant incontournable dans les montages électroniques. Voici d'autres composants dits "passifs" que nous avons souvent besoin d'utiliser…

I.4.3.0.2. Les Diodes

"Ahhhhh…" j'entends déjà les puristes – "une diode est un semi-conducteur, pas un composant passif…" Eh oui, mais je vais traiter le sujet ici et maintenant car nous l'avons déjà vue plusieurs fois dans nos précédents montages.
La diode est le premier vrai composant électronique qui a été inventé. Au début c'était en forme d'énorme tube électronique qui chauffait un max! Maintenant elle existe sous plusieurs formes:

FIGURE I.4.10. – Une diode 1N4148

FIGURE I.4.11. – Une diode 1N4007

Leur fonction de base est toute simple: laisser passer le courant dans un seul sens. On peut penser à un tuyau d'eau : une diode est comme un clapet anti-retour. Voici le symbole électronique:

FIGURE I.4.12. – Symbole de la diode

La forme de flèche n'est pas un hasard, le sens de la flèche indique le sens de circulation du courant électrique. En utilisation "normale", la diode laisse passer le courant quand l'anode est plus positive que la cathode. La cathode est identifiée sur une diode par une bande colorée (voir les photos).
Une forme spéciale de la diode est la diode électroluminescente, ou DEL. Son symbole:

FIGURE I.4.13. – Symbole de la DEL

Les deux petites flèches indiquent que cette diode émet de la lumière… Rappelons notre montage de la semaine passée :

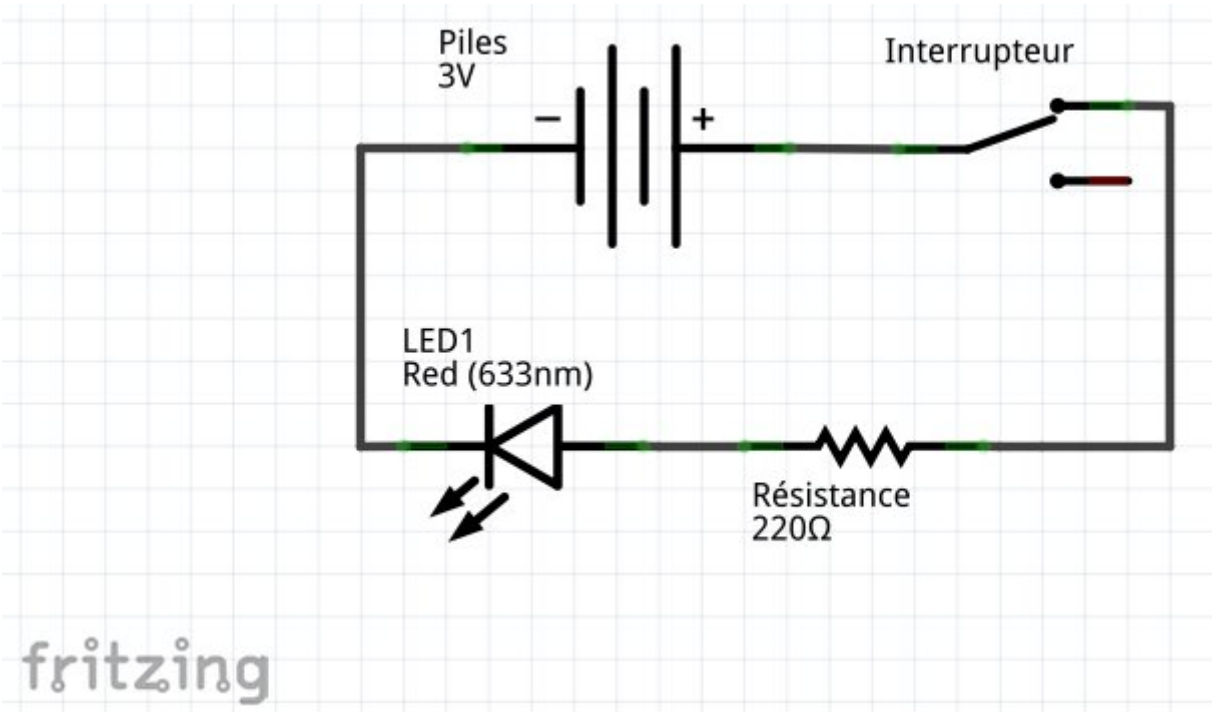


FIGURE I.4.14. – Une LED dans un circuit

On voit bien que l'anode est au positif, et la cathode au négatif. (je sais – ce n'est pas la bonne valeur de résistance avec une alimentation de 3v…).
Si on inverse la LED, elle ne s'allume plus et **le courant ne passe plus**.

I.4.3.0.2.1. Références

— La page Wiki sur comment brancher une LED

Merci à Glenn Smith pour ce cours!

I.4.4. Travaux pratiques

I.4.4.0.1. TP à faire pour la semaine 5

Cette semaine, nous allons tenter de réaliser un Thérémine lumineux qui est un bon exemple pour aborder les entrées analogiques.
Ce montage va être l'occasion d'approfondir les fonctions `map` et `analogRead` mais également de découvrir une nouvelle fonction liée à l'utilisation d'un buzzer piézo-électrique, un composant qui peut produire du son. Le buzzer piézo-électrique transforme l'impulsion électrique envoyée par une broche en une onde sonore de fréquence identique et audible.

FIGURE I.4.15. – Un buzzer

L'exercice consiste à modifier l'exemple *AnalogInOutSerial* vu cette semaine et l'adapter pour que la luminosité mesurée par la photorésistance soit proportionnelle à la fréquence jouée un buzzer branché sur la broche 8: plus la luminosité reçue par la photorésistance est importante, plus la fréquence jouée par le buzzer sera élevée (et inversement, si on passe notre main devant le capteur de luminosité alors le son émis sera grave→fréquence basse).

I.4.4.0.1.1. Quelques indices Vous aurez besoin de la fonction `tone` qui prend deux paramètres:

- **la broche** où le buzzer est branché
- **la fréquence** que l'on veut jouer avec le buzzer (exprimée en Hertz)

```
// la note La (440Hz) jouée sur un buzzer branché sur la broche 8
tone(8, 440);
```

- Lorsque l'on reçoit beaucoup de lumière sur notre photo-résistance, la fréquence générée devra être proche de $30000Hz$ et lorsque la lumière est faible, la fréquence devra être proche de $50Hz$;
- Le buzzer doit avoir une patte connectée à la broche 8 et l'autre au GND.

I.4.4.0.1.2. Quelques conseils

1. N'allez pas regarder la solution sur Internet sinon il n'y a pas de fun;
2. Prenez toujours les hypothèses qui vous arrangent ;
3. Seules les notions abordées dans le cours et sur cette page sont nécessaires pour mener à bien ce TP;
4. Il n'y a pas une mais plusieurs solutions à chaque problème. La meilleure est celle que vous comprenez!

Bon courage et bonne semaine,
L'équipe du MOOC

I.4.5. L'étalonnage, principe et application

Comment faire si nous possédons un capteur mais n'avons pas sa caractéristique exacte? Comment feriez-vous pour faire correspondre une valeur analogique lue avec une donnée physique réelle? Par exemple, si je vous donne un composant en vous disant "Voilà un capteur de température, je sais qu'il s'alimente en 5V sur telle et telle broches, je sais que le signal de sortie est une tension en fonction de la température mais je suis incapable de te dire quelle est la caractéristique (la courbe de tension en fonction de la température)".
Nous allons maintenant voir comment résoudre ce problème en voyant une méthode pour étalonner son capteur. Nous allons ainsi nous même déterminer la courbe caractéristique du capteur et déterminer son coefficient liant la température et la tension.

I.4.5.0.0.1. Le capteur utilisé Pour étudier la méthode que je vous propose ici, nous allons utiliser un capteur de température assez répandu qui se nomme "LM35. Il existe dans différents boîtiers que voici:

Figure I.4.16. – Le capteur de température LM35

Vous aurez probablement deviné le branchement, il est assez simple. Il suffit de relier +VS au 5V et GND à la masse. Le signal sera ensuite lu sur la broche Vout.

I.4.5.0.1. La méthode

La méthode pour caractériser un capteur est assez simple. À l'aide d'une multitude de mesures et d'un appareil témoin, nous allons créer un tableau qui nous servira à calculer la courbe liant la tension à la donnée mesurée (à l'aide d'un tableur). Pour cela, en plus de votre capteur vous aurez besoin d'un appareil de mesure "témoin" qui vous servira de référence. Par exemple le bon vieux thermomètre qui traîne accroché à votre fenêtre fera parfaitement l'affaire!

I.4.5.0.1.1. Prise de mesures Vous êtes prêts, alors allons-y, commençons à travailler. Reliez le capteur à l'Arduino et l'Arduino à l'ordinateur, de la manière la plus simple possible, comme ceci par exemple:

Figure I.4.17. – Branchement simple du LM35

Ensuite, nous devons récupérer les données envoyées par le capteur de manière régulière (ou rajoutez un bouton et faite des envois lors de l'appui). Pour cela, voici un petit programme sans difficulté qui vous enverra les valeurs brutes ou converties en volts toutes les demi-secondes:

```arduino
const int capteur = 0; //capteur branché sur la pin analogique 0
float tension = 0.0;
int valeur = 0;

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

void loop()
{
    valeur = analogRead(capteur);
    tension = (valeur*5.0)/1024;

    Serial.print("Tension : ");
    Serial.print(tension);
    Serial.println(" V");
    Serial.print("Valeur : ");
    Serial.println(valeur);
    Serial.println("---------------------");

    delay(500);
}
```

Maintenant que tout est prêt, il nous faut un banc de test. Pour cela, préparez une casserole avec de l'eau contenant plein de glaçons (l'eau doit être la plus froide possible). Faites une première mesure avec votre capteur plongé dedans (attention, les broches doivent être isolées électriquement ou alors mettez l'ensemble dans un petit sac plastique pour éviter que l'eau n'aille faire un court-circuit). Faites en même temps une mesure de la température réelle observée à l'aide du thermomètre. Une fois cela fait, relevez ces mesures dans un tableau qui possédera les colonnes suivantes:

— Température réelle (en °C)
— Tension selon Arduino (en V)
— Valeur brute selon Arduino

Quand la première mesure est faite, commencez à faire réchauffer l'eau (en la plaçant sur une plaque de cuisson par exemple). Continuez à faire des mesures à intervalle régulier (tous les 5 degrés voire moins par exemple). Plus vous faites de mesure, plus l'élaboration de la courbe finale sera précise. Voici à titre d'exemple le tableau que j'ai obtenu :

Température (°C)	Tension (V)	Valeur CAN
2	0,015	3
5	0,054	11
10	0,107	22
16	0,156	32
21	0,210	43

24	0,234	48
29	0,293	60
35	0,352	72
38	0,386	79
43	0,430	88
46	0,459	94
50	0,503	103

TABLE I.4.2. – Les mesures réalisées

I.4.5.0.1.2. Réalisation de la caractéristique Lorsque vous avez fini de prendre toutes vos valeurs, vous allez pouvoir passer à l'étape suivante qui est: Calculer la caractéristique de votre courbe !! Sortez vos cahiers, votre calculatrice et en avant! … Non j'blague (encore que ça ferait un super TP), on va continuer à utiliser notre logiciel tableur pour faire le travail pour nous! On va donc commencer par regarder un peu l'allure de la courbe. Je vais en faire deux, une symbolisant les valeurs brutes de la conversion du CAN (entre 0 et 1023) en rouge et l'autre qui sera l'image de la tension en fonction de la température en bleu. Nous pourrons alors déterminer deux caractéristiques, selon ce qui vous arrange le plus.

FIGURE I.4.18. – Valeur CAN en fonction de la température

FIGURE I.4.19. – Tension en fonction de la température

Une fois cela fait, il ne reste plus qu'à demander gentiment au logiciel de graphique de nous donner la **courbe de tendance** réalisée par ces points. Sous Excel, il suffit de cliquer sur un des points du graphique et choisir ensuite l'option "Ajouter une courbe de tendance…" . Vous aurez alors le choix entre différents types de courbe (linéaire, exponentielle…). Ici, on voit que les points sont alignés, il s'agit donc d'une équation de courbe linéaire, de type $y = ax + b$. Cochez la case "Afficher l'équation sur le graphique" pour pouvoir voir et exploiter cette dernière ensuite.

FIGURE I.4.20. – Ajouter une courbe de tendance

Voici alors ce que l'on obtient lorsque l'on rajoute notre équation:

FIGURE I.4.21. – Équation de la valeur CAN en fonction de la température

FIGURE I.4.22. – Équation de la tension en fonction de la température

Grâce à l'équation, nous pouvons déterminer la relation liant la température et la tension (ou les valeurs du CAN). Ici nous obtenons:

— $y = 0.01x$–0.0003 (pour la tension)
— $y = 2.056x$–0.0707 (pour les valeurs du CAN)

Le coefficient constant (-0.003 ou -0.0707) peut ici être ignoré. En effet, il est faible (on dit **négligeable**) comparé aux valeurs étudiées. Dans les équations, x représente la température et y représente la tension ou les valeurs du CAN. On lit donc l'équation de la manière suivante: Tension en Volt égale 0,01 fois la température en degrés celsius. Ce qui signifie que dorénavant, en ayant une mesure du CAN ou une mesure de tension, on est capable de déterminer la température en degrés celsius 🍊
Super non? Par exemple, si nous avons une tension de 300mV, avec la formule trouvée précédemment on déterminera que l'on a 0.3=0.01×Temperature, ce qui équivaut à Temperature=0.3/0.01=30oC . On peut aisément le confirmer via le graphique 🍊
Maintenant j'ai trois nouvelles, deux bonnes et une mauvaise… La bonne c'est que vous êtes capable de déterminer la caractéristique d'un capteur. La deuxième bonne nouvelle, c'est que l'équation que l'on a trouvé est correcte… … parce qu'elle est marquée dans la documentation technique qui est super facile à trouver
(ça c'était la mauvaise nouvelle, on a travaillé pour rien !!) Mais comme c'est pas toujours le cas, c'est toujours bien de savoir comment faire 🍊

I.4.5.0.1.3. Adaptation dans le code

Puisque nous savons mesurer les valeurs de notre capteur et que nous avons une équation caractéristique, nous pouvons faire le lien en temps réel dans notre application pour faire une utilisation de la grandeur *physique* de notre mesure. Par exemple, s'il fait 50°C nous allumons le ventilateur. En effet, souvenez-vous, avant nous n'avions qu'une

valeur entre 0 et 1023 qui ne signifiait physiquement pas grand chose. Maintenant nous sommes en mesure (oh oh oh 🍊) de faire la conversion. Il faudra pour commencer récupérer la valeur du signal. Prenons l'exemple de la lecture d'une tension analogique du capteur précédent:

```
int valeur = analogRead(monCapteur); //lit la valeur
```

Nous avons ensuite deux choix, soit nous le transformons en tension puis ensuite en valeur physique grâce à la caractéristique du graphique bleu ci-dessus, soit nous transformons directement en valeur physique avec la caractéristique rouge. Comme je suis feignant, je vais chercher à économiser une instruction en prenant la dernière solution. Pour rappel, la formule obtenue était: y=2.056x–0.0707. Nous avions aussi dit que le facteur constant était négligable, on a donc de manière simplifiée y=2.056x soit "la température est égale à la valeur lue divisé par 2.056 (x=y2.056). Nous n'avons plus qu'à faire la conversion dans notre programme!

```
float temperature = valeur/2.056;
```

Et voilà! Si l'on voulait écrire un programme plus complet, on aurait:

```
int monCapteur = 0; //Capteur sur la broche A0;
int valeur = 0;
float temperature = 0.0;

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

void loop()
{
    valeur = analogRead(monCapteur);
    temperature = valeur/2.056;

    Serial.println(temperature);

    delay(500);
}
```

Et si jamais notre coefficient constant n'est pas négligeable?
Eh bien prenons un exemple! Admettons qu'on obtienne la caractéristique suivante: y=10x+22
On pourrait lire ça comme "ma valeur lue par le CAN est égale à 10 fois la valeur physique plus 22. Si on manipule l'équation pour avoir x en fonction de y, on aurait:

$$beginarrayrcly = 10x + 22y - 22 = 10x\ x = \frac{y - 22}{10}$$

Dans le code, cela nous donnerait:

```
void loop()
{
    valeur = analogRead(monCapteur);
    temperature = (valeur-22)/10;

    Serial.println(temperature);

    delay(500);
}
```

J'espère que tout cela vous aura intéressé! Nous en apprendrons bientôt plus à propos des capteurs de température avec la réalisation de l'arduinomètre au cours de la semaine 6.

Contenu masqué

Contenu masqué n°1

```
/*
  Feu tricolore + Feu Piétons

 TP de la semaine 3 du MOOC "La Fabrication Numérique"

 Le montage :
 * Une LED rouge sur la broche 12 en série avec une résistance de 220Ω
 * Une LED orange sur la broche 11 en série avec une résistance de 220Ω
 * Une LED verte sur la broche 10 en série avec une résistance de 220Ω

 * Une LED rouge sur la broche 9 en série avec une résistance de 220Ω
 * Une LED verte sur la broche 8 en série avec une résistance de 220Ω

 * Bouton poussoir branché sur la broche 2 depuis +5V
 * Une résistance de 1KΩ branché sur la broche 2 depuis GND

 créé le 9 Avril 2014
 par Baptiste Gaultier
 avec les conseils de fb251

 Ce code est en CC0 1.0 Universal

 https://www.france-universite-numerique-mooc.fr/courses/MinesTelecom/04002/Trimestre_1_2014/about
```

```
 */

// Initialisation des constantes pour les LED
const int rouge = 12;
const int orange = 11;
const int verte = 10;

const int rougePieton = 9;
const int vertePieton = 8;


// Numéro de la broche à laquelle est connecté le bouton poussoir
const int bouton = 2;

// Déclaration des variables :
int etatBouton = 0;


// le code dans cette fonction est exécuté une fois au début
void setup() {
  // indique que les broches des LED
  // sont des sorties :
  pinMode(rouge, OUTPUT);
  pinMode(orange, OUTPUT);
  pinMode(verte, OUTPUT);

  pinMode(rougePieton, OUTPUT);
  pinMode(vertePieton, OUTPUT);

  // indique que la broche bouton est une entrée :
  pinMode(bouton, INPUT);
}

// le code dans cette fonction est exécuté en boucle
void loop() {
  // Dans le fonctionnement normal, le feu piéton est toujours
    rouge
  digitalWrite(rougePieton, HIGH);

  // Fonctionnement normal du feu voiture
  digitalWrite(verte, HIGH);
  delay(3000);
  digitalWrite(verte, LOW);

  // lit l'état du bouton et stocke le résultat
  // dans etatBouton :
  etatBouton = digitalRead(bouton);

  // Si etatBouton est égal à HIGH
  // c'est que le bouton est appuyé
```

```
73    if (etatBouton == HIGH) {
74      digitalWrite(orange, HIGH);
75      delay(1000);
76      digitalWrite(orange, LOW);
77
78      digitalWrite(rouge, HIGH);
79
80      // Le feu piéton passe au vert pendant 5s
81      digitalWrite(rougePieton, LOW);
82      digitalWrite(vertePieton, HIGH);
83
84      delay(5000);
85
86      // On remet le feu piéton au vert
87      digitalWrite(rougePieton, HIGH);
88      digitalWrite(vertePieton, LOW);
89
90      // Puis on remet le feu au rouge
91      digitalWrite(rouge, LOW);
92    }
93    else {
94      // Fonctionnement normal du feu voiture
95      digitalWrite(orange, HIGH);
96      delay(1000);
97      digitalWrite(orange, LOW);
98
99      digitalWrite(rouge, HIGH);
100     delay(3000);
101     digitalWrite(rouge, LOW);
102   }
103 }
```

Listing 9 – Une solution au TP du feu tricolore

Retourner au texte.

I.5. Semaine 5 : Bibliothèques et sorties

Introduction

Arduino est modulaire, et c'est tant mieux!
Nous avons choisi la plateforme Arduino pour sa facilité d'utilisation, son ouverture mais aussi parce que cette plateforme est incroyablement modulaire!
Arduino est le cerveau de projets issus de domaines variés: domotique, vêtements intelligents, drones, robotique, objets connectés, installations artistiques… Cette modularité est rendue possible grâce à l'extension des fonctions de base d'Arduino par des modules d'extensions. Transition parfaite puisque c'est le sujet de la semaine!
Après avoir abordé des composants électroniques simples dans les précédentes semaines, nous nous lançons dans la découverte d'un composant plus complexe: le **servomoteur**. C'est un moteur équipé d'un asservissement électronique qui nous permettra de contrôler sa position angulaire. Cette semaine, nous verrons comment brancher ce composant sur notre Arduino et comment utiliser une **bibliothèque** logicielle (un ensemble de fonctions) pour faciliter notre travail (en anglais *library*).
Nous verrons également comment améliorer notre code grâce à l'utilisation de **boucles**.
La semaine s'annonce chargée!

I.5.1. Corrigé du TP 3

I.5.1.0.1. Corrigé du TP Thérémine Lumineux

Voici la correction du TP de la semaine dernière qui reprend des éléments du cours sur les capteurs analogiques.

I.5.1.0.1.1. Code Voici une des solutions possibles pour répondre au problème dans l'état actuel de nos connaissances.

```
/*
    Thérémine lumineux
  TP de la semaine 4 du MOOC "La Fabrication Numérique"

 Le montage :
 * Un piezo branché sur la broche 8
 * Une photorésistance branchée sur la broche A0 depuis +5V
 * Une résistance de 10kΩ branchée sur la broche A0 depuis GND

 créé le 9 Avril 2014
 par Baptiste Gaultier

```

```
 Ce code est en CC0 1.0 Universal

 https://www.france-universite-numerique-mooc.fr/courses/MinesTelecom/04002/Trimestre_1_2014/about
*/

// variable pour stocker la valeur reçue sur A0
int sensorValue;

void setup() {
}

void loop() {
    // lire la valeur de la photorésistance
    // et stocker ça dans une variable
    sensorValue = analogRead(A0);

    // re étalonne la variable vers un grand intervalle de fréquences audibles
    int pitch = map(sensorValue, 0, 1023, 50, 3000);

    // jouer la fréquence sur le piezo branché sur la broche 8
    tone(8, pitch);

    // attendre 10 ms
    delay(10);
}
```

I.5.1.0.1.2. Schéma électronique

Comme on nous l'a demandé sur le forum, voici le schéma électronique créé avec Fritzing :

FIGURE I.5.1. – Theremine lumineux - schéma

I.5.1.0.1.3. Montage

FIGURE I.5.2. – Theremine lumineux - montage

Pour réaliser ce montage, vous avez besoin de:

— Un Arduino;
— Une platine de prototypage;
— Un câble USB;
— Une résistance de 10kΩ;
— Des fils de prototypage;
— Une photorésistance;
— Un piézo;
— Du temps

Notre correction s'arrête là. Pour des éclaircissements, nous vous invitons à poser vos questions sur le forum et à reparcourir le cours.
On vous laisse avec un montage réalisé par ProPhil qui nous a éclaté avec ce montage photo sur le forum:

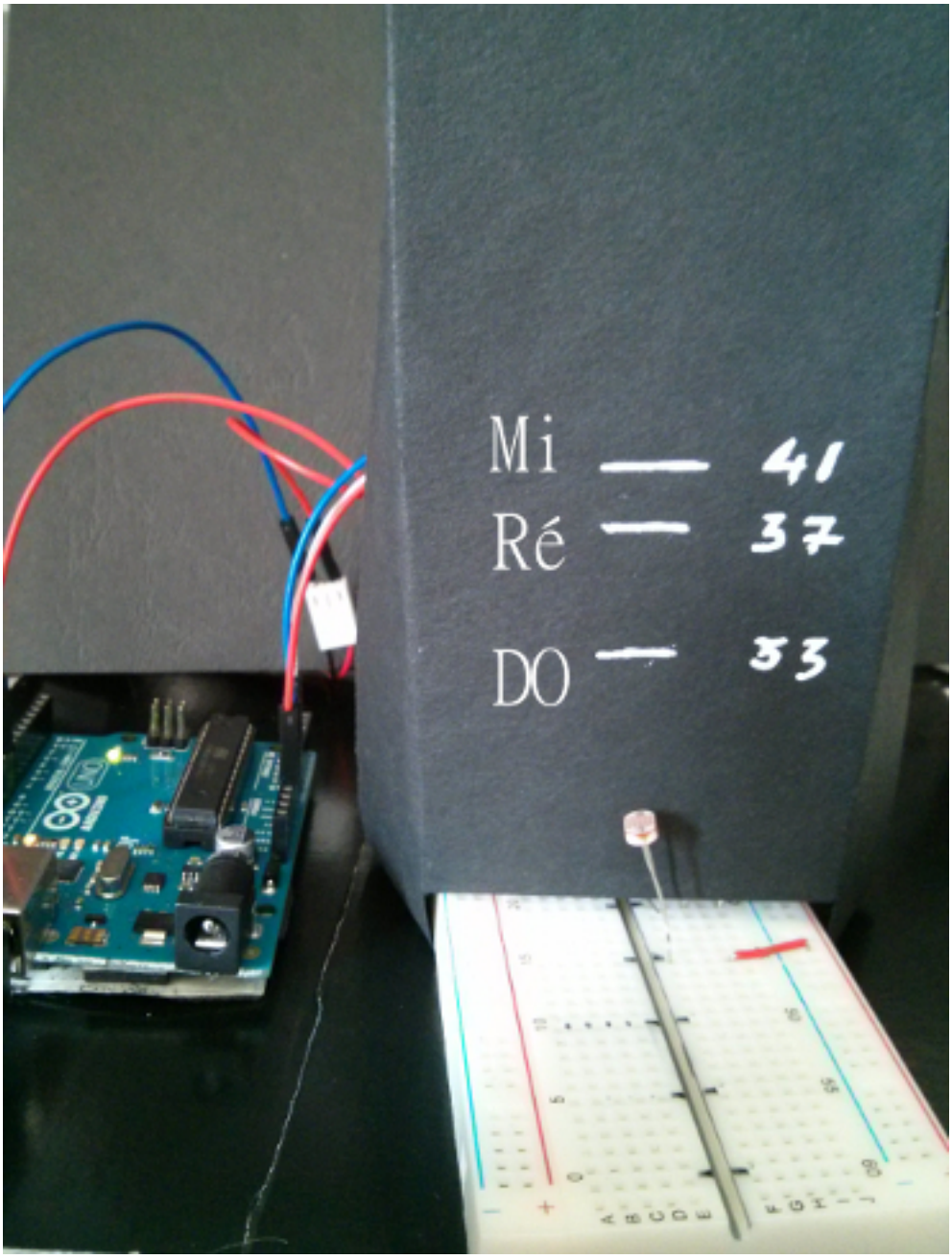


FIGURE I.5.3. – Exemple de montage

I.5.2. Bibliothèques et sorties

Élément externe (Video) —
Consultez cet élément à l'adresse https://geo.dailymotion.com/player.html?video=x2kgzy0&.

I.5.2.0.1. Le Sweep

Cette semaine, nous tentons une nouvelle approche dans le cours avec le code directement expliqué dans le texte de la vidéo que vous venez de voir:
Salut à toutes et tous et bienvenue dans ce 5ème épisode où l'on va aborder les bibliothèques de code.
Bibliothèques, ça vient de l'anglais *Libraries*, qui signifie **Bibliothèques**. Peu importe ce que vous trouverez comme terme sur internet, l'idée c'est d'avoir sous la main un rassemblement de morceaux de code, classés par thématique, que vous invoquez à la demande. Un peu comme des supers pouvoirs qui vous évitent d'avoir à réinventer la roue à chaque projet!
Les Bibliothèques pour Arduino sont nombreuses, et abordent la plupart des besoins courants. On trouve ainsi les bibliothèques standard, pour par exemple gérer le Wifi, les écrans à cristaux liquide, utiliser simplement une carte SD, ou encore des moteurs. Pour l'occasion, nous allons nous intéresser à des moteurs un peu particuliers que l'on retrouve dans le monde du modélisme: les servomoteurs.
Au boulot! Je vous laisse ouvrir l'exemple «sweep» que l'on trouve dans les exemples du dossier *Servo*: *Fichier→Exemples→Servo→Sweep*.
Détaillons maintenant les nouvelles instructions présentes dans ce programme.
Après quelques lignes de commentaires, nous trouvons une instruction particulière:

```
1 #include<Servo.h>
```

Voilà, vous venez de charger la bibliothèque et obtenez du même coup la boîte à outils correspondante. À partir de là, vous pouvez créer des objets en partant du «moule» `Servo`, un peu comme l'on fait de nombreuse gaufres à partir d'un seul moule, à gaufre forcement! C'est ce qu'on va tout de suite faire. En créant une gaufre chantilly…
Non, pardon! Un servomoteur, que nous appelons ici `myservo`.

```
1 Servo myservo;
```

On déclare ensuite une variable `pos`, pour stocker une position au cours du programme:

```
1 int pos = 0;
```

Il est temps de passer au bloc `setup()`. C'est vite réglé, puisqu'il suffit d'attacher notre servo fraîchement créé à la broche 9:

```
1 myservo.attach(9);
```

La méthode `attach()` est disponible pour les objets de type Servo. La bibliothèque de code gère le reste pour vous. Sympa, non?
Comme indiqué sur la page de Mon Club Elec , les méthodes disponibles sur les objets de type Servo sont assez explicites: `attach()`, `write()`, `detach()`.
Fin du setup, il est temps de looper à présent!
La **boucle principale** commence par… faire une boucle!

```
1 for(pos = 0; pos < 180; pos +=1) {
2     myservo.write(pos);
3     delay(15);
4 }
```

Cette "bouclette" à pour but de faire varier la position cible demandée au moteur. Autrement dit, elle sert uniquement à faire varier la valeur de `pos` du minimum au maximum. En français, ça donne quelque chose comme «Pour une valeur `pos`ition allant de 0 à 179 et une marche à la fois, demande au servo `myservo` d'aller en position `pos` » Un `pos += 4` vous aurait fait monter l'escalier 4 à 4
On laisse quelques millisecondes au moteur afin qu'il ait le temps d'aller une marche plus loin, concrètement de tourner de 1°.
C'est reparti pour un tour, cette fois-ci dans l'autre sens. Au lieu d'incrémenter la valeur de +1 à chaque passage de bouclette, on la décrémente ici de -1:

```
1 for(pos = 180; pos >= 1; pos -= 1) {
2     myservo.write(pos);
3     delay(15);
4 }
```

Puisque nous arrivons en fin de boucle, notre servo va bientôt repartir. Certains d'entre vous doivent se dire: «Pourquoi ne pas être aller directement au but avec par exemple?»:

```
1 myservo.write(0);
2 myservo.write(180);
```

La réponse tient en un mot: la douceur!
Utiliser un délai dans une **boucle secondaire** permet de ralentir le processus. Plus la valeur du délai sera importante dans votre programme, plus le déplacement du moteur sera doux dans la vrai vie… À vous d'adapter la valeur du délai à vos besoins de tendresse.
Nous pouvons maintenant téléverser (ou *upload* en anglais) notre programme sur l'Arduino.

I.5.2.0.1.1. Montage

FIGURE I.5.4. – Montage d'un servo-moteur

Pour le réaliser, vous aurez besoin de:

— Un Arduino;
— Un câble USB;
— Des fils de prototypage;
— Une platine de prototypage;
— Un servomoteur.

I.5.2.0.1.2. Schéma

FIGURE I.5.5. – Branchement d'un servomoteur

I.5.2.0.1.3. Code

```
#include <Servo.h>

Servo myservo;
// créer un objet appelé myservo à partir
// du moule Servo

int pos = 0; // variable pour stocker la position courante du servo

void setup()
{
    myservo.attach(9);
    // attacher notre objet myservo au servomoteur branché sur la
        broche 9
}


void loop()
{
    for(pos = 0; pos < 180; pos += 1)  // aller de 0° à 180°
    {
        // une étape à la fois
        // aller à la position stocké dans 'pos'
        myservo.write(pos);
        // attendre 15ms que le servomoteur se rende à 'pos'
```

```
24          delay(15);
25      }
26      for(pos = 180; pos>=1; pos-=1) // aller de 180° à 0°
27      {
28          // aller à la position stocké dans 'pos'
29          myservo.write(pos);
30          // attendre 15ms que le servomoteur se rende à 'pos'
31          delay(15);
32      }
33  }
```

Listing 10 – Un balayage en douceur

I.5.2.0.1.4. Références

— La bibliothèque Servo par Xavier Hinault
— Référence officielle du langage Arduino (anglais) par l'équipe d'Arduino

I.5.3. Complément de cours

Puisque vous n'êtes pas aussi fainéant(e)s qu'il n'y paraît, nous allons appliquer le savoir nouvellement acquis, en réalisant un exemple d'application concrète.
Seulement voilà, un seul moteur qui bat la mesure, c'est vite barbant. Que diriez-vous d'une machine qui servirait votre boisson favorite, sans broncher? Que diriez-vous de faire votre premier pas en robotique sans avoir l'air d'y toucher?
... Voici venir le BoissonMatic'!
Vous qui rêvez d'une invention géniale et susceptible de révolutionner l'univers, j'ai ce qu'il vous faut!
Le **BoissonMatic'** est une réalisation totalement absurde et c'est bien ce qui la rend indispensable. Elle nous servira ici d'illustration de principe pour un montage fort utile : le **Pan Tilt**, ou Monture azimutale ("Fiche wikipédia sur la monture azimutale") dans notre bonne vieille langue de Molière.
Le principe est simple et consiste à monter un servo sur l'autre, perpendiculairement. Pendant que le premier moteur tourne à l'horizontal (**pan**oramique), le second tourne à la vertical (azimut).

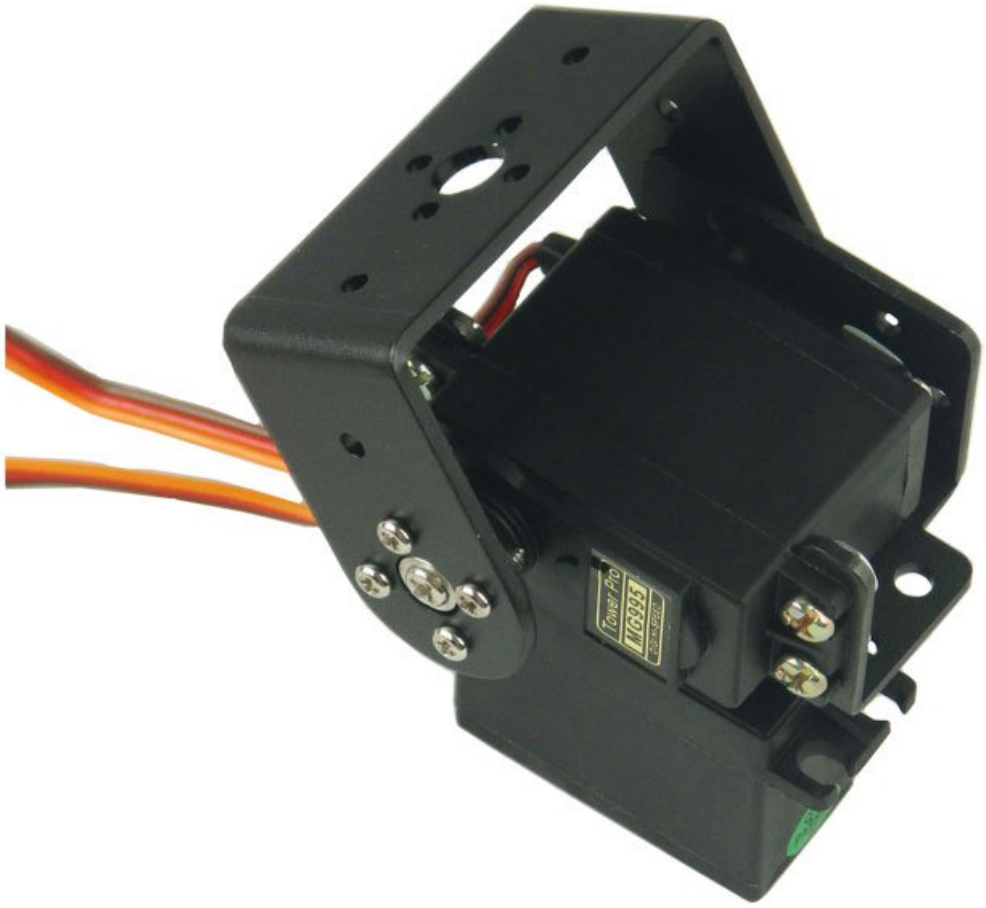


FIGURE I.5.6. – Assemblage sur un servomoteur

Avec des servomoteurs basiques, les déplacements de chacun des axes seront limités à 180°. Ceci étant, vous pouvez facilement trouver un servo tournant à 360° dans n'importe quelle boutique de modèle réduit, ou encore modifier un servo pour une rotation continue .

I.5.3.0.0.1. Code *again*

```
// BoissonMatic' 2014// john@labfab.fr sous licence WTFPL v2.0
#include <Servo.h>

Servo servoBas;  //  création d'un premier objet servo, pour le Pan
Servo servoHaut;  //  création du second, pour le tilt

int pos = 0;    // variable pour stocker une position

// valeur mini du Pan, en degré.
int  const MINPAN = 0;
// Un quart de tour panoramique
int const MAXPAN = MINPAN + 90;
// valeur mini du Tilt (en cas de servo monté "à l'envers" ! :] )
int  const MINTILT = 180;
// avec un max au quart de tour
```

```
16 int const MAXTILT = MINTILT - 90;
17
18 void setup()
19 {
20     servoBas.attach(9);   // attache le servo du bas à la broche 9
21     servoHaut.attach(10); // celui du haut à la broche 10
22
23     // Mise en place de la machine en position de départ
24     servoBas.write(MINPAN);
25     servoHaut.write(MINTILT);
26
27     delay(1000); // admettant qu'il faille une seconde pour faire
           demi-tour
28                  // à un servo bien nourri.
29 }
30
31
32 void loop()
33 {
34     // on fait faire un quart de tour au servo panoramique
35     servoBas.write(MAXPAN);
36
37     // La boucle "for" permet d'adoucir le déplacement du moteur
38     for(pos = 180; pos>=70; pos-=1) // On dépasse le quart de tour,
39                                     // pour que le versement du
       liquide soit plus rapide
40     {
41         servoHaut.write(pos);
42         // le servo tourne de la valeur de l'angle indiqué par la
               variable 'pos'
43         delay(20);
44         // on laisse un peu de temp au servo pour arriver à la
               postion demandée
45     }
46     delay(3000); // encore du temps pour qu'assez de liquide coule
           du récipient.
47
48     for(pos = 70; pos<=180; pos+=1) // la même dans l'autre sens
49     {
50         servoHaut.write(pos);
51         delay(25);
52     }
53
54     servoBas.write(MINPAN ); // retour en position initiale
55     delay(400);
56
57     servoHaut.detach(); // on détache les moteurs un par un.
58     servoBas.detach();
59     // la séquence n'aura lieu donc physiquement lieu qu'une seule
           fois,
```

```
60      // même si la boucle principale loop() continue à tourner.
61
62      // devrait retourner quelque chose  // comme "false" sur votre
            moniteur serie.
63      servoHaut.attached();
64  }
```

Vous l'aurez compris, la boucle `for` est fort fort utile. C'est même la base du fonctionnement d'Arduino, vous ne pouvez pas *looper* ça… 🍊
Pour rappel et pour ceux qui roupillaient au fond de la salle, sa syntaxe est toujours la même.

```
1  for (condition) {
2      // ici du code pour bouclette
3  }
```

Ce qui se trouve entre les `{}` s'appelle un **bloc d'instruction**. Les variables y ont un comportement spécifique que nous aborderons une autre fois si vous le voulez bien!
Vous allez devoir apprendre à la boucler, car vous utiliserez cette **structure de contrôle** pour de nombreux propos. Imaginez par exemple une multitude de servomoteurs que vous voudriez "détacher". En plaçant vos objets dans un tableau, vous appliquerez un bon principe de fainéantise. Il ne vous restera plus qu'à parcourir ce tableau d'un `for`, et d'appliquer pour chaque tour de boucle secondaire, la méthode `detach` sur l'objet.
Si vous vous êtes amusé(e)s dans cette modeste découverte d'une bibliothèque standard particulièrement adaptée aux servos, adoptez le bon réflexe et sachez que pour chacun de vos projets, il en existe certainement une qui fera votre affaire.
Vous n'aurez ainsi pas à résoudre des problèmes déjà réglés, ni à recoder ce qui l'a déjà été mille fois. Vous pourrez ainsi vous concentrer sur votre problématique, et pas sur le code qui gère "la tuyauterie" sous-jacente. N'hésitez pas à parcourir les bibliothèques existantes, le soir au coin du feu. Vous apprendrez beaucoup, et gagnerez au final un temps précieux.

I.5.3.0.0.2. Ressources

— fichier source au format .dxf pour découpe laser du socle BoissonMatic'.
— des exemples de chassis sur thingiverse.

Pour celles et ceux qui n'auraient pas un second servomoteur à disposition, ni un lab dans les parages, vous pouvez toujours suivre le TP général qui n'utilise qu'un seul servo. Vous améliorerez votre feu bicolore en y ajoutant une barrière. Ceinture et bretelles!

I.5.4. Les condensateurs

I.5.4.0.1. Composants passifs

Nous avons pas mal discuté des résistances, composant incontournable dans les montages électroniques. Voici d'autres composants dits "passifs" que nous avons souvent à utiliser:

I.5.4.0.1.1. Les Condensateurs ou capacitances

Sujet un peu plus difficile si on veut tout comprendre des capacitances – mais nous allons rester simple… Voici le symbole électronique:

FIGURE I.5.7. – Symbole du condensateur

Leur construction et leur forme peut varier beaucoup mais leur principe reste le même: deux matériaux conducteurs (les deux traits verticaux), séparés par une matière isolante, appelée diélectrique (l'espace entre les traits), l'ensemble souvent en forme de "sandwich" ou parfois enroulés.

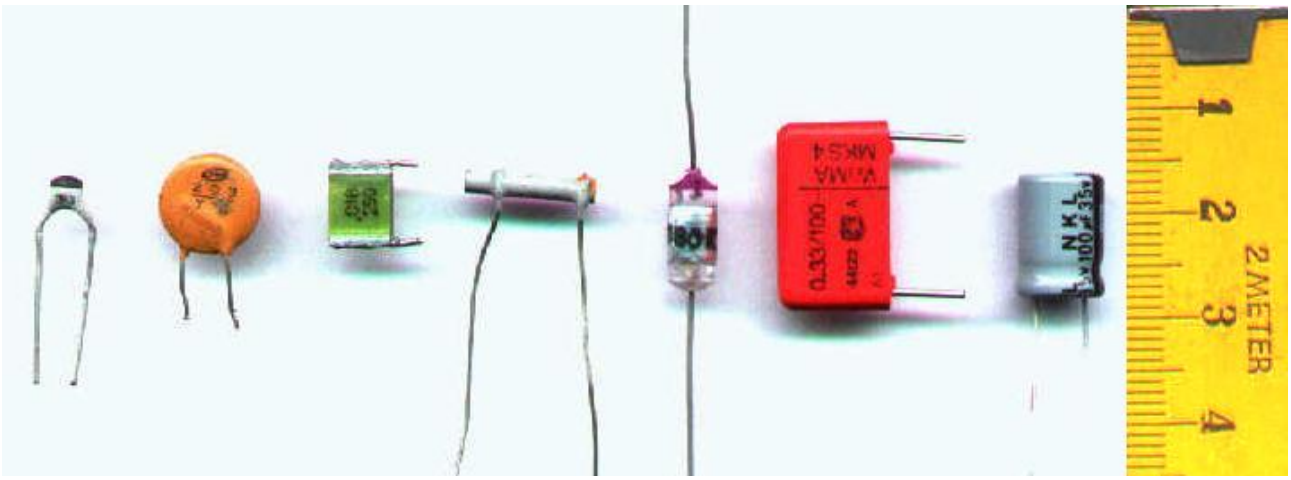


FIGURE I.5.8. – Un échantillon de condensateurs

I.5.4.0.1.2. Condensateurs polarisés Il faut noter que, à partir de la valeur d'environ 1uF, bon nombre de condensateurs sont polarisés - c'est à dire qu'ils ont un "+" et un "-". On les appellent souvent condensateurs chimiques. Il faut les brancher dans le bon sens afin d'éviter les surprises désagréables: inversés ils ont tendance à se gonfler jusqu'à parfois se rompre carrément. Les produits chimiques contenus dedans sont très corrosifs…
D'habitude c'est le fil "-" qui est repéré avec une bande noire sur le coté. Photo: merci Wikipedia:

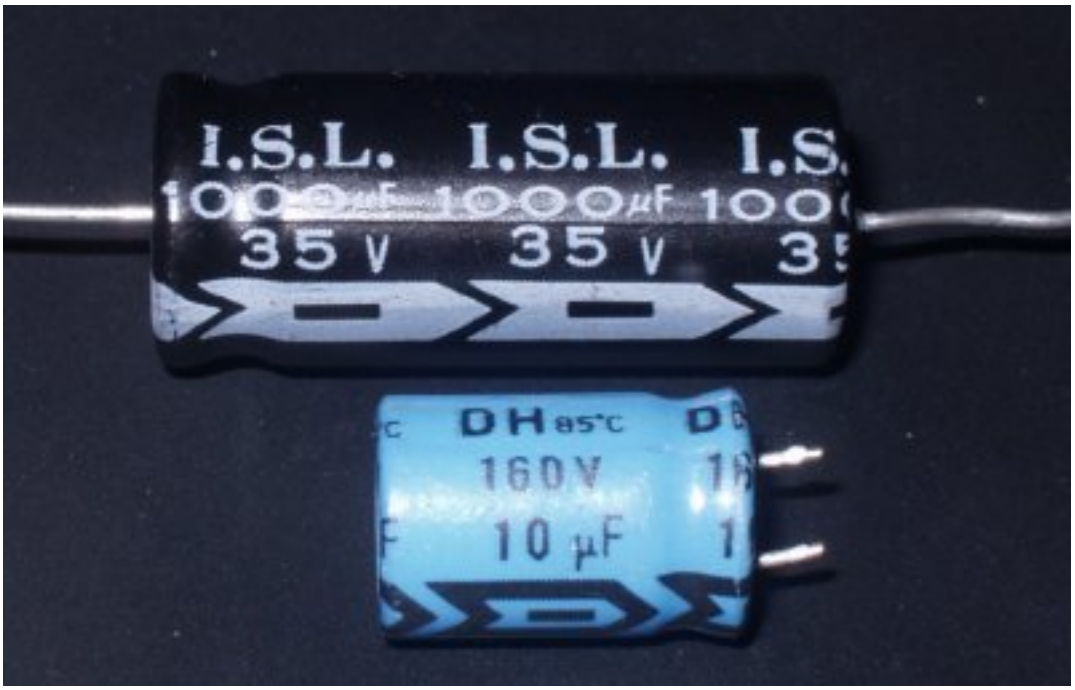


FIGURE I.5.9. – La bande symbolise le côté '-'

Le condensateur est conçu pour stocker les charges électriques. Leur valeur est mesurée et calculée en Farads (F). Plus la valeur est grande et plus le condensateur peut stocker de charge. Les valeurs rencontrées dans l'électronique classique sont de l'ordre de nanofarads (10^{-9} Farads), jusqu'à des centaines de microfarads (10^{-6} F). Souvent les valeurs sont écrites de la manière suivante:

- 47uF = 47 microfarads ou 47×10^{-6};
- 100nF = 100 nanofarads ou 100×10^{-9};
- 14pF = 14 picofarads ou 14×10^{-12} (circuits radio);

À quoi servent des condensateurs? Je vais vous montrer quelques exemples pratiques de leur utilisation.

I.5.4.0.1.3. Anti-rebond Nous avons déjà utilisé des boutons afin de commander nos montages Arduino et vous avez sûrement eu à faire avec le problème de rebond des contacts (*contact bounce*, en anglais). La construction mécanique des boutons fait que les contacts rebondissent plusieurs fois avant de s'arrêter en position ouverte ou fermée. Avec un Arduino capable de réagir en un milliardième de seconde, ces rebonds sont un vrai casse-tête.
D'habitude on résout le problème dans le logiciel - on attend un certain nombre de millisecondes et on lit à nouveau l'entrée concernée pour déterminer l'état. Sachez toutefois qu'il y a aussi une solution électronique qui peut parfois nous simplifier la vie. Voici le branchement typique d'un bouton en entrée d'un microcontrôleur:

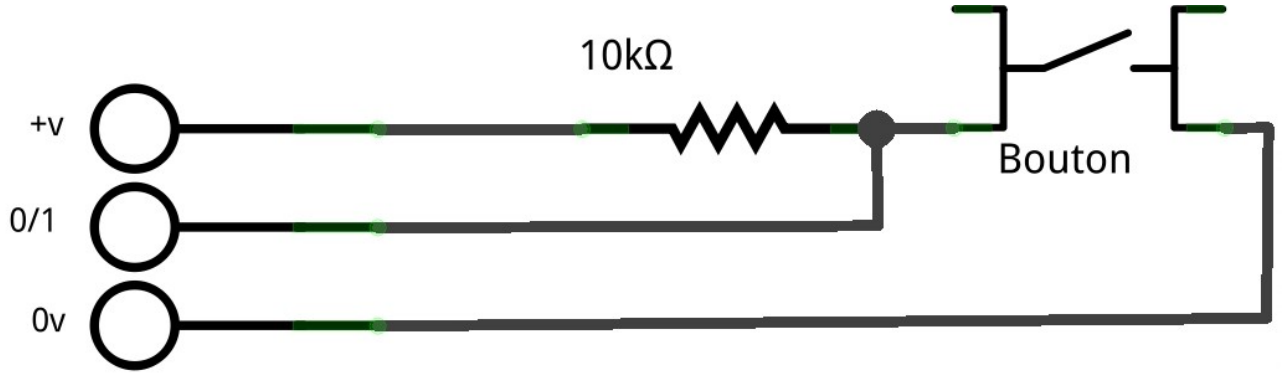


FIGURE I.5.10. – Branchement simple d'un bouton

L'entrée est tenue au niveau 1 par la résistance. Quand on appuie sur le bouton l'entrée est 'tirée' vers le niveau 0. Regardons déjà à quoi ressemble le 'bounce':

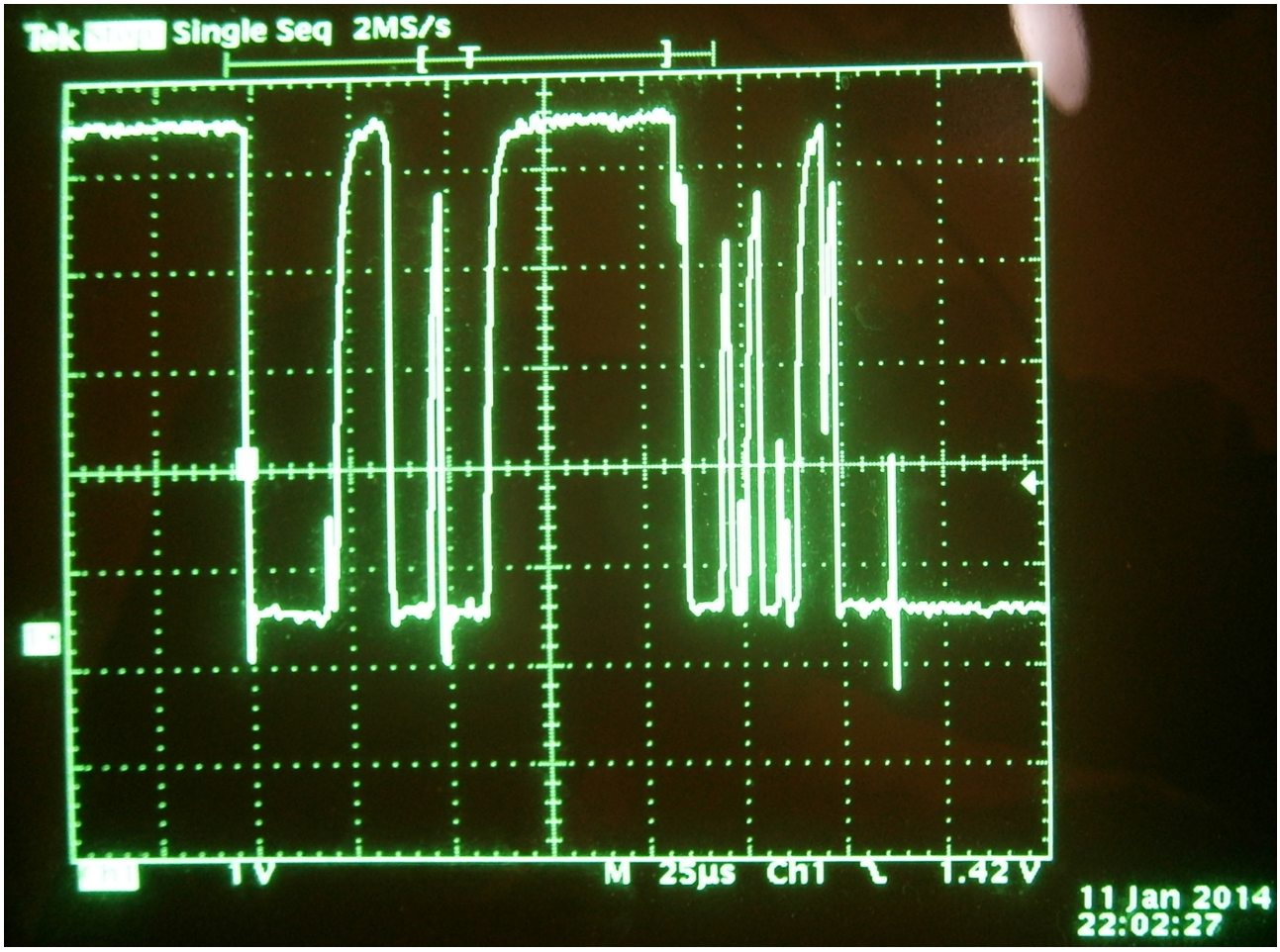


FIGURE I.5.11. – Un signal pas très propre

Cette trace d'oscilloscope correspond à une seule pression sur le bouton ! La durée dans le temps de cette trace est de 200 micro-seconds (uS). Chaque division fait 25uS. On voit bien que cela peut être interprété par le microcontrôleur comme au moins six pressions! On peut améliorer la situation on ajoutant un condensateur (et oui, c'est toujours le sujet de ce chapitre…) comme ici:

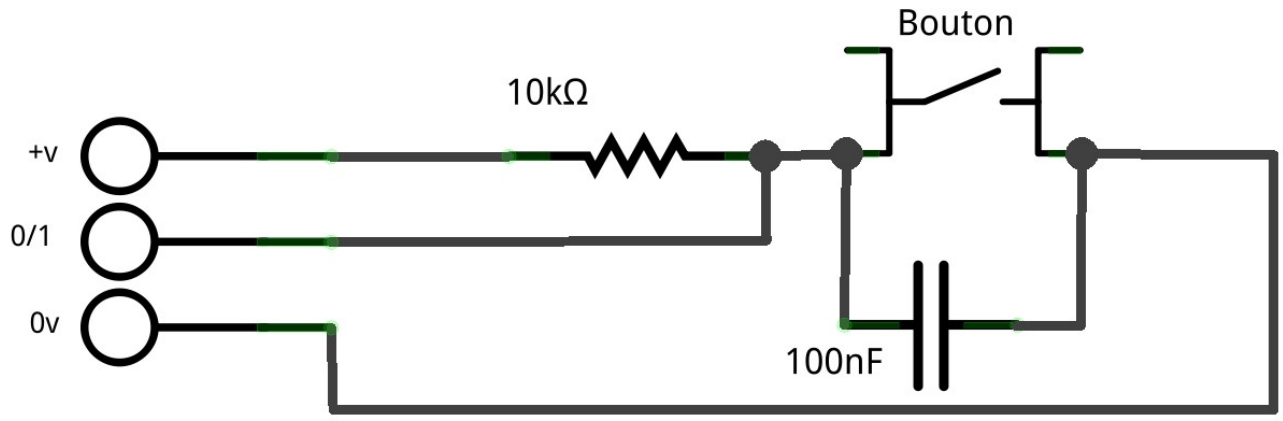


FIGURE I.5.12. – Branchement d'un bouton avec un condensateur

Un condensateur a besoin de temps pour se charger et se décharger. L'ajout de ce (petit) condensateur en parallèle avec le bouton va 'ralentir' les changements de tension, supprimant les effets aléatoires dus aux faux contacts. Et avec cette petite modification, nous avons:

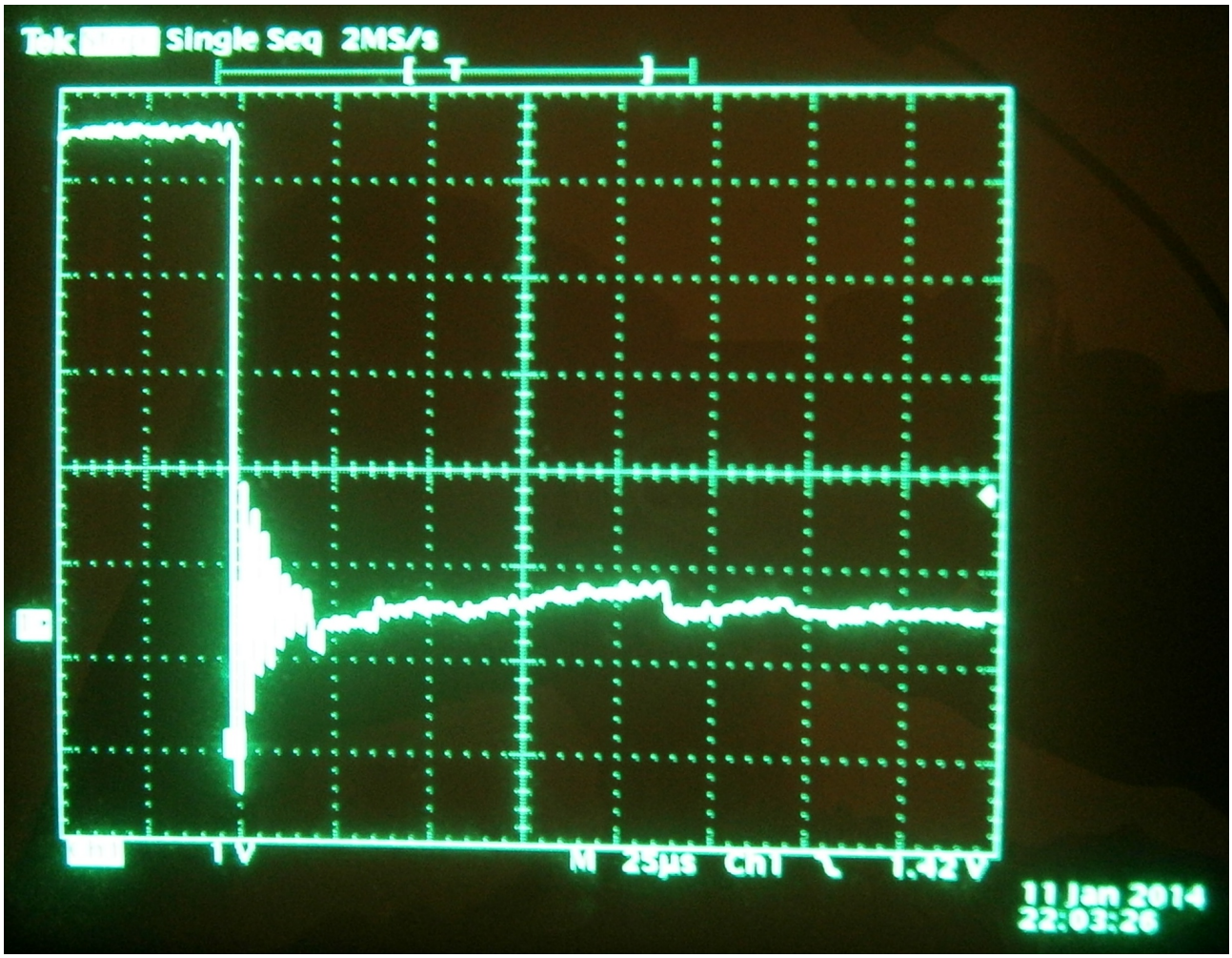


FIGURE I.5.13. – Un signal plus propre

Ce n'est pas un carré parfait, mais notre Arduino ne verra qu'une seule pression. Sans avoir à ajouter du code spécifique anti-rebond. Utile les petits condensateurs!

I.5.4.0.1.4. Base de temps RC Voilà un montage simple à construire avec l'Arduino. D'abord le montage sur la platine de prototypage ou "breadboard" (littéralement "planche à pain" en anglais…):

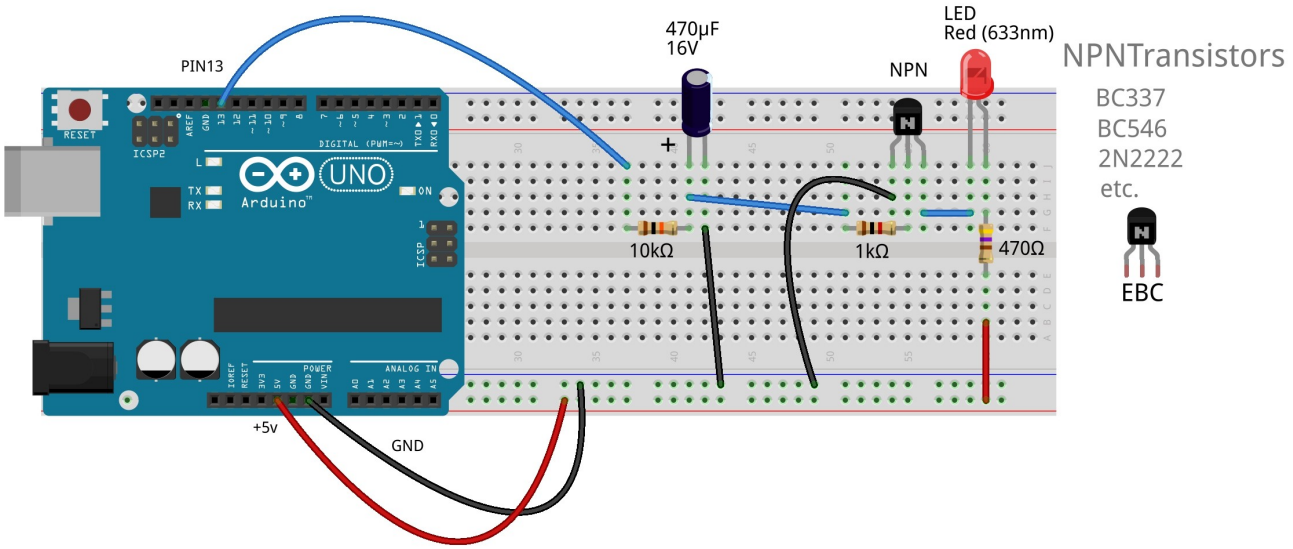


FIGURE I.5.14. – Expérimentation sur breadboard

Très simple à câbler, ce petit montage sert pour voir comment les condensateurs peuvent nous fournir des bases de temps. La sortie 13 de l'Arduino est celle équipée d'une DEL (LED) sur la platine de l'Arduino. Quand la sortie est au niveau '1' la DEL s'allume. Nous prenons la même sortie logique pour piloter notre montage. À travers la résistance de 10kΩ le condensateur se charge pendant que la sortie 13 est au niveau '1', et se décharge pendant le niveau '0'. Le transistor est utilisé comme une sorte d'interrupteur piloté (nous allons parler des transistors bientôt…) pour allumer la DEL rouge. Voici le schéma:

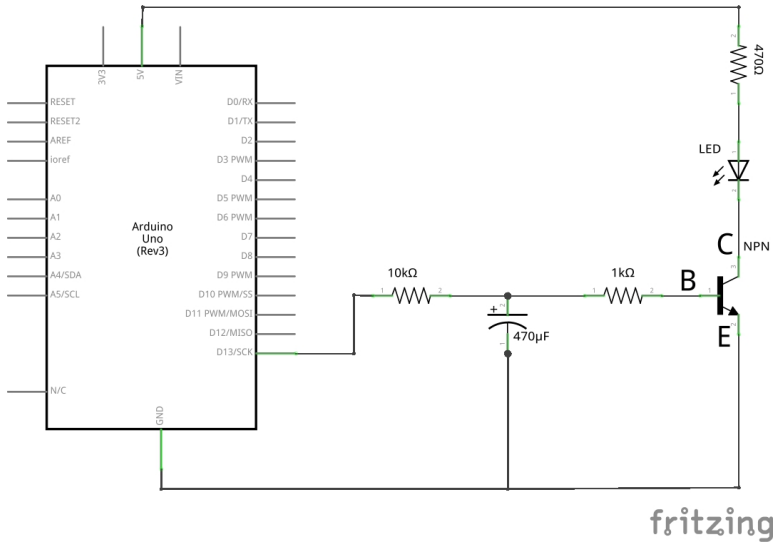


FIGURE I.5.15. – Schéma de l'expérience

Et le programme (ou sketch en anglais pour l'Arduino): (l'exemple *Blink* modifié un peu)

```
/*
    SlowBlink
    Turns on an LED on for 5 seconds, then off for 5 seconds,
    repeatedly.
    Derived from the 'Blink' example in the Arduino suite.
    This example code is in the public domain.
 */

// Pin 13 has an LED connected on most Arduino boards.
// give it a name:
int led = 13;

// the setup routine runs once when you press reset:
void setup() {
    // initialize the digital pin as an output.
    pinMode(led, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {
    digitalWrite(led, HIGH);   // turn the LED on (HIGH is the
        voltage level)
    delay(5000);               // wait for 5 seconds
    digitalWrite(led, LOW);    // turn the LED off by making the
        voltage LOW
    delay(5000);               // wait for 5 seconds
}
```

À partir des schémas, branchez les composants sur la platine de prototypage puis téléchargez le programme et vérifiez que la LED sur l'Arduino clignote (très) lentement. Regardez la LED rouge sur la platine d'expérimentation: elle clignote aussi, mais avec presque une seconde de retard. Le délai exact dépend de plusieurs paramètres, notamment le type de transistor.
Dans un monde idéal[1], le temps de charge/décharge d'un condensateur se calcule facilement: $T = R \times C$

- T le temps en secondes;
- R la résistance en Ohms;
- C la capacitance en Farads.

Merci à Glenn Smith pour ce cours!

1. Mais ce temps dépend beaucoup des conditions de branchement, notamment la résistance de sortie du montage (souvent appelée l'impédance). Prenant l'exemple ici, nous avons R = 10000Ω et C = 0,00047F ce qui donne le temps de charge ou décharge de 4,7 secondes. Pourquoi notre DEL n'avait qu'un retard d'environ 1 seconde, alors? La réponse au chapitre "transistors"! Vous pouvez changer le condensateur pour un plus grand ou plus petit, ou en mettre plusieurs en parallèle ou série. Vous verrez que les **valeurs des condensateurs en parallèle s'additionnent**, et la valeur des condensateurs en série se calcule comme ceci: $C_{tot} = \frac{1}{\frac{1}{C_1}+\frac{1}{C_2}+\cdots+\frac{1}{C_2}}$ (exactement l'inverse que pour les résistances)

I.5.5. Travaux pratiques

I.5.5.0.1. TP à faire pour la semaine 6

Cette semaine, nous restons dans nos montages de feux en compliquant un peu la chose avec une barrière. Le montage à réaliser devra comporter:

- Un servomoteur qui jouera le rôle de barrière;
- Un bouton pour demander l'ouverture de la barrière;
- Un feu bicolore qui passera au vert lorsque la barrière sera complètement ouverte.

Le scénario sera le suivant:

FIGURE I.5.16. – Illustration du fonctionnement

Le fonctionnement normal est un feu allumé au rouge et une barrière fermée (0°). Le fonctionnement normal est interrompu par l'appui sur un bouton poussoir.
Si l'appui du bouton est détecté, alors la barrière (actionnée par le servomoteur) se relève

doucement. Lorsque la barrière est à la verticale (90°), le feu vert s'allume pendant 5 secondes pendant lesquelles la barrière reste ouverte (90°). Après les 5 secondes, le feu repasse au rouge, la barrière redescend doucement et le fonctionnement normal reprend.
Aussi, nous souhaitons recevoir le message "Bouton appuyé" dans le moniteur série lorsque l'appui a été détecté.

I.5.5.0.1.1. Quelques indices Vous aurez besoin de mobiliser toutes les compétences vues ces dernières semaines pour réaliser ce TP:

- L'utilisation de boucles `for` qui ont été décrites cette semaine;
- L'utilisation d'entrée et de sorties numériques;
- Importation de bibliothèques et d'un servomoteur;
- Utilisation des instructions `Serial`.

I.5.5.0.1.2. Quelques conseils

- Avertissez nous sur le fil de discussion ci-dessous si les consignes ne vous semblent pas claires;
- N'allez pas regarder la solution sur Internet sinon il n'y a pas de fun;
- Prenez toujours les hypothèses qui vous arrangent;
- Seules les notions abordées dans le cours et sur cette page sont nécessaires pour mener à bien ce TP;
- Il n'y a pas une mais plusieurs solutions à chaque problème. La meilleure est celle que vous comprenez!

Bon courage et bonne semaine,
L'équipe du MOOC

I.5.6. Environnements de développement

I.5.6.0.1. Annexe

Lors du montage de ce MOOC, nous avons fait le choix de parler du **code**: le langage que nous utilisons pour programmer Arduino.
Sachez cependant qu'il existe des alternatives pour développer des programmes pour Arduino:

- **S4A** (Scratch pour Arduino): est un environnement de programmation libre qui permet d'aborder la programmation de façon ludique et intuitive. Le concept d'écriture du programme repose sur un principe d'assemblage de modules classés par code couleur (comme l'idée des Lego) et chaque module correspond à une instruction (`if`, `delay`, `for`, `digitalWrite`…). Pour installer ce logiciel et en savoir plus sur son utilisation, nous vous invitons à lire la section consacrée à S4A sur le FlossManual Arduino .

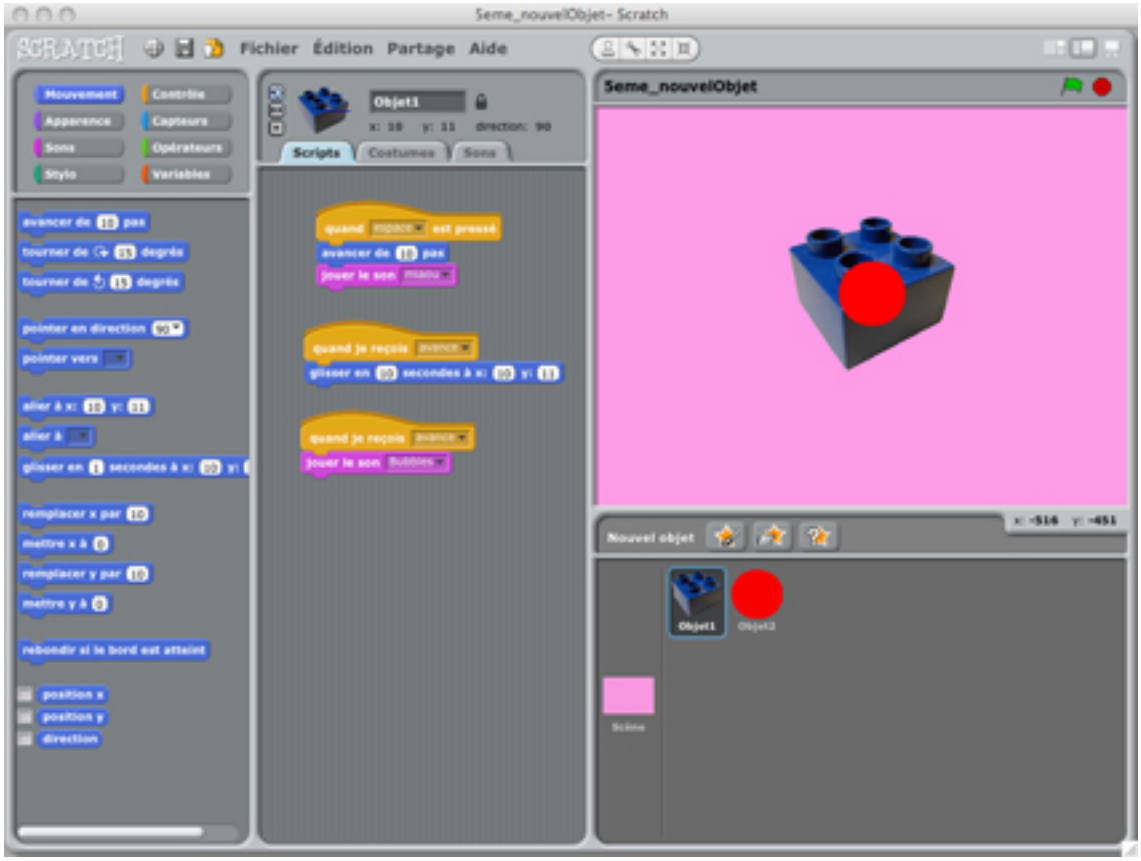


FIGURE I.5.17. – S4A - Scratch For Arduino

— **MATLAB**: est un langage de haut niveau et un environnement interactif pour le calcul numérique, la visualisation et la programmation. Une extension de MATLAB appelée Simulink Coder permet de générer et exécuter du code pour Arduino à partir d'un langage graphique basé sur l'assemblage de briques symbolisant des instructions ou fonctions. Dans le cadre de ce MOOC, la société MathWorks a réalisé les vidéos d'initiations suivantes:
  — Installation du Support Package pour Arduino
  — Création d'un Modèle pour la Cible Arduino
  — Exécution de Modèles sur la Cible Matérielle

  MathWorks nous permet également de télécharger gratuitement une version de leurs outils durant toute la durée du MOOC. Cliquez ici pour en savoir plus et là pour récupérer une licence du logiciel.

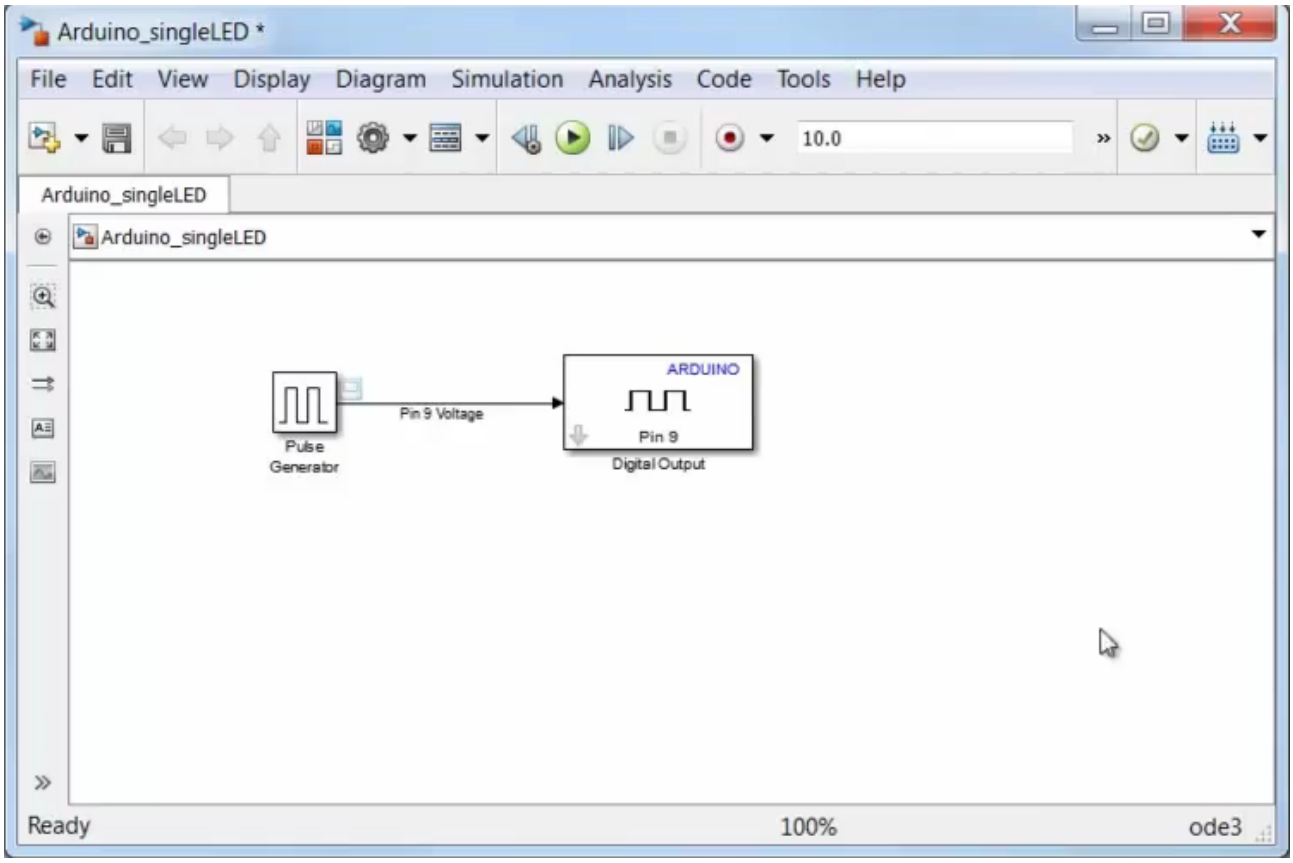


FIGURE I.5.18. – SIMULINK

— **ArduBlock**: est une extension libre du logiciel Arduino (celui utilisé dans le MOOC) qui permet de programmer en utilisant là encore des blocs symbolisants des tests, des boucles et des fonctions. Pour installer ce logiciel et en savoir plus sur son utilisation, nous vous invitons à lire cette page .

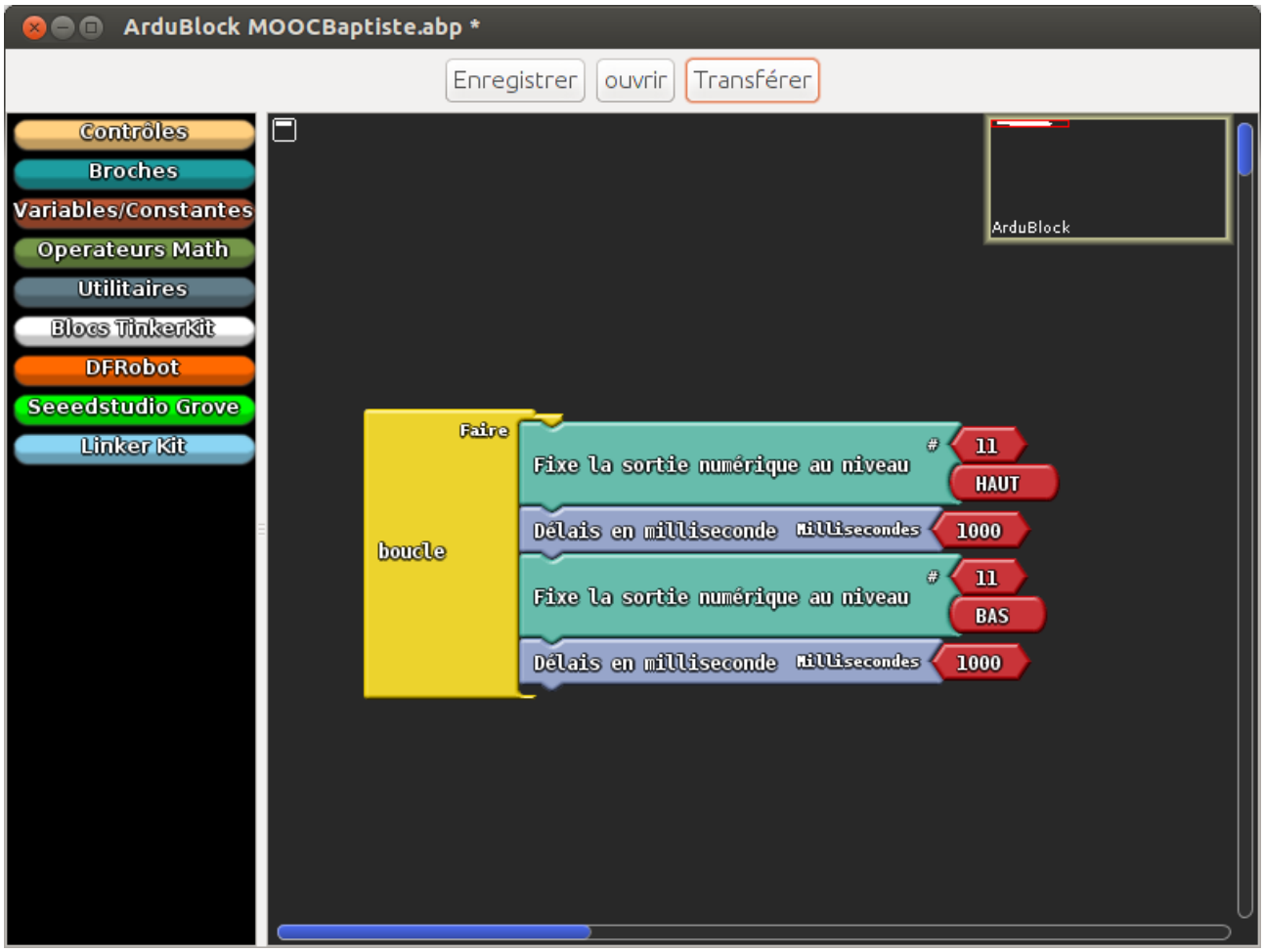


FIGURE I.5.19. – ArduBlock

I.5.6.0.1.1. Licence

/opt/zds/data/contents-public/la-fabrication-n

FIGURE I.5.20. – CC-BY-SA

I.5.7. Séparer en fichiers

Lorsque vous commencez à faire de gros projets, il devient utile voire indispensable de (très) bien organiser son code. Cela commence par séparer son code en différents fichiers afin d'avoir des entités logiques séparées les unes des autres.
Voyons cela!

Une opération simple à faire et qui permet de gagner beaucoup en organisation de son code est de séparer ce dernier en différents fichiers. Généralement, on fait un fichier par unités "logiques". Par exemple, imaginons que nous utilisions un composant un peu compliqué qui sert d'horloge. Ce composant peut renvoyer une date en entier, juste le jour, mois, année ou encore juste l'heure, la minute ou la seconde courante. Pour bien faire, il nous faudrait une fonction par unité de temps. On aurait ainsi au moins 6 fonctions pour récupérer heure/minutes/secondes/jour/mois/année et 6 fonctions pour les régler dans le composant. 12 fonctions + la `loop()` et le `setup()` et vous voilà avec un fichier original bien encombré! 🍊
Pour créer un nouveau fichier dans l'IDE Arduino, il suffit de cliquer sur la petite flèche en haut de l'espace d'édition du code puis ensuite de cliquer sur "Nouvel Onglet" ou "New Tab" comme mis en évidence sur la capture d'écran ci-dessous:

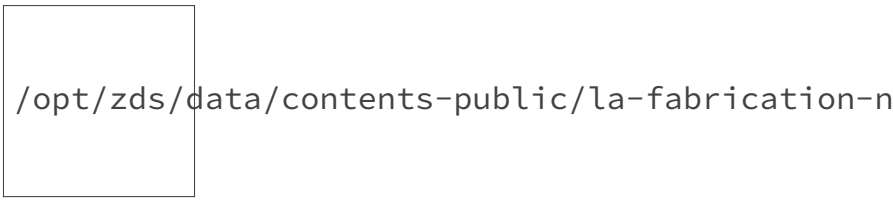


Figure I.5.21. – Bouton de nouvel onglet

I.5.7.0.0.1. Le fichier .h lorsque l'on veut séparer son code en plusieurs fichiers, il y a certaines choses à respecter. Ainsi, à chaque fois que l'on veut créer un nouveau fichier de code on ne vas pas en créer un mais deux! Le premier fichier aura l'extension `.h` signifiant **header**, c'est ce que nous allons voir maintenant.
Ce fichier va regrouper les **prototypes** des fonctions ainsi que les définitions de structures ou de classes mais nous verrons cela après.
Le prototype d'une fonction représente un peu un contrat. Il va définir le nom de la fonction, ce qui rentre à l'intérieur (les paramètres) et ce qui en sort (la variable de retour). Ainsi, votre programme principal aura une idée de comment fonctionne *extérieurement* votre fonction. Un peu comme s'il s'adressait à une boîte noire.
Si l'on devait écrire l'exemple ci-dessus on pourrait avoir le contenu de fichier suivant:
horloge.h

```
char getHeure();
char getMinute();
char getSeconde();
char getJour();
char getMois();
char getAnnee();

void setHeure(char val);
void setMinute(char val);
void setSeconde(char val);
void setJour(char val);
void setMois(char val);
void setAnnee(char val);

void afficherDate();
```

```
16 void afficherHeure();
17 void afficherDateHeure();
```

Comme vous pouvez le voir, avec ces définitions on peut savoir ce qu'est supposée faire la fonction grâce à son nom et le type de variable qu'elle manipule en entrée et en sortie.
Bien, maintenant passons à la suite pour voir où et comment implémenter ces fonctions.

I.5.7.0.0.2. Le second fichier .cpp {#fichiercpp} Le second fichier que nous allons créer sera avec une extension .cpp (pour C plus plus ou C++). Il regroupera le code à proprement parler, l'implémentation de vos fonctions. C'est ici que vous allez écrire le contenu de vos fonctions, ce qui est censé se passer à l'intérieur de ces dernières.
Pour faire cela, la première étape sera d'inclure le fichier de prototypes via la commande de préprocesseur `#include`:

```
1 #include "horloge.h" // horloge.h pour notre exemple
```

Cette ligne doit être la **première** de votre fichier .cpp et elle ne prend pas de `;` à la fin.
Une fois cela fait, il va falloir taper le code de vos fonctions.
Pour le besoin de l'exercice, je vais me contenter d'écrire des instructions bidons. Dans la vraie vie de tous les jours, vous auriez bien sûr fait un joli code pour communiquer avec un module où je ne sais quoi encore bien sûr! 🍍
horloge.cpp

```
1  /* fichier horloge.cpp */
2  #include "horloge.h"
3
4  char getHeure() {
5      Serial.println("getHeure");
6      return 0;
7  }
8
9  char getMinute() {
10     Serial.println("getHeure");
11     return 0;
12 }
13
14 char getSeconde() {
15     Serial.println("getHeure");
16     return 0;
17 }
18
19 char getJour() {
20     Serial.println("getHeure");
21     return 0;
22 }
23
24 char getMois() {
```

```
25      Serial.println("getHeure");
26      return 0;
27  }
28
29  char getAnnee() {
30      Serial.println("getHeure");
31      return 0;
32  }
33
34
35  void setHeure(char val) {
36      Serial.print("setHeure : ");
37      Serial.println(val, DEC);
38  }
39
40  void setMinute(char val) {
41      Serial.print("setMinute : ");
42      Serial.println(val, DEC);
43  }
44
45  void setSeconde(char val) {
46      Serial.print("setSeconde : ");
47      Serial.println(val, DEC);
48  }
49
50  void setJour(char val) {
51      Serial.print("setJour : ");
52      Serial.println(val, DEC);
53  }
54
55  void setMois(char val) {
56      Serial.print("setMois : ");
57      Serial.println(val, DEC);
58  }
59
60  void setAnnee(char val) {
61      Serial.print("setAnnee : ");
62      Serial.println(val, DEC);
63  }
64
65
66  void afficherDate() {
67      Serial.println("afficherDate");
68  }
69
70  void afficherHeure() {
71      Serial.println("afficherHeure");
72  }
73
74  void afficherDateHeure() {
```

```
75      Serial.println("afficherDateHeure");
76  }
```

I.5.7.0.0.3. Lier nos fichiers au programme principal

Vos définitions sont écrites et vos fonctions sont implémentées? Il ne reste plus qu'à les ajouter à votre programme principal! C'est en fait très simple vous allez voir.
Tout d'abord, il va falloir s'assurer que vos fichiers .h et .cpp sont dans le même dossier que votre .ino où se trouve votre fichier de programme Arduino.
Comme ceci:

FIGURE I.5.22. – Les fichiers sont dans le même dossier

C'est bon?
Bien, il ne reste qu'une seule chose à faire, l'inclure dans le programme. Pour cela c'est tout bête, il suffit d'ajouter la ligne `#include "horloge.h"` en haut de votre fichier.
Et voilà! Il ne vous reste plus qu'à faire des appels tout simples à vos fonctions perso dans le programme (`setup` ou `loop` ou où vous voulez!).
Maintenant, quand vous allez compiler, le compilateur va aller chercher le fichier pointé par le include, le compiler puis le lier dans votre programme principal.
Il peut arriver que le lien avec les symboles/bibliothèques Arduino ne se fasse pas correctement. Dans ce cas là, rajoutez l'include suivant au début de votre .h ou .cpp: `#include "Arduino.h"`
Séparer son code en fichiers est important pour facilement s'y retrouver, j'espère que vous l'avez bien compris. Une fois cette étape faite, vous devriez y voir plus clair dans vos gros programmes.
Les plus aguerris d'entre vous qui connaissent le C++ peuvent même coder en C++ pour créer des classes et ainsi pousser l'organisation encore plus loin!

Deuxième partie

Modélisation

II.1. Semaine 6 : Modélisation 2D

Introduction

Avant d'imprimer des objets, il faut les modéliser Bienvenue dans cette 6ème semaine du MOOC "La Fabrication Numérique". Cette semaine marque le début du **module 2** consacré aux machines à commandes numériques.
Avant de parler de ces machines et de leur fonctionnement, nous avons souhaité vous parler d'une phase importante de la création d'un objet: **la modélisation**! En effet, pour réaliser une pièce avec une fraiseuse ou une imprimante 3D, la première étape est de modéliser cette pièce grâce à un logiciel dédié et ainsi produire un fichier informatique compréhensible pour la machine cible. Nous verrons donc cette semaine au travers de trois vidéos quel vocabulaire, quels outils et quelles techniques sont nécessaires pour produire des modèles 2D exploitables par les découpeuses laser.
En parallèle de ces vidéos introductives, nous continuons à vous proposer des cours sur les concepts de base du **prototypage électronique** et du développement **Arduino**.
Bonne semaine!

II.1.1. Corrigé du TP 4

II.1.1.0.1. Corrigé du TP 4 : Feu bicolore et barrière

Voici la correction du TP de la semaine dernière qui reprend des éléments du cours sur les librairies logicielles .

II.1.1.0.1.1. Code Voici une des solutions possibles pour répondre au problème dans l'état actuel de nos connaissances:

Contenu masqué n°2

II.1.1.0.1.2. Montage

FIGURE II.1.1. – Montage de correction du TP 4

II.1.1.0.1.3. Schéma

FIGURE II.1.2. – Schéma électronique de correction du TP 4

Pour réaliser ce montage, vous avez besoin de:
- Un Arduino
- Une platine de prototypage
- Un câble USB
- Une résistance de 10kΩ
- Deux résistances de 220Ω
- Des fils de prototypage
- Une photorésistance
- Un servomoteur
- Un bouton possoir
- Une LED rouge
- Une LED verte
- Du temps

Notre correction s'arrête là. Pour des éclaircissements, nous vous invitons à poser vos questions sur le forum et à reparcourir le cours.
On vous laisse avec un montage très sympa réalisé par albert (vidéo postée sur la communauté Google+) :

ÉLÉMENT EXTERNE (VIDEO) —
Consultez cet élément à l'adresse https://www.youtube.com/embed/WwkhKg_eWWQ?feature=oembed.

II.1.2. Introduction à la modélisation 2D

II.1.2.0.1. La modélisation 2D

ÉLÉMENT EXTERNE (VIDEO) —
Consultez cet élément à l'adresse https://geo.dailymotion.com/player.html?video=x2kgzt9&.

Merci d'avoir regardé cette vidéo. Celle-ci constitue une brève introduction à la modélisation 2D. La suite consiste à l'installation d'Inkscape , un logiciel de dessin vectoriel qui va nous permettre de dessiner les formes qui seront ensuite transformées en instructions machine compréhensibles par une découpeuse laser, une fraiseuse numérique…
Pour utiliser Inkscape, nous vous invitons à regarder la vidéo ci-dessous. Vous pouvez télécharger le texte de la vidéo ici . Aussi, il est fortement recommandé de lire les références proposées ici:

II.1.2.0.1.1. Références

- Guide d'installation du logiciel Inkscape (en anglais)
- Guide du logiciel Inkscape par Tavmjong Bah (traduit en français par Loïc Guégant)
- Manuel complet consacré à Inkscape par l'équipe de FlossManuals

Merci à Laurent Mattlé pour ce cours!

II.1.2.0.1.2. Initiation Inkscape

Élément externe (Video) —
Consultez cet élément à l'adresse https://geo.dailymotion.com/player.html?video=x2kgzsw&.

II.1.3. Les Diodes (suite)

II.1.3.0.1. Les Diodes, suite

Nous avons déjà rencontré la diode, le plus simple des semi-conducteurs. Voici quelques éléments de plus à son sujet, et aussi quelques cousins plus ou moins éloignés.
Un petit rappel du symbole électrique :

Figure II.1.3. – Symbole de la diode

Quand l'anode se trouve à +0,7v par rapport à la cathode, la diode se met en conduction. Il ne faut pas laisser passer trop de courant, pour les 1N4148 c'est de l'ordre de 200mA maximum, pour un 1N4001 c'est de l'ordre de 1A. Au delà, les diodes surchauffent et elles se détruisent.
Dans le sens inverse (c'est-à-dire la cathode positive par rapport à l'anode) le courant ne passe pas, ou pas beaucoup : quelques nanoampères au plus à des tensions assez élevées. Notons aussi que les diodes ne supportent pas des tensions inversées sans limites : les petites 1N4148, par exemple, ne supportent pas plus qu'une centaine de volts…

II.1.3.0.2. Les DEL ou LEDs

C'est le truc qui s'allumait dans le tout premier montage. Les DEL sont fabriquées dans toutes sortes de formes et toute une gamme de couleurs – même des couleurs 'invisibles' par l'œil humain comme l'infra-rouge (pour les télécommandes et les détecteurs optiques), et l'ultra-violet.
Le symbole :

FIGURE II.1.4. – Symbole de la led

Comme pour les diodes classiques ; les DEL exhibent une chute de tension. Cette tension est plus élevée, de l'ordre de quelques volts, et dépend du type et de la couleur. Il faut se référer aux documents techniques, ou au moins aux infos données à titre d'indication sur les sites de vente de composants électroniques, afin de savoir quelle est la chute de tension pour une DEL particulière. Exemple, le site Gotronic :

- une DEL rouge : $V_F = 1.6V$;
- une DEL blanche : $V_F = 3.5V$

? Pourquoi c'est si important ?

Parce que, comme pour les diodes classiques, si on laisse passer trop de courant la DEL est détruite très rapidement. Et comment limiter ce courant ? Allez, réfléchissez bien…
Oui, avec une résistance, bien sûr ! (Sinon, pourquoi avoir fait tout ce chemin, hein ?)
Et pour calculer la résistance pour limiter l'intensité, il nous faut deux valeurs : l'**intensité maximale**, et la **chute de tension**.
Prenons des exemples :

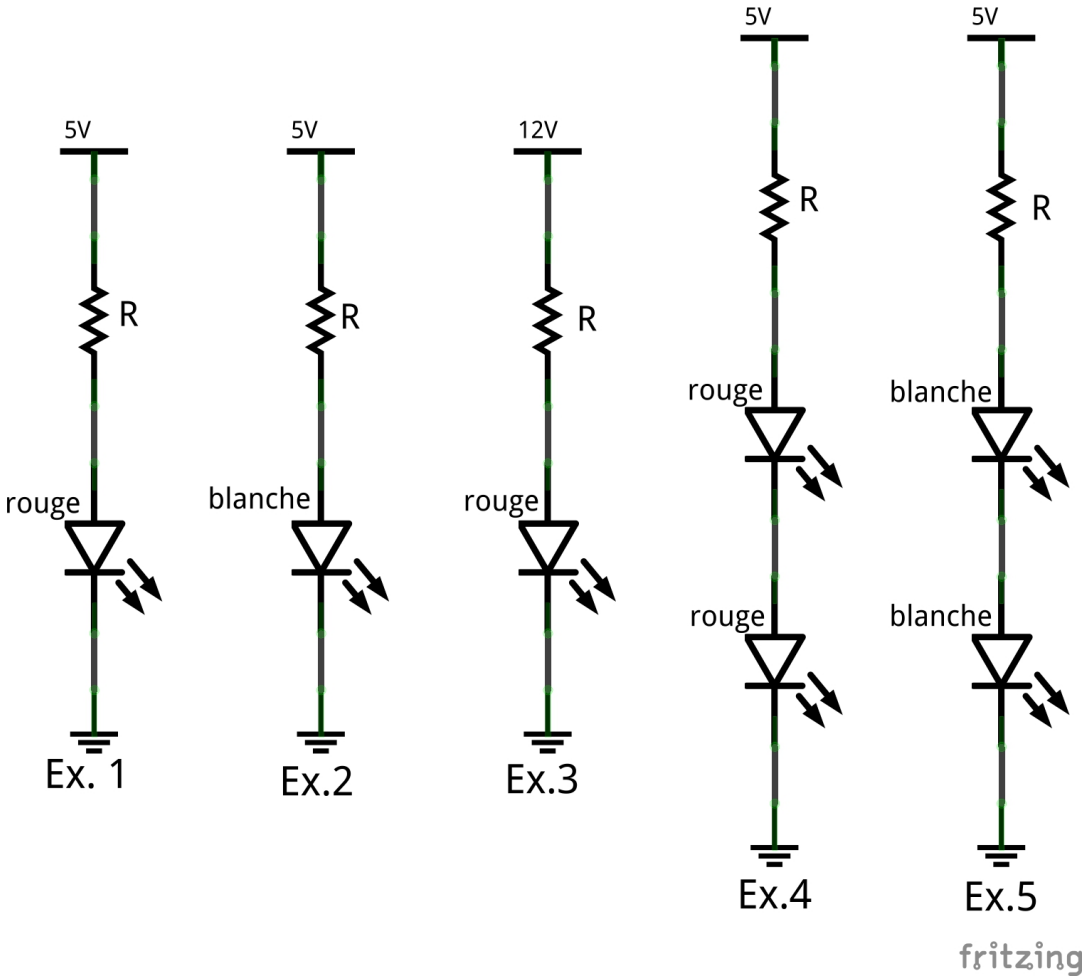


FIGURE II.1.5. – Quelques exemples de montage

II.1.3.0.2.1. Ex. 1. Pour calculer R (en se rappelent que $R = \frac{U}{I}$) :

— $DEL_{rouge} V_F = 1.6V$;
— $I_{Max} = 20mA$ (intensité maximale);
— $V_{Alim} = 5V$.

Calculer R :

Ce circuit est alimenté par une tension de $5V$. Allumée, la LED provoque une chute de tension de $1.6V$. Il faut calculer la résistance pour qu'elle fasse 'chuter' $5 - 1.6 = 3.4V$. On va appeler cette tension V_R

Pour limiter l'intensité à $20mA$, utilisant $R = \frac{U}{I}$; $R = \frac{V_R}{I_{Max}} = \frac{3.4V}{0.02A} = 170\Omega$

II.1.3.0.2.2. Ex. 2.

— $DEL_{blanche} V_F = 3.5V$;
— $I_{Max} = 20mA$ (intensité maximale);
— $V_{Alim} = 5V$.

Calculer R :

La chute de tension aux bornes de R : $V_R = V_{Alim}$–$V_F = 5$–$3.5 = 1.5V$

Pour limiter l'intensité à 20mA, $R = \frac{V_R}{I_{Max}} = \frac{1.5V}{0.02A} = 75\Omega$

II.1.3.0.2.3. Ex. 3.

— $V_{Alim} = 12V$.

Calculer R :
La chute de tension aux bornes de R : $V_R = V_{Alim}–V_F = 12–1.6 = 10.4V$
Pour limiter l'intensité à 20mA, $R = \frac{V_R}{I_{Max}} = \frac{10.4V}{0.02A} = 520\Omega$

II.1.3.0.2.4. Ex. 4

— $DEL_{rouge} V_F = 1.6V$;
— $I_{Max} = 20mA$ (intensité maximale).

Le courant que traverse la 1ère DEL traverse aussi la 2ème, donc I_{Max} reste inchangé.
Calculer R :
La chute de tension aux bornes de R : $V_R = V_{Alim}–2 \times V_F = 5–2 \times 1.6 = 5 - 3.2 = 1.8V$
Pour limiter l'intensité à 20mA, $R = \frac{V_R}{I_{Max}} = \frac{1.8V}{0.02A} = 90\Omega$

II.1.3.0.2.5. Ex. 5

— $DEL_{blanche} V_F = 3.5V$

Calculer R :
La chute de tension aux bornes de R : $V_R = V_{Alim}–2 \times V_F = 5 - 7 = -2V$
Aïe! Ça ne marche pas : 5v ce n'est pas suffisant pour allumer les deux DEL en série !
Comment contourner le problème ? On peut toujours mettre les 2 DEL en parallèle :

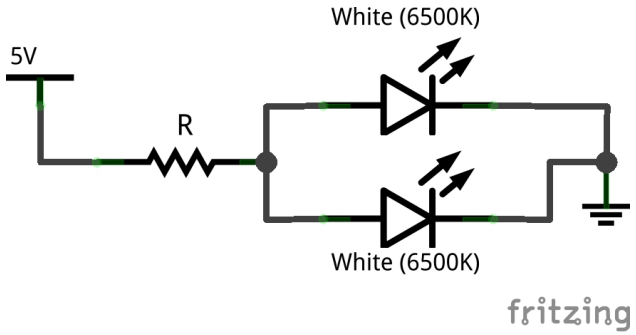


FIGURE II.1.6. – Deux DEL en parallèle

Maintenant le courant qui traverse R est la somme de l'intensité à travers chaque DEL.
Donc I_{Max} devient 40mA. La chute de tension est la même pour chaque DEL, donc $V_R = V_{Alim}–V_F = 5–3.5 = 1.5V$
Et donc $R = \frac{V_R}{I_{Max}} = \frac{1.5V}{0.04A} = 37.5\Omega$
Attention cependant : on fait passer deux fois plus de courant à travers la résistance. Tout courant électrique à travers une résistance génère de la chaleur. Les résistances standard les moins chères sont des résistances 0,25W. Il faut vérifier si on dépasse 0,25W.
Pour calculer la puissance perdue (chaleur) par les 40mA à travers la résistance il nous faut une autre formule – encore très simple : $P = I \times U$
avec:

— P: Puissance en Watts (la chaleur)
— I: Intensité (en Ampères)
— U: Tension (en Volts)

Dans ce cas nous avons $I = 0.04A$ et $U = 1.5V$ donc $P = 0.06W$. Ouf !
À noter que, si on veut allumer plusieurs LED/DEL ensembles, et que nous n'avons pas assez de tension pour les câbler en série, il vaut mieux donner une résistance en série pour **chaque** DEL .
Merci à Glenn Smith pour ce cours!

II.1.4. Arduinomètre : Thermomètre avec diode pour sonde

II.1.4.1. Première partie

Nous avons abordé le sujet des diodes du point de vue de leur rôle principal, c'est-à-dire de ne laisser passer le courant que dans un sens. Il existe plus de types de diodes qu'il y a de jours dans l'année : des diodes pour des tensions élevées, des diodes qui supportent les intensités élevées, des diodes rapides et j'en passe… Comme toujours il faut chercher la documentation technique pour 'sa' diode afin de savoir comment l'utiliser.
Et dans cette documentation on trouve parfois des informations intéressantes… Nous avons déjà parlé beaucoup de la chute de tension pour les LEDs – or une LED est quand-même une diode.
Cette chute de tension est de l'ordre de 0,5 à 0,7v pour la plupart des diodes. Mais si on creuse dans la documentation, les fabricants avouent parfois que tout n'est pas aussi simple.
Pour beaucoup de diodes, la chute de tension n'est pas fixe – et peut varier en fonction de plusieurs paramètres. Regardons cette courbe, tirée de la documentation **1N4148** de chez ***Fairchild Semiconductors*** :

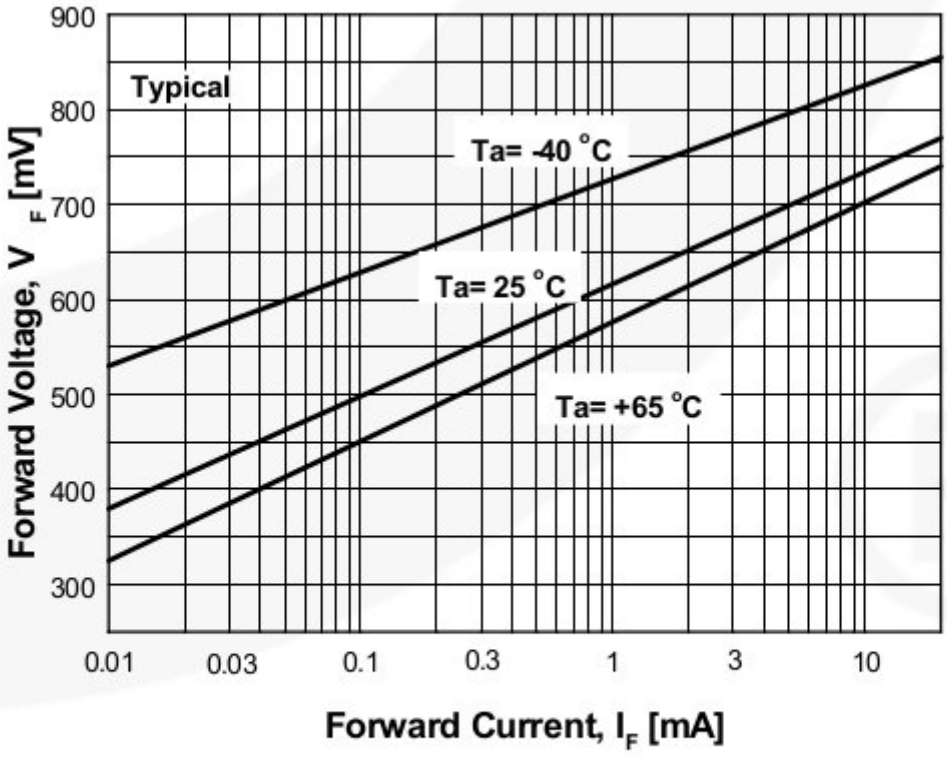


Figure 6. Forward Voltage vs. Ambient Temperature
V_F - 0.01 - 20 mA (- 40 to +65°C)

FIGURE II.1.7. – Chute de tension en fonction du courant et de la température

Cette courbe nous renseigne sur deux phénomènes :
- la chute de tension varie en fonction de l'intensité (dans des limites bien précises) ;
- *la chute de tension varie en fonction de la température de la diode.*

La fonction Température/Intensité n'est pas très évidente avec cette courbe, mais si on prend quelques valeurs en suivant la ligne verticale qui correspond à une intensité de 500 uA, nous avons ceci:

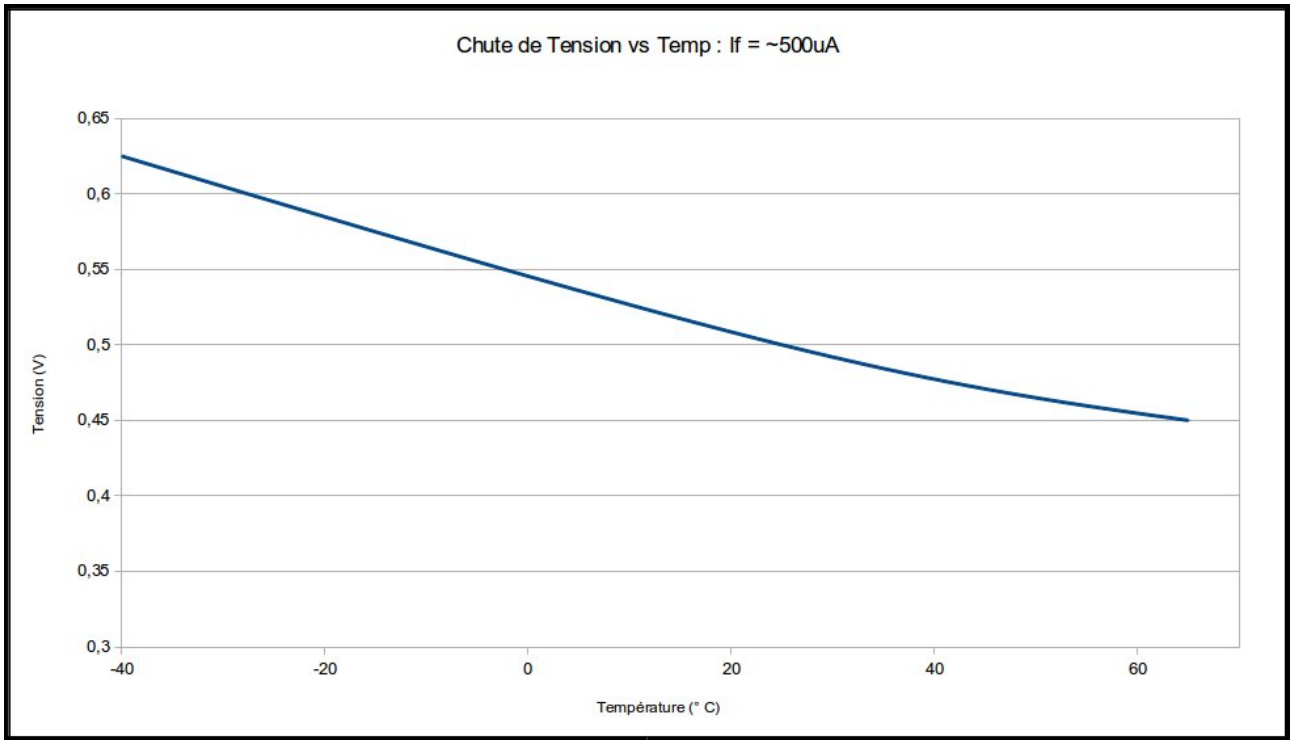


FIGURE II.1.8. – Chute de tension en fonction de la température

C'est une fonction assez linéaire d'environ -2mV (-0,002v) par °C.
Donc, il devrait être possible de fabriquer un thermomètre Arduino avec une diode pour sonde.
Comment faire ?
Pour les plus impatients d'entre vous on va tout de suite à l'essentiel :

II.1.4.1.0.1. Le schéma

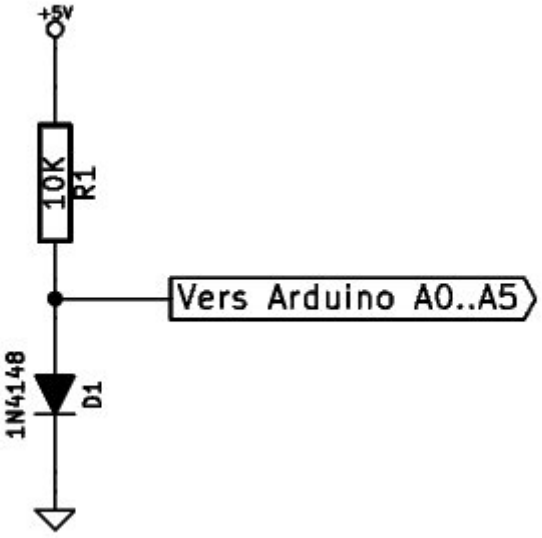


FIGURE II.1.9. – Branchement de la diode

C'est tout ? Eh oui !
D'après la courbe ci-dessous on voit qu'il faut une faible intensité pour 'voir' le phénomène.
Dans ce montage, l'intensité à travers la diode, c'est combien ? (réponse en bas)
À cette faible intensité la diode ne risque pas de se chauffer elle-même et on peut lire la chute de tension directement par l'Arduino.
Mais il reste un problème de taille : comment calibrer ce montage ?
Deux possibilités…

1. Calibration avec au moins deux températures connues (et donc un autre thermomètre);
2. Approximation et triche…

(Moi, j'adore la deuxième !)
On va les aborder dans le désordre, parce que l'option 2 est rapide et facile à mettre en œuvre.

II.1.4.1.0.2. Approximation et triche Il nous faut au moins un point de repère – une valeur connue de chute de tension pour une température donnée. À partir d'un point il est possible de travailler en 'extrapolant' (on va faire comme si la variation de chute de tension est *exactement* -2mV/°C). C'est de l'approximation.
Si on regarde à nouveau la courbe de chute de tension, on voit que cette chute de tension est aussi fonction de l'intensité. Cela veut dire qu'on peut varier la valeur de tension 'lue' par l'Arduino simplement en variant l'intensité qui traverse la diode. Chouette ! On sait maintenant (car nous avons avalé la loi d'ohm en long et en travers) qu'on peut varier l'intensité en variant la résistance en série. Si on remplaçait la résistance de 10 kOhms par une résistance variable (potentiomètre)? Comme ceci :

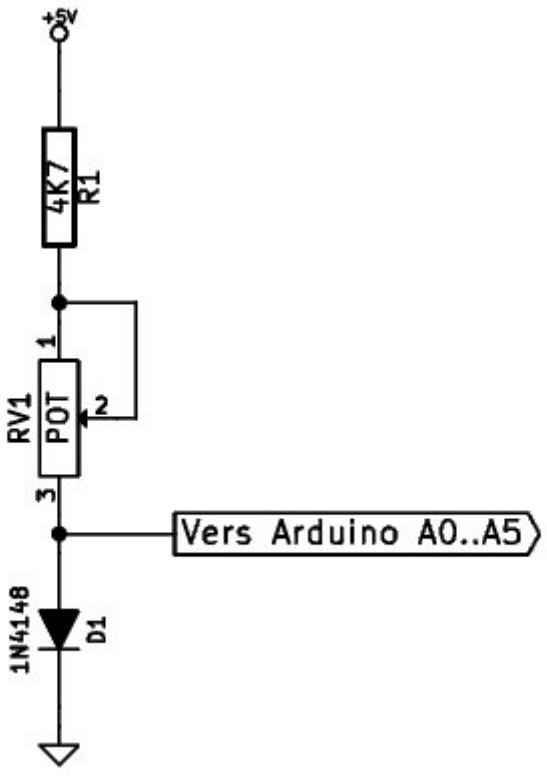


FIGURE II.1.10. – La diode en série avec un potentiomètre

? Encore une résistance fixe en série ! Pourquoi ?

Si on ne mettait qu'une résistance variable, et si on réglait cette résistance à bout de course cotée +5v notre pauvre diode se trouverait branchée directement entre le +5v et 0V/GND – sans rien pour limiter l'intensité. Elle ne durerait pas longtemps comme ça. J'ai mal juste en l'imaginant!
Donc la résistance de 4,7 kOhms est là pour protéger notre diode. L'intensité maximum sera de combien d'ampères ? (réponse en bas)
Maintenant il suffit de régler la résistance variable jusqu'à avoir une chute de tension bien précise.
Exemple : regardez la 1ère courbe : une valeur de chute de tension avec l'intensité d'environ 1mA et une température d'environ 20° c'est 600mV.
Voici notre montage sur une platine :

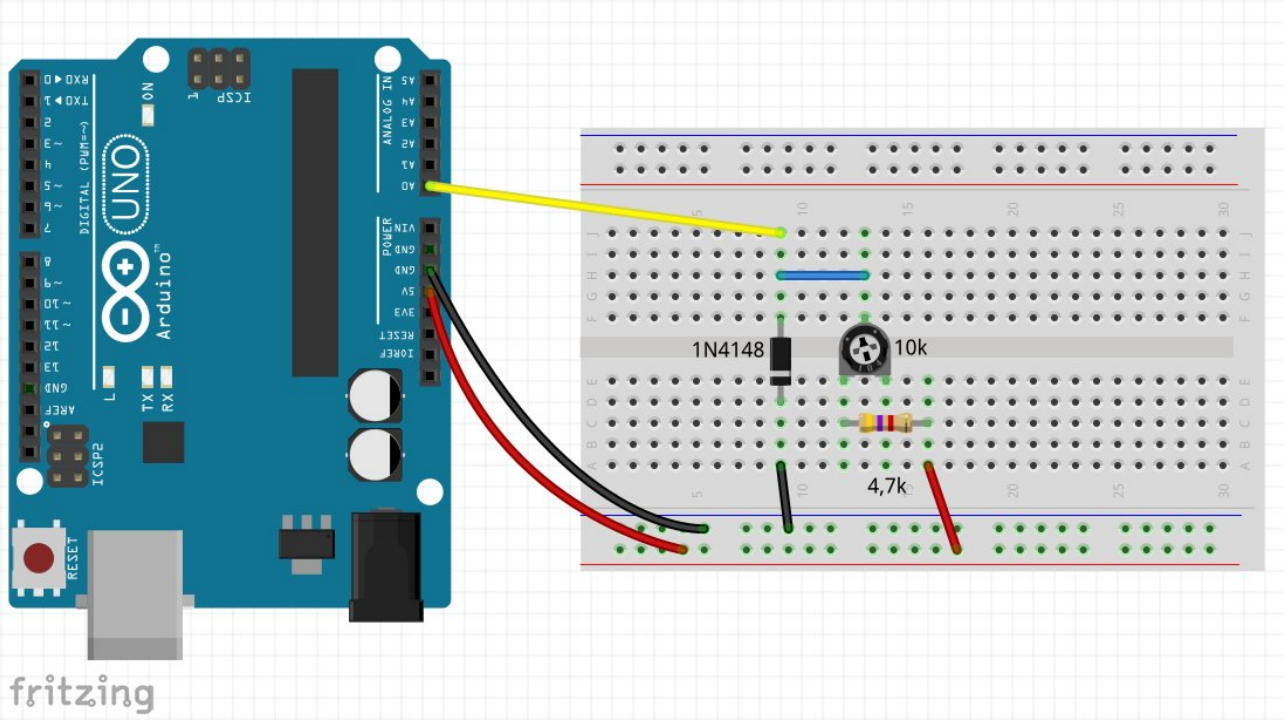


FIGURE II.1.11. – Montage de notre capteur de température fait maison

On va se mettre dans une pièce où on sait que la température est de 20° (le salon), on va laisser notre Arduino et la platine d'expérimentation s'acclimater pendant un quart d'heure (sur la table du salon : ça fait réagir la famille et donne un sujet de discussion) et on va régler la résistance variable pour que la valeur retournée par `analogRead()` corresponde à 600mV.
Voici un bout de logiciel pour le réglage :

```
/*
  Montage pour faire un thermomètre avec une diode pour sonde
  Premier partie : réglage avec potentiomètre
  La diode est montée en sens 'conduction' (cathode branche sur
    GND)
```

```
  avec une résistance fixe de 4.7k et une résistance variable de
  préférence de 10k en serie.

  On cherche à régler le montage pour que le résultat de
    analogRead();
  donne 558 (environ 600mV) a 20°C

  Avril 2014 MOOC Fabrication Numérique

*/
// Définition des broches
#define Diode A0 // broche de la diode
#define LED 13   // broche de la led

// variables:
int DiodeValue = 0;  // la valeur de la diode
float Result = 0.0;  // pour les calculs

void setup() {
  pinMode(LED, OUTPUT);
  Serial.begin(9600);
  // Réference maximum pour le convertisseur ADC = 1,1v
  //  au lieu des 5v par défaut.
  analogReference(INTERNAL);
}

void loop() {
  DiodeValue = analogRead(Diode); // Lire la valeur
  Serial.print("Valeur : ");
  Serial.print(DiodeValue);
  // allumer la LED si la valeur est 558
  if (DiodeValue == 558 ) {
    digitalWrite(LED, HIGH); }
  else {
    digitalWrite(LED, LOW);
  }
  // conversion de la valeur en tension.  1,075 = 1,1v/1023
  Result = DiodeValue * 1.075;
  Serial.print("\t environ ");
  Serial.print(Result);
  Serial.println(" mV");
  delay(2000);
}
```

Quelques détails sur ce code :

Les nouvelles instructions:

— `analogReference(INTERNAL)` : Configure la tension de référence utilisée avec les entrées analogiques.

II.1.4.1.0.3. Tension de référence La chute de tension que nous voudrions mesurer est d'environ un demi-volt, et toujours inférieure à 1V (c'est le fabricant qui nous le dit). Par défaut, le Convertisseur Analogique -> Numérique (CAN ou ADC en anglais) de l'Arduino nous donne une valeur de 1023 pour une tension en entrée de 5v. Cela veut dire qu'une valeur de 1 dans cette échelle équivaut (5/1023) = 0,0049v ou presque 5mV. La chute de tension que nous allons mesurer varie de 2mV/°C et donc la résolution de lecture sera de presque +/- 3°C…
Mais l'Arduino possède des talents cachés… On peut lui demander, au lieu des 5v par défaut, de se référer à une valeur de tension de 1,1v avec la fonction `analogReference(INTERNAL)` Maintenant une valeur de 1023 retournée par `analogRead()` correspond à 1,1v. Ce qui nous donne une résolution de lecture de $\frac{1.1}{1023} = 0.001075V$: moins de 2mV. C'est beaucoup mieux !

II.1.4.1.0.4. Suite(s) Dans la deuxième partie on va fabriquer une sonde flexible et étanche afin de pouvoir mesurer la température dans des conditions contrôlées, mais aussi afin de faciliter l'étalonnage pour augmenter la fiabilité et la précision.
En attendant, réfléchir à comment convertir notre valeur de tension en valeur de température…

- Réponse 1 : $I = \frac{U}{R} = \frac{5V}{10000=0.5mA}$ (ou 500 uA si vous préférez…)
- Réponse 2 : $\frac{5V}{4700=1.06mA}$

Merci à Glenn Smith pour ce cours!

II.1.5. Travaux pratiques

II.1.5.0.1. Tp à faire pour la semaine 7

Vous avez certainement entendu parler de la Useless Box .

Figure II.1.12. – La Useless Box en action

Cette machine actionnée par un un interrupteur comporte un bras qui vient basculer l'interrupteur dans le sens opposé rendant cette machine totalement inutile (et donc indispensable).
L'objectif de cette semaine est de dessiner avec Inkscape le bras de cette machine qui devra répondre aux impératifs fonctionnels décrits ici :

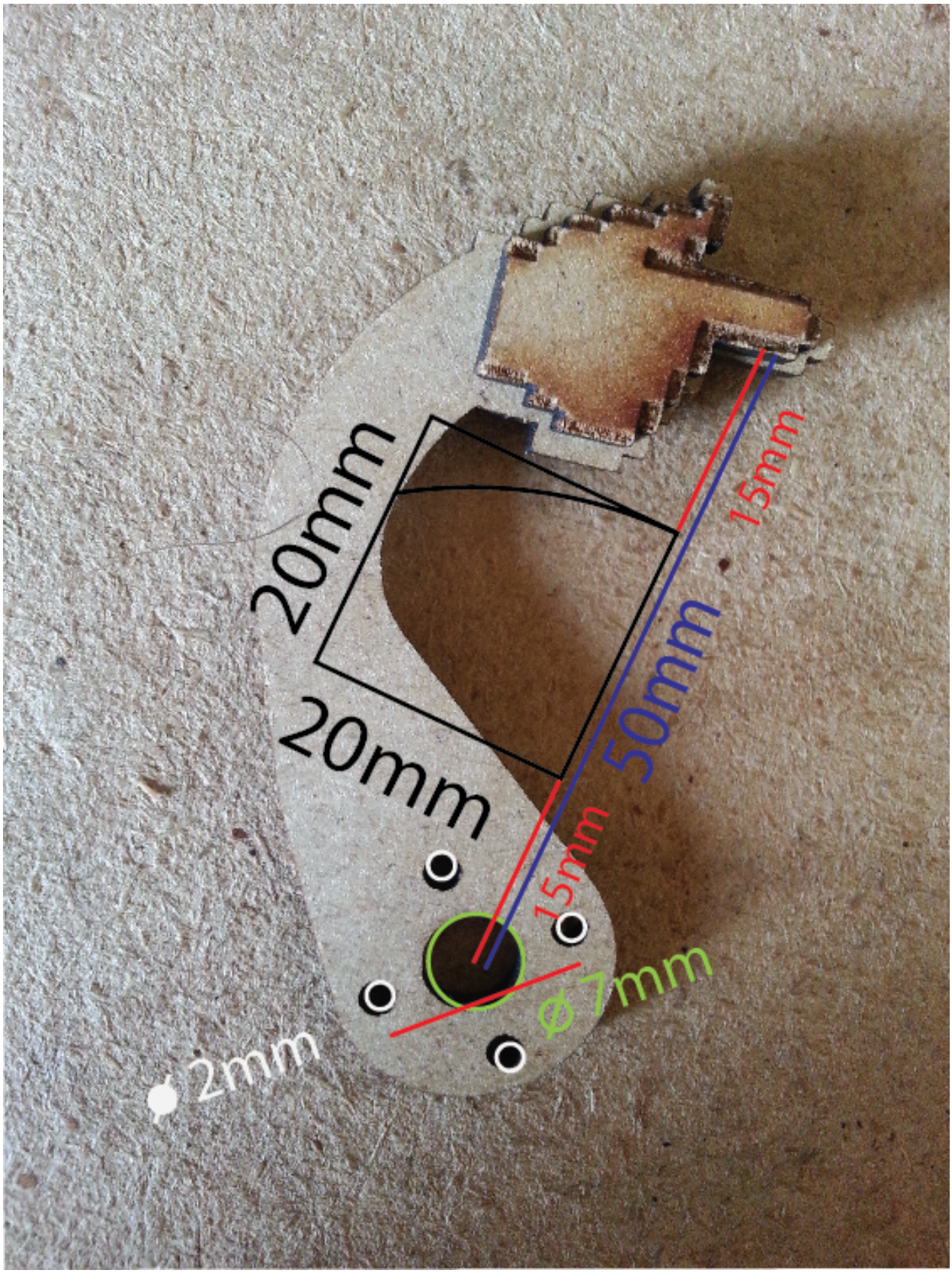


FIGURE II.1.13. – Cotes du bras à dessiner

Ce bras sera ensuite utilisé dans la vidéo sur les découpeuses laser à venir dans 3 semaines!
À très bientôt,
L'équipe du MOOC

Contenu masqué

Contenu masqué n°2

```
/*
  Feu bicolore et barrière

 TP de la semaine 5 du MOOC "La Fabrication Numerique"

 Le montage :
 * Une LED rouge sur la broche 3 en serie avec une resistance de 220Ω
 * Une LED verte sur la broche 4 en serie avec une resistance de 220Ω

 * Un servomoteur branché sur les broches 9, +5V et GND

 * Bouton poussoir branché sur la broche 2 depuis +5V
 * Une résistance de 1KΩ bracnhé sur la broche 2 depuis GND

 créé le 18 Avril 2014
 par Baptiste Gaultier

 Ce code est en CC0 1.0 Universal

 https://www.france-universite-numerique-mooc.fr/courses/MinesTelecom/04002/Trimestre_1_2014/about

 */

#include <Servo.h>

Servo servo;  // création de l'objet servo issu
              // du moule Servo


// Initialisation des constantes
const int bouton = 2;

const int ledRouge =  3;
const int ledVerte =  4;

// Déclaration des variables :
int etatBouton = 0;
int pos = 0;

// le code dans cette fonction est exécuté une fois au début
void setup() {
  // on souhaite communiquer avec l'ordinateur
```

```
  Serial.begin(9600);

  // indique que les broches des LED
  // sont des sorties :
  pinMode(ledRouge, OUTPUT);
  pinMode(ledVerte, OUTPUT);

  // indique que la broche bouton est une entrée :
  pinMode(bouton, INPUT);

  // on accroche notre servomoteur branché sur la broche 9
  servo.attach(9);

  // allume le feu rouge
  digitalWrite(ledRouge, HIGH);

  // positionne la barrière horizontalement
  servo.write(0);
}
[[secret]]
```cpp
/*
  Feu bicolore et barrière

 TP de la semaine 5 du MOOC "La Fabrication Numerique"

 Le montage :
 * Une LED rouge sur la broche 3 en serie avec une resistance de 220Ω
 * Une LED verte sur la broche 4 en serie avec une resistance de 220Ω

 * Un servomoteur branché sur les broches 9, +5V et GND

 * Bouton poussoir branché sur la broche 2 depuis +5V
 * Une résistance de 1KΩ bracnhé sur la broche 2 depuis GND

 créé le 18 Avril 2014
 par Baptiste Gaultier

 Ce code est en CC0 1.0 Universal

 https://www.france-universite-numerique-mooc.fr/courses/MinesTelecom/04002/Trimestre_1_2014/about

 */

#include <Servo.h>

Servo servo;  // création de l'objet servo issu
```
```

```
90                // du moule Servo
91
92
93  // Initialisation des constantes
94  const int bouton = 2;
95
96  const int ledRouge =  3;
97  const int ledVerte =  4;
98
99  // Déclaration des variables :
100 int etatBouton = 0;
101 int pos = 0;
102
103 // le code dans cette fonction est exécuté une fois au début
104 void setup() {
105   // on souhaite communiquer avec l'ordinateur
106   Serial.begin(9600);
107
108   // indique que les broches des LED
109   // sont des sorties :
110   pinMode(ledRouge, OUTPUT);
111   pinMode(ledVerte, OUTPUT);
112
113   // indique que la broche bouton est une entrée :
114   pinMode(bouton, INPUT);
115
116   // on accroche notre servomoteur branché sur la broche 9
117   servo.attach(9);
118
119   // allume le feu rouge
120   digitalWrite(ledRouge, HIGH);
121
122   // positionne la barrière horizontalement
123   servo.write(0);
124 }
125
126 // le code dans cette fonction est exécuté en boucle
127 void loop(){
128   // read the state of the pushbutton value:
129   etatBouton = digitalRead(bouton);
130
131   // si le bouton est appuyé
132   if (etatBouton == HIGH) {
133     // alors on envoie un message sur le moniteur série
134      Serial.print("Bouton appuye");
135
136     // puis on remonte la barrière de 90°
137     for(pos = 0; pos <= 90; pos++) {
138       servo.write(pos);
139       delay(15);
```

```
140     }
141
142     // puis on allume le feu vert durant 5 secondes
143     digitalWrite(ledRouge, LOW);
144     digitalWrite(ledVerte, HIGH);
145     delay(5000);
146
147     // et on repasse au rouge
148     digitalWrite(ledVerte, LOW);
149     digitalWrite(ledRouge, HIGH);
150
151     // enfin, on redescend la barrière
152     for(pos = 90; pos>=0; pos--) {
153       servo.write(pos);
154       delay(15);
155     }
156   }
157 }
```

Listing 11 – Un exemple de correction du TP 4

// le code dans cette fonction est exécuté en boucle
// si le bouton est appuyé if (etatBouton == HIGH) { // alors on envoie un message sur le moniteur série Serial.print(”Bouton appuye”);
// puis on remonte la barrière de 90° for(pos = 0; pos <= 90; pos++) { servo.write(pos); delay(15); }
// puis on allume le feu vert durant 5 secondes digitalWrite(ledRouge, LOW); digitalWrite(ledVerte, HIGH); delay(5000);
// et on repasse au rouge digitalWrite(ledVerte, LOW); digitalWrite(ledRouge, HIGH);
// enfin, on redescend la barrière for(pos = 90; pos>=0; pos—) { servo.write(pos); delay(15); } } }

```
1 Code: Un exemple de correction du TP 4
```

Retourner au texte.

II.2. Semaine 7 : Les découpeuses Laser

Introduction

Découpeuses laser La semaine passée, nous abordions la modélisation 2D. Il est temps cette semaine de passer du virtuel au réel avec un cours sur une machine fascinante: **la découpeuse laser**. Pour présenter cette machine, on retrouve John du LabFab de Rennes pour une vidéo où on vous montre tout ce qu'il est possible de faire avec cette machine.
En parallèle de cette vidéo d'introduction, nous revenons cette semaine sur les **boucles et fonctions.**
Bonne semaine!
L'équipe du MOOC

II.2.1. Les découpeuses Laser

Élément externe (Video) —
Consultez cet élément à l'adresse https://geo.dailymotion.com/player.html?video=x2kgzwi&.

Merci d'avoir regardé cette vidéo sur les principes de fonctionnement de la découpeuse laser.
Comme il existe de très nombreux logiciels pour passer d'un fichier informatique aux instructions machines (le fameux `.gcode` dont nous parlions les semaines passées), nous n'avons pas souhaité rentrer dans la partie logicielle des découpeuses. Nous vous invitons plutôt à visiter un FabLab qui dispose d'une machine de ce type et à regarder la vidéo ci-dessous produite par Dimitri du FabLab de Lyon (car sa vidéo est pédagogique et efficace).
Merci à John Lejeune, FabManager du FabLab de Rennes pour ce cours
Si vous voulez télécharger le pdf de la vidéo c'est par ici .

II.2.1.1. La découpe laser par Dimitri

Élément externe (Video) —
Consultez cet élément à l'adresse https://www.youtube.com/embed/tBG8asUFFoE?feature=oembed.

II.2.2. Fonctions et boucles

II.2.2.0.1. Fonctions et boucles

Comme beaucoup de questions ont été soulevées les semaines passées sur les boucles et les fonctions, nous avons jugé bon de revenir sur deux concepts importants lorsque l'on développe:

II.2.2.0.1.1. Les boucles Les boucles permettent de répéter une séquence d'instructions tant que la condition d'arrêt n'est pas vraie. Nous avions ainsi pu voir dans les semaines passées la boucle `for` . Rappelez-vous, elle nous permettait de faire bouger notre servomoteur d'une position de départ vers une postion d'arrivée avec un pas donné.
Sachez qu'il existe d'autres boucles comme `while` . Cette boucle **tant que** boucle sans fin, et indéfiniment, jusqu'à ce que la condition ou l'expression entre les parenthèses `()` devienne fausse. N'oubliez donc pas de modifier doit modifier la variable testée, sinon la boucle while ne se terminera jamais!
Si nous reprenons l'exemple `Sweep` que nous avions vu en semaine 5 et que nous souhaitons remplacer les boucles for par des boucles while. Voici ce que ça pourrait donner:

II.2.2.0.1.2. Avec `for`

```
for(pos = 0; pos < 180; pos += 1)
{
  myservo.write(pos);
  delay(15);
}
for(pos = 180; pos >= 1; pos -=1 )
{
  myservo.write(pos);
  delay(15);
}
```

II.2.2.0.1.3. Avec `while`

```
pos = 0;
while(pos < 180)
{
```

```
  myservo.write(pos);
  delay(15);
  pos += 1;
}
pos = 180;
while(pos >= 1)
{
  myservo.write(pos);
  delay(15);
  pos-=1;
}
```

II.2.2.0.1.4. Les fonctions Quoi de mieux que citer le wiki ! Une fonction permet d'encapsuler du code avec une petite latitude de paramétrage. L'intérêt est que cela évite de dupliquer des instructions et qu'en découpant le code en série de fonctions celui-ci devient plus clair et plus compréhensible.
Une fonction peut accepter des paramètres en entrée et renvoyer une valeur en sortie. Aussi bien les paramètres que la valeur retournée sont facultatifs. La déclaration classique d'une fonction se fait comme ceci :

```
typeSortie nomFonction(type1 paramètre1, type2 paramètre2)
{
  // instructions...
}
```

où `typeSortie` et `type*` sont des types (comme par exemple `int` pour un nombre entier). Nous allons prendre comme exemple une fonction qui allume une LED pendant une durée exprimée en millisecondes. Cette fonction ne retournera pas de valeur mais prendra deux paramètres: la broche où la LED est raccordée et la durée pendant laquelle la LED devra être allumée. Cela se traduit de la façon suivante:

```
void allumerLED(int broche, int duree)
{
    digitalWrite (broche, HIGH);
    delay (duree);
    digitalWrite (broche, LOW);
}
```

Si on reprend le TP n°2 mais en utilisant cette nouvelle fonction, `loop()` se réduit à:

```
void loop ()
{
    allumerLED(verte, 3000);
    allumerLED(orange, 1000);
    allumerLED(rouge, 3000);
```

```
}
```

Même avec cet exemple très simple l'intérêt est assez évident et immédiat.
Il est également possible d'utiliser une fonction pour effectuer une opération et de retourner un résultat. Pour retourner une valeur l'instruction `return` est utilisée, par exemple pour une fonction qui réalise une addition:

```
int addition(int a, int b)
{
    return (a+b);
}
```

Merci à fb251, DDR et mary1in pour avoir rédigé et corrigé ce cours sur le wiki .

II.2.3. Les transistors

II.2.3.1. Les Transistors (enfin)

Sujet très, très vaste : il y a pléthore de bouquins sur les transistors et leur mise en œuvre…
Ici (car moi je suis loin de ***tout*** connaître à ce sujet), nous allons rester simple et nous abordons quelques petits points au sujet de quelques types de transistors.
Nous allons parler principalement de deux familles de transistors : les transistors bipolaires (les transistors classiques) et les transistors à effet de champ (”Field-Effect Transistors” en anglais – d'où leur petit nom de FET).
Dans nos montages autour des microcontrôleurs, l'utilisation des transistors se limite souvent à alimenter (marche / arrêt) quelque chose qui a besoin de plus de puissance que celle fournie directement par les broches du microcontrôleur (qui fournissent de l'ordre de 20mA maximum), où l'alimentation des composants fonctionnent à des tensions autres que les 3,3v ou 5v disponibles en sortie des microcontrôleurs. Ex. des moteurs, des LEDs de puissance, une ampoule…
Nous allons regarder comment faire. D'abord, quelques symboles :

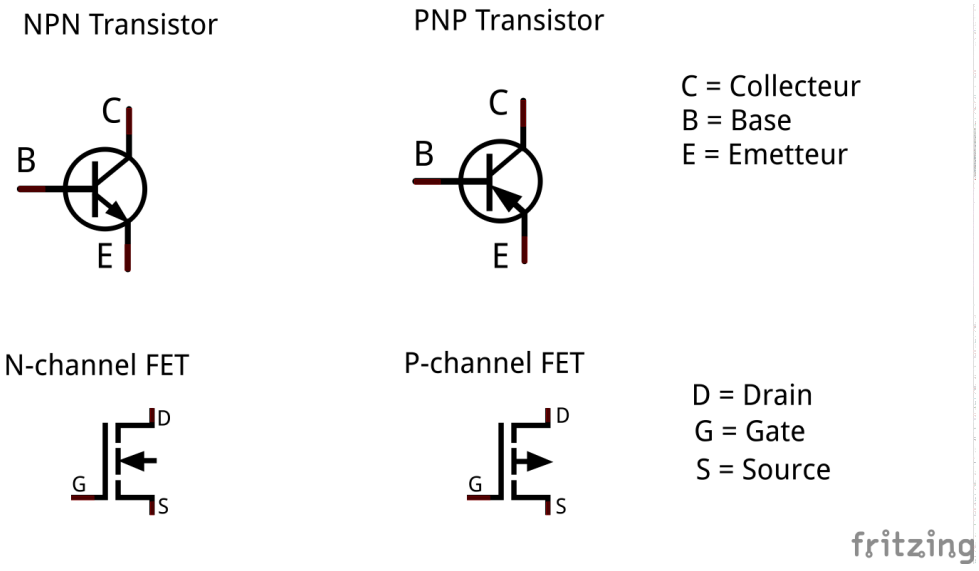


Figure II.2.1. – Symbole des transistors bipolaires en MOSFET

Notez bien que, comme pour les diodes, la flèche indique le sens du courant. Dans les deux familles on trouve une variante de type 'n' (ou NPN) et une variante de type 'P' (ou PNP). Pour l'instant on va regarder la variante 'n' car c'est la plus pratique pour un grand nombre de nos montages.

? Comment ça marche, alors ?

Les transistors bipolaires fonctionnent un peu comme une vanne pilotée : un petit ***courant*** qui traverse les bornes Base-Emetteur est capable de 'contrôler' un plus gros courant qui traverse les bornes Collecteur-Emetteur – avec deux conditions qui nous intéressent beaucoup : le cas où le courant C-E est bloqué (condition arrêt), et le cas où l'intensité C-E est ou maximum (condition marche).
Pour les FET le fonctionnement est similaire, mais c'est la ***tension*** Gate-Source qui contrôle l'intensité Drain-Source.
On va sauter tout de suite dans les montages, espérant que nous comprendrons quelque chose à la fin !

II.2.3.1.1. Transistor NPN comme interrupteur (de puissance)

C'est peut-être l'utilisation première des transistors autour des microcontrôleurs. Une fois n'est pas coutume : le montage sur la 'planche à pain'...

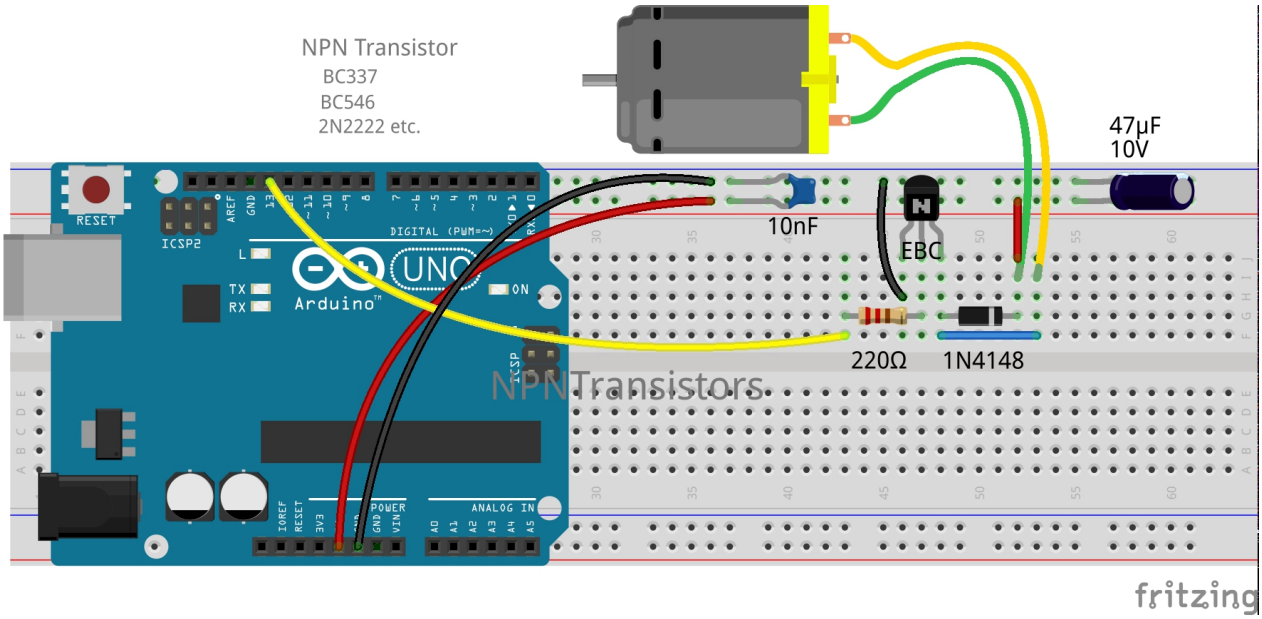


FIGURE II.2.2. – Montage d'un transistor

Notez que j'ai mis un transistor NPN dans Fritzing, sans vérifier si l'orientation "EBC" correspond à l'orientation standard pour ces transistors. Il faut vérifier avec la bonne fiche technique.
Voici le schéma...

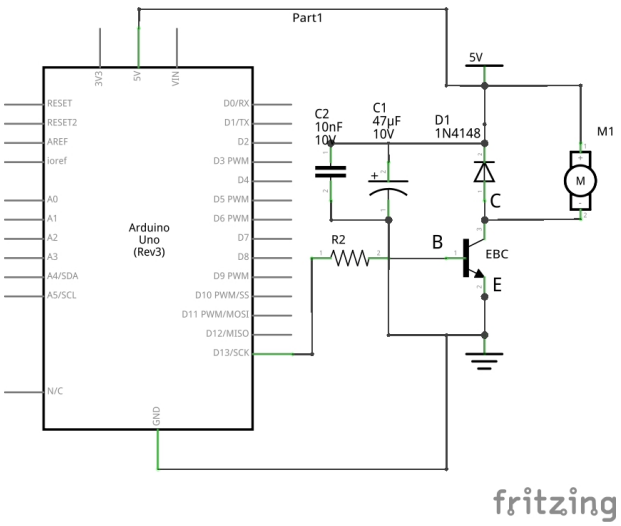


FIGURE II.2.3. – Branchement d'un transistor

Quelques questions pour voir si vous avez bien suivi jusque-là :

?

1. Pourquoi les deux condensateurs ?
2. A quoi sert la diode en parallèle avec le moteur ?

Réponses :

Contenu masqué n°3

Retournons à nos moutons : ce montage de transistor. Si la sortie (pin 13) est maintenue au niveau "low" (0v) la base et l'émetteur du transistor se trouvent au même potentiel, pas de courant B-E et le transistor reste bloqué : pas de courant entre collecteur et émetteur et le moteur ne tourne pas.
Si maintenant la sortie est basculée "high", un courant traverse la résistance R2 et la jonction B-E du transistor. L'intensité B-E dépend de la tension appliquée (ici c'est 5v), et la valeur de la résistance. Si l'intensité est suffisante, le transistor entre en conduction et la tension entre les bornes C-E chute pour s'approcher de 0v. Le moteur se trouve branché entre le 5v et (presque) 0v et va donc tourner.
Regardons cela de plus près avec un exemple concret - le transistor BC337 :

1. 0,7v (c'est pas si difficile – il faut suivre un peu…)

II.2.3.1.2. Fiche technique : Transistor BC337

CHARACTERISTICS

T_j = 25 °C unless otherwise specified.

SYMBOL	PARAMETER	CONDITIONS	MIN.	TYP.	MAX.	UNIT
I_{CBO}	collector cut-off current	I_E = 0; V_{CB} = 20 V	–	–	100	nA
		I_E = 0; V_{CB} = 20 V; T_j = 150 °C	–	–	5	µA
I_{EBO}	emitter cut-off current	I_C = 0; V_{EB} = 5 V	–	–	100	nA
h_{FE}	DC current gain	I_C = 100 mA; V_{CE} = 1 V; see Figs 2, 3 and 4				
	BC337		100	–	600	
	BC337-16		100	–	250	
	BC337-25		160	–	400	
	BC337-40		250	–	600	
	DC current gain	I_C = 500 mA; V_{CE} = 1 V; see Figs 2, 3 and 4	40	–	–	
V_{CEsat}	collector-emitter saturation voltage	I_C = 500 mA; I_B = 50 mA	–	–	700	mV
V_{BE}	base-emitter voltage	I_C = 500 mA; V_{CE} = 1 V; note 1	–	–	1.2	V
C_c	collector capacitance	I_E = i_e = 0; V_{CB} = 10 V; f = 1 MHz	–	5	–	pF
f_T	transition frequency	I_C = 10 mA; V_{CE} = 5 V; f = 100 MHz	100	–	–	MHz

FIGURE II.2.4. – Fiche technique du BC337

Plus : $\mathbf{IC_{MAX}}$: 500mA ; $\mathbf{IB_{MAX}}$:200mA ; $\mathbf{VCE_{MAX}}$:50v ;$\mathbf{VBE_{MAX}}$:5v ; $\mathbf{P_{TOT}}$:625mW

?

Woaaaah ! T'as vu ça ! Où a-t-il trouvé ce truc ? C'est super compliqué tout ça !

Mais non, mais non. OK j'admets - c'est en anglais ce qui (vous) complique pas mal la chose ! C'est la première difficulté en électronique : trouver les informations concernant les composants que l'on souhaite utiliser ! Beaucoup de fiches techniques existent en anglais, il faut pas mal fouiner sur internet pour en trouver en français...
Bon, c'est quoi tout ce charabia ?
Quelques paramètres sont assez simples à comprendre :

- $\mathbf{P_{TOT}}$: Puissance totale : 625mW (dépasser ce seuil ça sent le cramé !)
- $\mathbf{IC_{MAX}}$: Intensité qui traverse la borne C (idem) ;
- $\mathbf{IB_{MAX}}$: Intensité qui traverse la borne B (idem) ;
- $\mathbf{VCE_{MAX}}$: Tension maximale entre les bornes C-E ;
- $\mathbf{VBE_{MAX}}$: ... je pense que vous avez compris le système, non ?

Autrement dit : vous êtes prévenus ! Pour que le transistor reste de ce monde, il ne faut pas trop jouer au malin !
Dans le tableau nous ne nous soucions que de quelques valeurs :

- $\mathbf{h_{FE}}$ (parfois appelé ***Beta*** ou) : le gain ou amplification d'intensité ;
- $\mathbf{V_{BE}}$: la chute de tension entre les bornes B-E

Laissons les autres valeurs aux puristes...
C'est le gain qui nous intéresse le plus. La valeur minimale de 100 veut dire qu'une petite intensité traversant B-E provoque une intensité 100 fois plus grande entre C et E. C'est ***ça*** l'intérêt des transistors !
Je remets le même schéma, simplifié :

FIGURE II.2.5. – Branchement simplifié d'un transistor

La 'charge' peut être le moteur, un relais, une ampoule 6v, un haut-parleur…
J'ai 'oublié' la diode, mais c'est ***obligatoire*** pour les charges dites 'inductives' : bobine, relais, moteur.
Examinons une valeur de la fiche technique **à ne pas dépasser** :
IC_{MAX}: (Intensité qui traverse la borne C) : 500mA.

?

Quelle est la valeur de résistance MINIMUM pour notre 'charge' ?

Contenu masqué n°4

?

Si =100 ; quel intensité I_{BE} 'génère' l'intensité maximale IC_{MAX} de 500mA ?

Contenu masqué n°5

Ça va jusque là ? Ce n'est pas si dur, hein ?
Last question (oups – je me suis laisser emporter) :

?

Quelle valeur de R2 pour limiter l'intensité I_{BE} à 5mA maximum ?

C'est un poil plus compliqué, mais pas beaucoup. Allons voir…
Le courant qui traverse R2 traverse aussi la jonction B-E du transistor. Ce n'est pas dit dans le texte, mais cette jonction est exactement comme une *DIODE*. Et nous avons vu qu'une diode provoque *TOUJOURS* une chute de tension d'environ 0,7v. Dans le tableur il est indiqué une valeur de 1,2v maximale, mais cette valeur varie avec la température ambiante (la température à l'intérieur du transistor, plutôt).
Restons donc avec nos 0,7v fétiches. Pour que I_{BE} ne dépasse pas 5mA avec une tension de 'pilotage' de 5v (la sortie de l'Arduino), nous devons 'chuter' $5–0,7 = 4.3V$ quand I_{BE} = 0,005A.
La valeur minimum de R2 est donc $R = \frac{U}{I} = \frac{4.3}{0.005} = 860\Omega$.
Par sécurité (et facilité) je mets toujours une résistance de **1K** pour R2.
Voila ! On est content ?
Notez que j'ai choisi la valeur de la plus faible : vous avez vu dans le tableur que la valeur maximum peut être jusqu'à six fois plus. C'est encore mieux pour nous : cela veut dire que le transistor est *vraiment* en conduction maximum avec les 5mA de I_{BE}.
Voilà, pour cette semaine, les transistors.

2. Il ne faut pas dépasser 500mA ou 0,5A. $R = \frac{U}{I} = \frac{5}{0.5} = 10\Omega$
3. $\frac{500mA}{100}$

Merci à Glenn Smith pour ce cours!

II.2.4. Arduinomètre, deuxième partie

II.2.4.0.1. Arduino thermomètre avec diode pour sonde

II.2.4.0.1.1. Deuxième partie La semaine dernière nous avons vu qu'une simple diode pouvait fonctionner comme sonde de température avec notre Arduino. Cette semaine nous allons continuer avec ce montage afin de le faire fonctionner sur une plage de températures de 0°C à 100°C. Souvenez-vous que nous avions (OK, moi j'ai) choisi un point arbitrairement sur la courbe comme point de repère. J'ai pris un point qui correspondait à une chute de tension de 600mV à 20°C.
Le montage et le "Arduino Sketch" de la semaine dernière nous ont permis de régler le potentiomètre (résistance variable) afin de calibrer ce point précis. La grande difficulté est de savoir si la diode est bien à 20°C au moment de ce réglage ! À la fin de ce module je vous donnerai une autre façon de calibrer le montage. Mais, finissons avec la version "Approximation et triche"!

II.2.4.0.1.2. Trichons... La triche ici est que nous allons supposer une correspondance linéaire entre la température et la chute de tension de -2mV / °C. De plus, nous avons décidé que, à 20°C, la chute de tension était de 600mV. À partir de cette valeur on peut déduire que, à **0°C** la chute de tension devrait être de $600mV + (20 \times 2mV) = 640mV$. Dans l'autre sens, la chute de tension à **100°C** devrait être de $600mV - (80 \times 2mV) = 440mV$.
Les valeurs retournées par la fonction `analogRead()` ne sont pas directement en mV, il faut les convertir. Souvenez-vous que nous avons demandé à l'Arduino de nous donner des valeurs avec une référence à 1,1V. Un résultat de 1023 équivaut 1,1V donc chaque intervalle de 1 est l'équivalent de 1,1 / 1023 = 1.075mV. Nous pouvions trouver la valeur qui correspond aux 640mV (point 0°C): $\frac{640}{1.075} = 595$. Et, idem, pour le point 100°C: $\frac{440}{1.075} = 409$.
Reste à faire en sorte de traduire une gamme de valeurs allant de 595 à 409 en température de 0°C à 100°C.
Comment faire?
Nous avons déjà vu une fonction Arduino très intéressante: `map()`
La *syntaxe* (c'est à dire la bonne façon de l'écrire) de cette fonction est:

```
map(valeur_en_entrée, gamme_départ_bas, gamme_départ_haut,
    gamme_résultat_bas, gamme_résultat_haut)
```

Nous pouvons donc écrire:

```
// Lire la valeur
DiodeValue = analogRead(Diode);
// Traduire valeur en degrés C
Température = map(DiodeValue, 595, 409, 0, 100);
```

Voici un exemple de Sketch complet:

```
/*
  Montage pour faire un thermomètre avec une diode pour sonde
  Deuxième partie : thermomètre fonctionelle (approximative)
  La diode est montée en sens 'conduction' (cathode branche sur GND)
  avec une résistance fixe de 4.7k et une résistance variable de
  préférence de 10k en serie.

  Il faut d'abord régler le montage pour que le résultat de analogRead();
  donne 558 (environ 600mV) a 20°C (Sketch "Diode_Thermo_1")

  Ne plus toucher le potentiomètre après !

  Avril 2014 MOOC Fabrication Numérique

*/
// Définition des broches
#define Diode A0 // Broche pour la diode

// variables:
int DiodeValue = 0;  // valeur lue sur la diode
int Temperature = 0; // Brrrrr
float Result = 0.0;  // pour les calculs

void setup() {
  pinMode(LED, OUTPUT);
  Serial.begin(9600);
  // Réference maximum pour le convertisseur ADC = 1,1v
  // au lieu des 5v par défaut.
  analogReference(INTERNAL);
}

void loop() {
  // Lire la valeur
  DiodeValue = analogRead(Diode);
  Serial.print("Valeur : ");
  Serial.print(DiodeValue);
  // Traduire valeur en degrés C
  Temperature = map(DiodeValue, 595, 409, 0, 100);
  // Puis afficher
  Serial.print("\t environ ");
  Serial.print(Temperature);
  Serial.println(" °C");
  delay(10000);
}
```

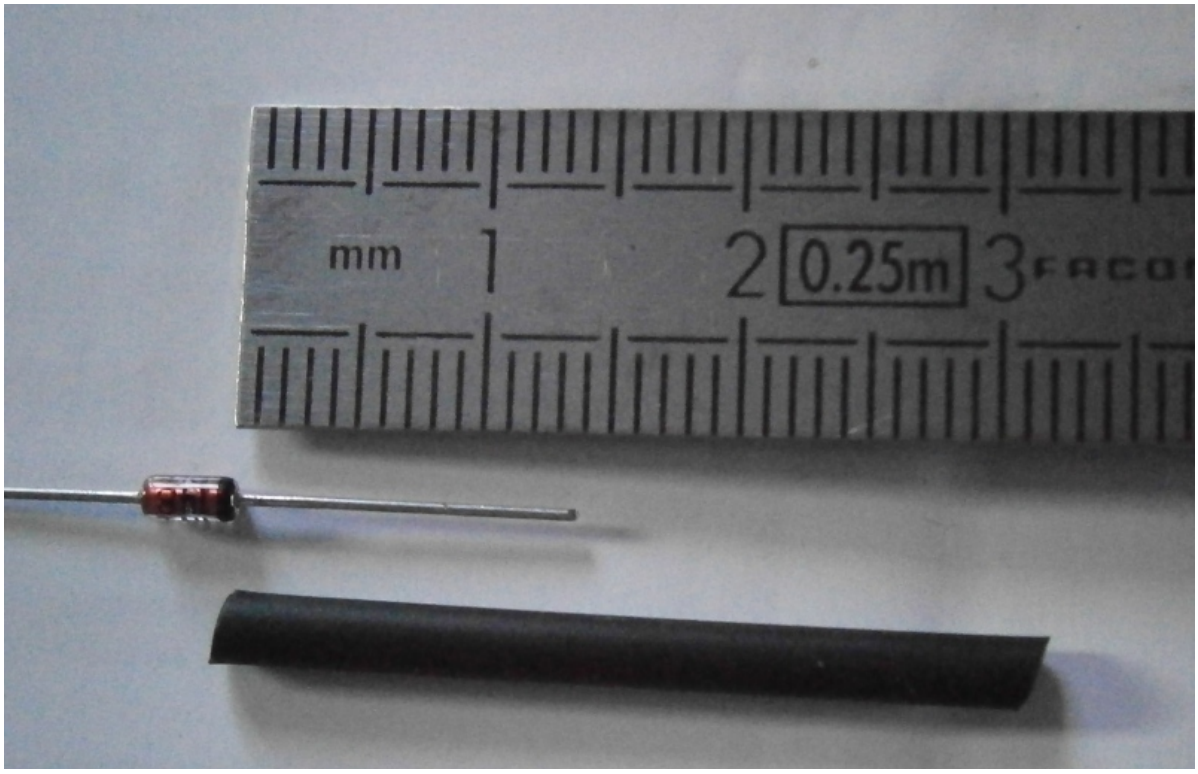
Listing 12 – Lecture de la température avec la diode

II.2.4.0.1.3. Sonde flexible Ce montage est utile pour nous donner une indication de la température ambiante autour de l'Arduino. Si on veut mesurer la température à un endroit précis, ou si on veut calibrer le système avec des températures connues, il faut fabriquer une sonde "déportée" et flexible. Voici comment le faire, étape par étape. Vous aurez besoin de:

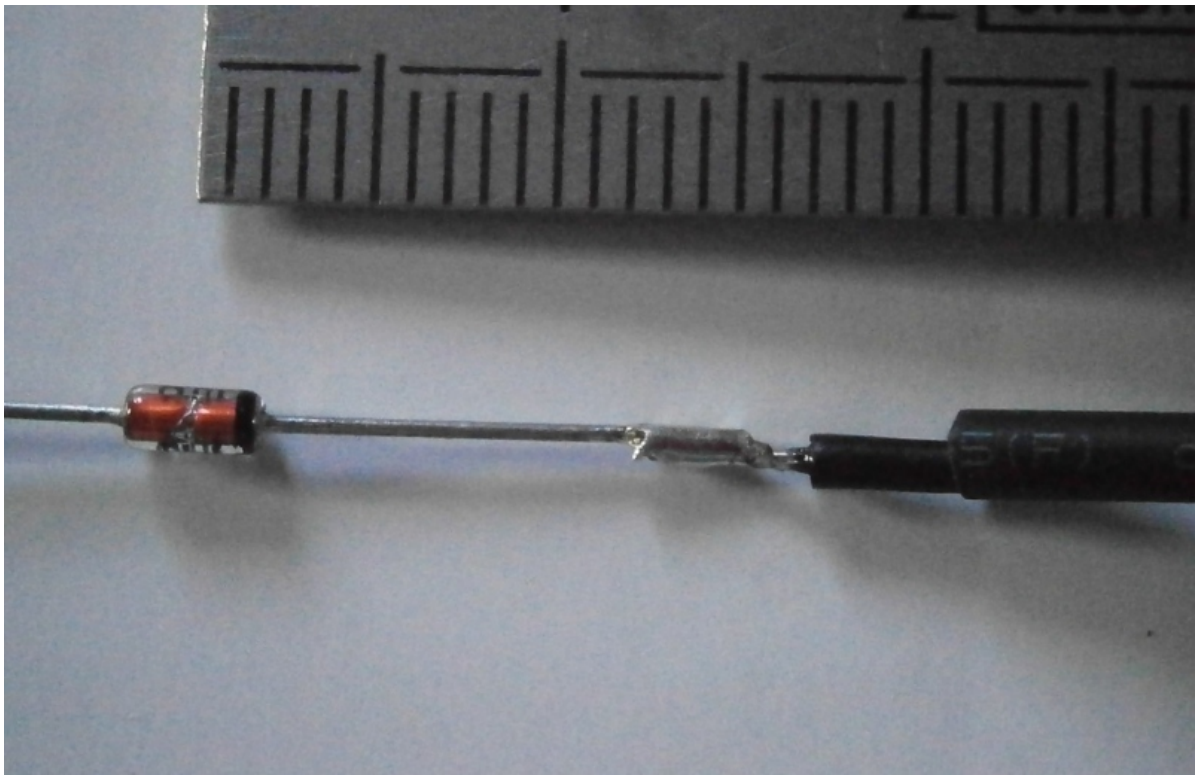
- Une diode 1N4148 ou équivalent ;
- Deux fils flexibles de même longueur mais de couleurs différentes ;
- Quelques petites longueurs de gaine thermorétractable (ou - si vous êtes patient - un peu de mastic silicone) ;
- Savoir souder avec un fer à souder…

Allons-y!

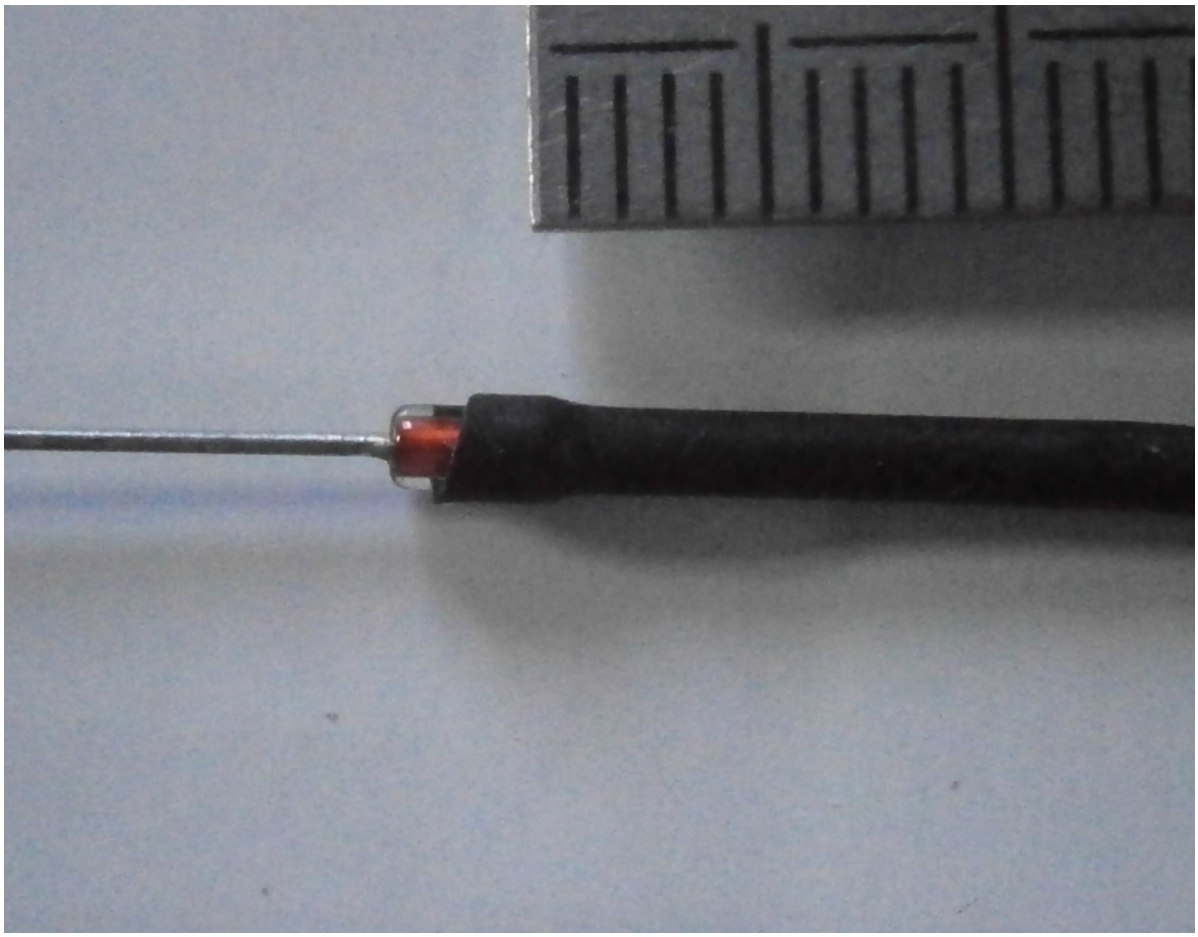
1. Couper le fil Cathode (bande colorée) à environ 15mm de long, et couper un morceau de gaine d'environ 3cm :



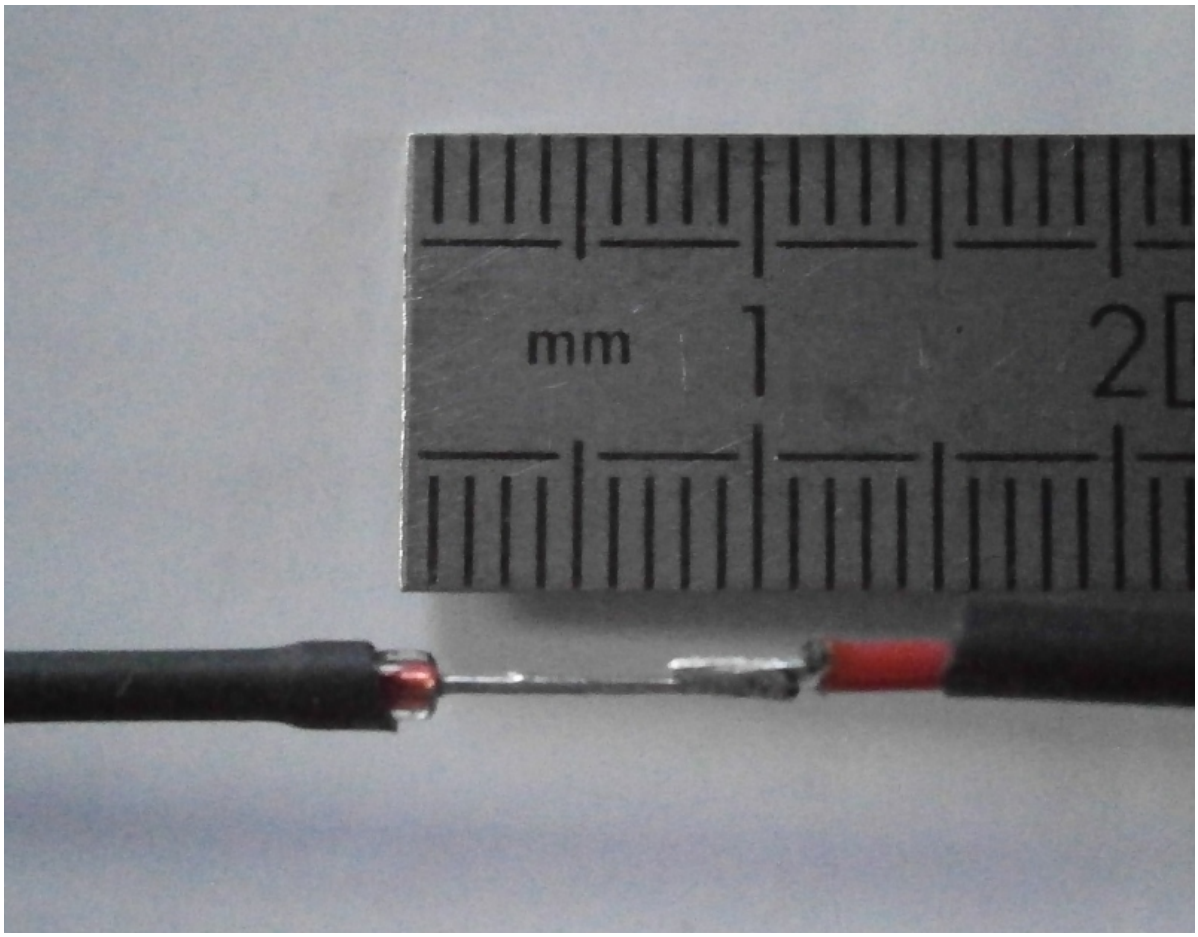
2. Souder un des fils sur la Cathode (et notez la couleur choisie !). Astuce: si vous n'êtes pas très expérimenté encore avec le fer à souder, tenir la diode par le fil Cathode avec une pince à bec fin (tenir entre le corps de la diode et la soudure) - ceci évite les doigts brûlés mais, surtout, aide à éviter la surchauffe de la diode.



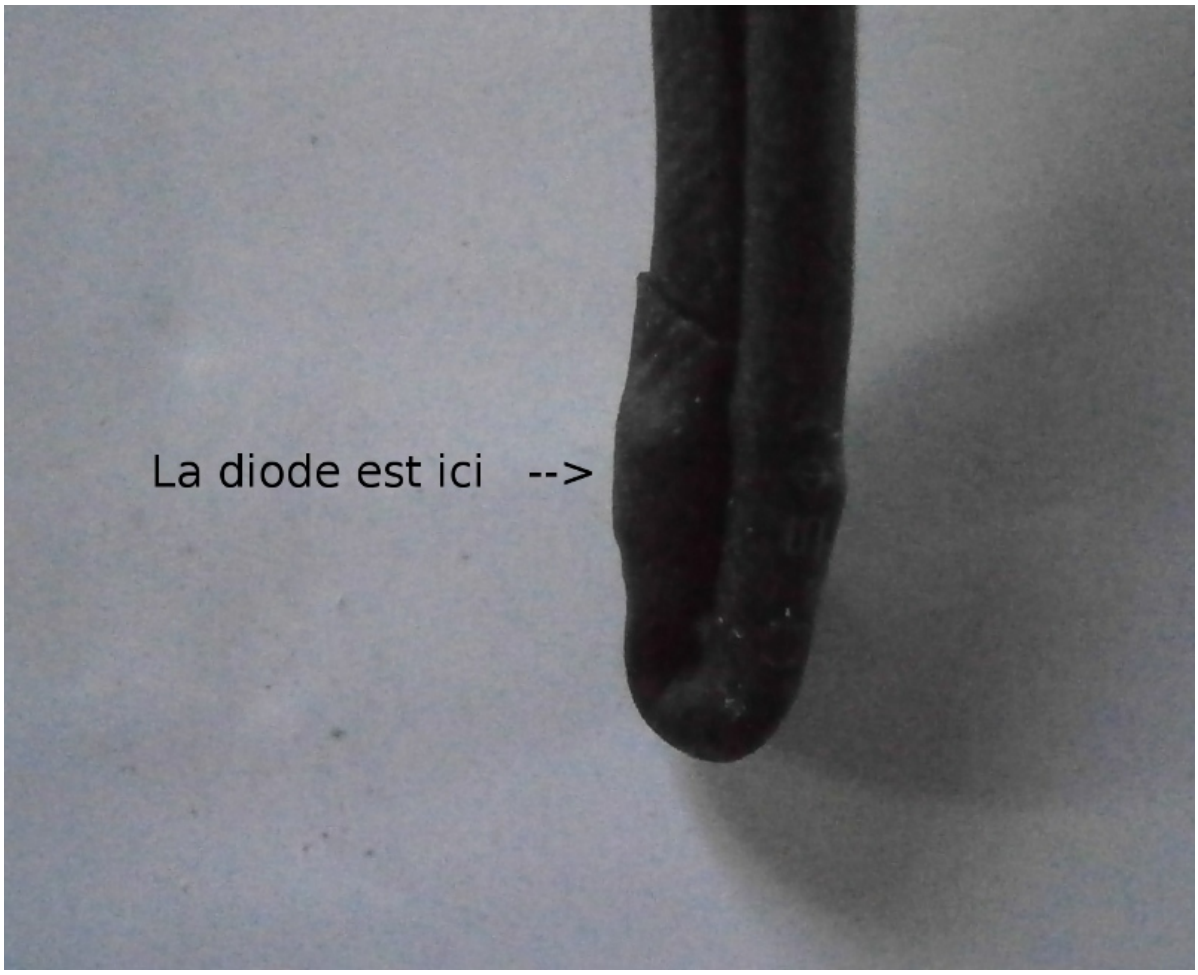
3. Laisser l'ensemble se refroidir puis faire glisser le morceau de "gaine thermo" sur votre soudure en couvrant bien au moins la moitié du corps de la diode. Faire chauffer la gaine ***rapidement*** avec un briquet jusqu'à un rétrécissement total.



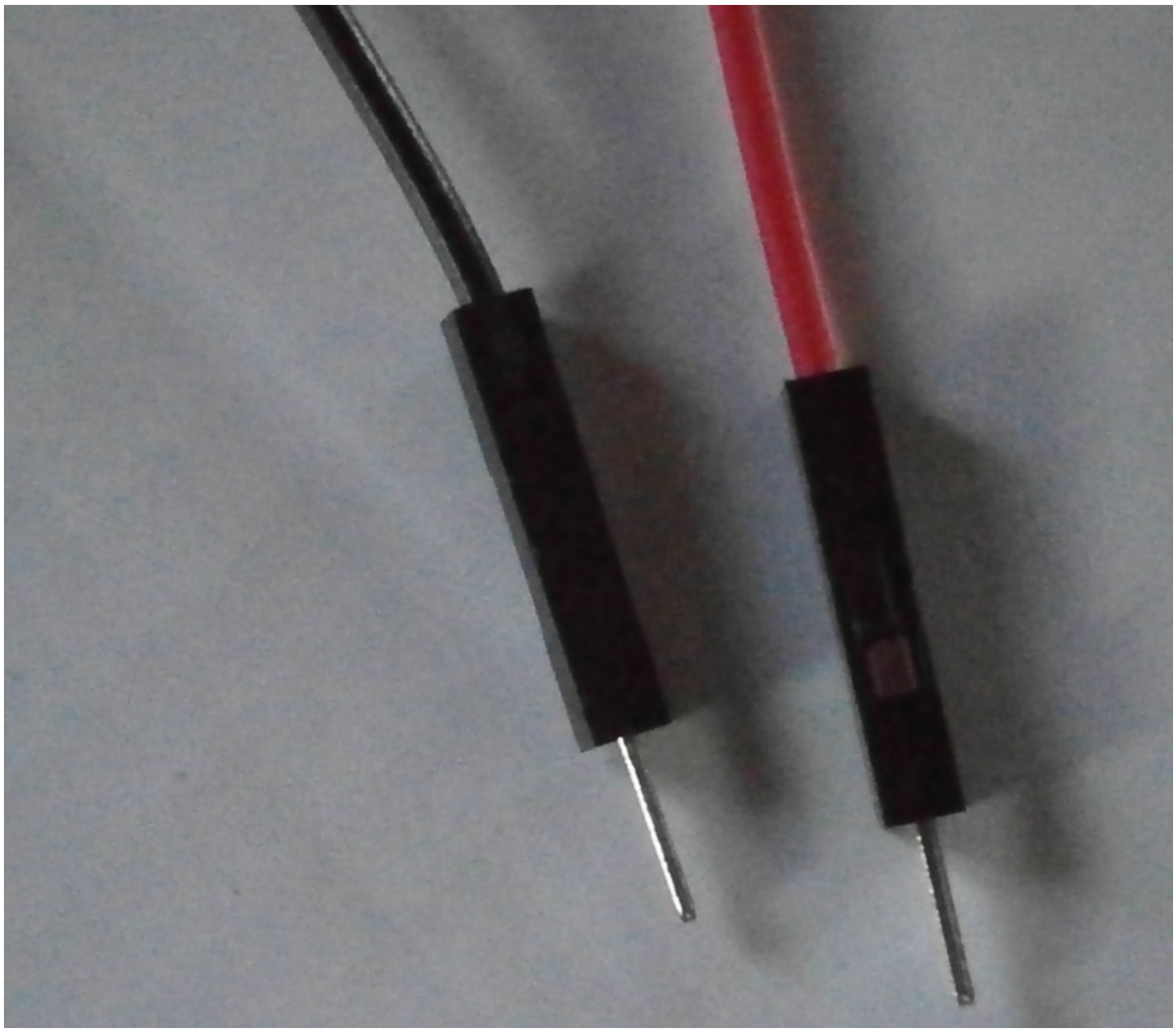
4. Côté Anode, couper le fil à environ 1cm et faire la deuxième soudure. Laisser refroidir.



5. Couper un deuxième morceau de gaine, glissez-le sur l'ensemble en faisant bien chevaucher la première gaine: faire chauffer (vite !) Si vous n'avez pas de la gaine thermorétractable vous pouvez enduire l'ensemble soudures / diode avec un mastic silicone APRÈS avoir tordu les fils dans leur forme définitive.
6. Plier un des deux fils contre la diode afin de donner une forme de sonde. Si vous avez de la gaine avec un diamètre un peu plus grand vous pouvez recouvrir l'ensemble.



7. Si vous avez la possibilité de mettre des connecteurs à l'autre extrémité des deux fils flexibles votre sonde sera plus facile à brancher sur la platine d'expérimentation. Sinon dénuder environ 15mm et faire étamer (couvrir d'une fine couche de soudure) les deux fils.



Ça y est!

II.2.4.0.1.4. Calibration avec deux points connus Pour améliorer ce montage il faut noter la valeur retournée par la fonction `analogRead()` avec une température connue basse et une température connue haute. Notre sonde flexible sera utile car, si vous avez bien protégé les soudures, vous pouvez l'immerger dans un liquide… Un verre d'eau plein de glaçons nous donnera une référence pour 0°C, de l'eau bouillante pour 100°C (ne faites pas chauffer la diode trop longtemps, par contre - juste le temps pour que le résultat se stabilise).
Ensuite vous n'avez qu'à mettre vos nouvelles valeurs dans la commande `map()`

II.2.4.0.1.5. Suite Maintenant que la fonctionnalité de base est maîtrisée, à vous de modifier et d'améliorer le Sketch pour le rendre encore plus utile. La semaine prochaine je vous donnerai quelques pistes et un montage plus complet.
Merci à Glenn Smith pour ce cours!

Contenu masqué

Contenu masqué n°3

1. J'espère bien que vous avez tous chanté ensemble "Pour filtrer !" Sinon je rends mon tablier… En effet les moteurs génèrent beaucoup de parasites et il vaut mieux protéger votre cher microcontrôleur.
2. C'est une protection supplémentaire, mais encore plus importante que les condensateurs (car l'Arduino a quand-même quelques condensateurs de filtration sur la platine) : nous allons en parler à nouveau au sujet des relais, mais n'importe quelles bobines électriques envoient des décharges négatives quand on essaie de les débrancher. Cette 'réaction', une tension négative, peut détruire le transistor (ou d'autres composants) car l'amplitude de cette réaction est souvent de centaines de volts… La diode est montée en sens inverse : en temps normal la cathode est au positif par rapport à l'anode et donc la diode ne fait rien. En cas de pic de tension négatif au-delà de la valeur de chute de tension de la diode (combien de volts ?[1]) la diode entre en conduction et l'énergie est court-circuitée vers 0v. Montage ***obligatoire*** pour les moteurs, bobines, relais etc.

Retourner au texte.

Contenu masqué n°4

Réponse: $R = \frac{U}{I} = 10\Omega$[2]

Retourner au texte.

Contenu masqué n°5

Réponse: $\mathbf{R} = 5\text{mA}$[3]

Retourner au texte.

II.3. Semaine 8 : Modélisation 3D

Introduction

Avant d'imprimer des objets, il faut les modéliser Bienvenue dans cette 8ème semaine du MOOC "La Fabrication Numérique". Cette semaine, nous continuons notre découverte des outils de modélisation avec une série de vidéos d'initiation à Blender. Nous verrons quel vocabulaire, quels outils et quelles techniques sont nécessaires pour produire des modèles 3D exploitables par les imprimantes 3D qui seront abordées la semaine prochaine.
En parallèle de ces vidéos d'introduction, nous continuons à vous proposer des cours sur les concepts de base du **prototypage électronique** et du développement **Arduino**.
Bonne semaine!

II.3.1. Introduction à la modélisation 3D

II.3.1.0.1. La modélisation 3D

Élément externe (Video) —
Consultez cet élément à l'adresse https://geo.dailymotion.com/player.html?video=x2kgztv&.

Merci d'avoir regardé cette vidéo. Celle-ci constitue une brève introduction à la modélisation 3D. La suite passe par l'installation de Blender , un incroyable logiciel de modélisation, d'animation et de rendu en 3D. Ce logiciel nous permettra de créer nos premiers modèles 3D, exportables pour les imprimantes 3D que nous verrons la semaine prochaine.
Pour utiliser Blender, nous vous invitons à regarder la vidéo ci-dessous. Aussi, il est fortement recommandé de lire les références proposées ici:

II.3.1.0.1.1. Références

— Manuel complet consacré à Blender par l'équipe de FlossManuals
— Manuel officiel par l'équipe Blender

Merci à Laurent Mattlé pour ce cours!

II.3.2. Initiation à Blender

Voici dès maintenant trois vidéos sur Blender, le logiciel de modélisation 3D que nous allons utiliser pour modéliser nos objets.

II.3.2.0.0.1. Vidéo 1/3

Élément externe (Video) —
Consultez cet élément à l'adresse https://geo.dailymotion.com/player.html?video=x2kgzr9&.

II.3.2.0.0.2. Vidéo 2/3

Élément externe (Video) —
Consultez cet élément à l'adresse https://geo.dailymotion.com/player.html?video=x2kgzrw&.

II.3.2.0.0.3. Vidéo 3/3

Élément externe (Video) —
Consultez cet élément à l'adresse https://geo.dailymotion.com/player.html?video=x2kgzse&.

II.3.3. Introduction à la modélisation 3D - Annexe

II.3.3.0.1. Annexe

Voici quelques compléments dont il faudra se souvenir lorsque nos modèles seront exportés vers des imprimantes 3D.

II.3.3.0.1.1. Types d'impression Il existe de nombreux procédés d'impression 3D mais les plus courants sont:

- **le FDM (Fused Deposition Modeling)**: où l'on vient faire fondre un filament de thermoplastique et le déposer couche par couche.
- **le SLS (Selective Laser Sintering)**: où l'on vient solidifier de la poudre avec une impulsion laser. Cette technique plus coûteuse donne de meilleurs résultats (elle est souvent la technique utilisée par les boutiques en ligne).

Si on modèlise pour le SLS il faut penser à faire un trou d'au moins 3mm pour que la poudre à l'intérieur puisse s'échapper de notre modèle.

II.3.3.0.1.2. Parois Une face seule ne sera pas considérée comme un objet 3D et ne sera pas imprimée. Pour qu'une face soit comprise comme telle, il faut que cette face ait de l'épaisseur. L'épaisseur dépend de votre outil et matériau cible. Pour le SLS (si vous passez par un imprimeur) il faut simplement suivre les instructions disponibles sur son site internet et qui conviennent à sa machine.
Pour le FDM, l'épaisseur du fil a son importance car chaque paroi est faite du dépôt d'une épaisseur de fil pour un côté et d'une autre pour l'autre côté. Il faudra donc que l'épaisseur de la paroi soit au moins de 2 épaisseurs de fils. Enfin, si on veut qu'il y ait un remplissage, il faut de la place pour celui ci sinon la solidité pourrait lourdement en pâtir.
Je resterai très superficiel sur ce qu'il faut faire tout simplement parce que chaque couple *"imprimante/fil"* réagit différemment selon le modèle et il est préférable d'aller discuter avec celui qui maintient l'imprimante ou de faire quelques tests avant, pour savoir à quoi vous attendre (pour avoir un ordre de grandeur, 0,25mm entre les parois est un bon minimum).

II.3.3.0.1.3. Manifold Lors de la création du modèle il faut avoir en tête que toutes les faces soient bien reliées. Chaque espace doit être fermé et c'est ce qu'on peut découvrir avec la fonction non manifold dans Blender.
Ensuite, ce n'est pas parce que deux vertices ou *edges* sont au même endroit qu'ils sont reliés d'où la nécessité de supprimer les doublons (*remove doubles*) et de vérifier s'il n'y a pas de parois qui se chevauchent ou bien des faces internes qui ne servent à rien et qui ne pourront être interprétées par le logiciel de l'imprimante.

II.3.3.0.1.4. Normales Vérifiez aussi les normales de chaque face pour qu'elles soient toutes tournées vers l'extérieur.

II.3.3.0.1.5. Tailles/poids En fait nous sommes vite tentés d'avoir beaucoup de subdivisions pour avoir le plus de "douceur" possible mais le nombre de polygons augmentet de façon dramatique.
Il faut savoir utiliser les polygons à bon escient: ne pas dépasser les 500 000 vertices tout en optimisant la taille des modèles.

II.3.3.0.1.6. Angle des parois (Overhang) Un concept que l'on oublie souvent est la gravité. En effet, si on dépose du plastique fondu dans le vide, il tombe! Mais si on le pose à moitié dans le vide ça peut peut-être tenir. Cela signifie que l'on peut se permettre de monter nos couches de plastique avec un angle de 45°.
Sinon il faut faire des échafaudages, soit automatiquement soit en les construisant soi-même dans le modèle, ce qui permet d'optimiser la consommation de plastique.

II.3.3.0.1.7. Vérification automatique Il y a dans Blender un outil 3D appelé "print toolbox" qui, une fois paramétré avec les limites de votre machine cible, met en lumière les points problématiques du modèle.
Dans le même esprit, voici deux logiciels très utiles:

- Net fabb studio
- Meshmixer

Ces logiciels prennent le modèle et le répare quasi automatiquement s'il n'est pas conforme aux exigences de l'impression. Il sont tous deux gratuits et finalement apportent sécurité et sérénité au flux de production.
Merci à Laurent Mattlé pour ce cours!

II.3.4. Transistors (suite)

II.3.4.0.1. Arduino chauffage

L'Arduino peut-être programmé pour calculer la température ambiante, en exploitant une tension variable fournie par un capteur. En précisant un seuil, l'Arduino peut allumer ou arrêter la chauffage. Mais comment brancher un chauffage de 2Kw à 240v sur l'Arduinomètre?
Une possibilité comme ceci:

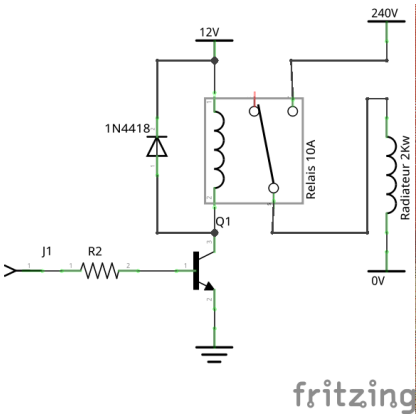


FIGURE II.3.1. – Un transistor et un relais

Ici la charge c'est un relais de puissance. La bobine du relais est prévue pour 12V, et sa résistance est d'environ 250Ω[1]. Notez bien que il n'y a aucune connexion entre le circuit 230V et le circuit Arduino / transistor. C'est un des grands intérêts des relais: une séparation complète de deux environnements électriques incompatibles entre-eux. Néanmoins, un tel montage devra se trouver bien enfermé dans une boite en plastique, à cause des branchements secteur!
Q: Quel est l'intensité maximum quand le transistor est en conduction?
R: La résistance du relais est de 250Ω[1]. Donc $I_C = \frac{12V}{250=0.048A}$ (48mA)
Mais il y a une dernière valeur à vérifier, surtout car nous avons commencé à augmenter la tension commandée par le transistor: la dissipation (de chaleur) totale, P_{tot}.
Si le transistor chauffe trop, il se détruit! Les données constructeur donnent une valeur de P_{tot} de 625mW. Vérifions que nous sommes dans les clous:
$P_{tot} = P_{CE} + P_{BE}$: C'est-à-dire qu'il ne faut pas oublier la (faible) intensité qui traverse les bornes $B - E$ dans notre calcul!
Pour calculer la puissance dissipée dans le transistor par les 48mA qui passent à travers les bornes $C - E$, il faut connaître la chute de tension. C'est aussi dans les données fabricant:
$VCE_{SAT} = 700mV$ (0,7v)
On peut donc calculer:
$P_{CE} = 0.7V \times 0,048A$ (l'intensité à travers le relais) $= 33,6mW$
$P_{BE} = 5V \times 0,005A$ (l'intensité de 'pilotage') $= 25mW$
Et donc $P_{tot} = 33.6 + 25 = 58.6mW$. Pas de soucis avec P_{tot} pour cette application.

II.3.4.0.2. Récapitulons : Transistor 'interrupteur'

— Chercher les données techniques;

1. Non, ce n'est pas un chiffre sorti par magie: soit on lit les données fabricant; soit on la mesure avec un ohm-mètre!

— Calculer IC_{MAX} avec la vraie valeur R de votre charge (relais, bobine, etc.) et, bien sûr, la vraie valeur de la tension d'alimentation;
— Calculer la résistance du Base $R2$ pour limiter l'intensité IB a $\frac{IC_{MAX}}{}$: n'oubliez pas la chute de tension VBE;
— Ne dépassez $JAMAIS$ les limites maximales du transistor (IC, P_{tot} …)

II.3.4.0.3. Seuil de tension V_{BE}

Vous vous souvenez de notre petit montage R-C avec l'Arduino d'il y à quelques semaines? Celui où la LED clignotait 'en retard' par rapport à la sortie de l'Arduino. Je vous ai dit que la formule pour calculer le temps pour un circuit RC était, tout simplement, $R \times C$.
Mais ce calcul est basé sur le principe que le temps T est le temps pour charger complètement le condensateur (en fait la définition c'est le temps pour arriver à 63% de la tension d'alimentation, mais passons). Notre montage, souvenons-nous:

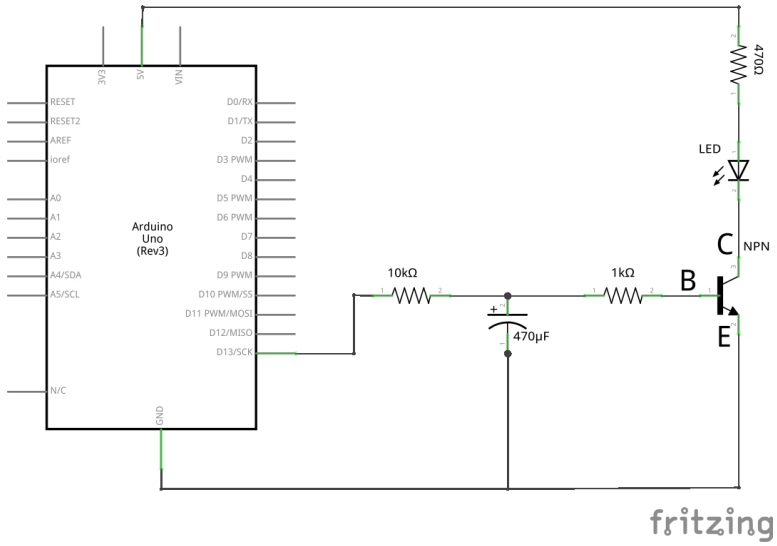


FIGURE II.3.2. – Le montage RC

La sortie de l'Arduino va charger le condensateur à travers la résistance de 10 kΩ. La tension d'alimentation va être 5V (le niveau 'high' de l'Arduino). Mais notre transistor commence à laisser passer du courant à partir d'environ 0.7 V de tension V_{BE} (l'équivalent diode). Le transistor va se mettre en conduction bien avant que le condensateur se trouve complètement chargé!
Voila un mystère résolu….

II.3.4.0.4. Transistors PNP

Les montages 'transistor' vus jusqu'à présent utilisent un transistor NPN. L'émetteur se trouvait toujours à une tension négative par rapport au collecteur. Il existe des transistors avec exactement les mêmes données techniques SAUF que leur polarité est inversée. Ce sont, donc, les transistors PNP. On les trouve beaucoup dans les circuits analogiques, surtout dans les amplificateurs audio. Mais parfois ils peuvent nous filer un coup de main pour nos montages numériques.
Regardons ce schéma ensemble:

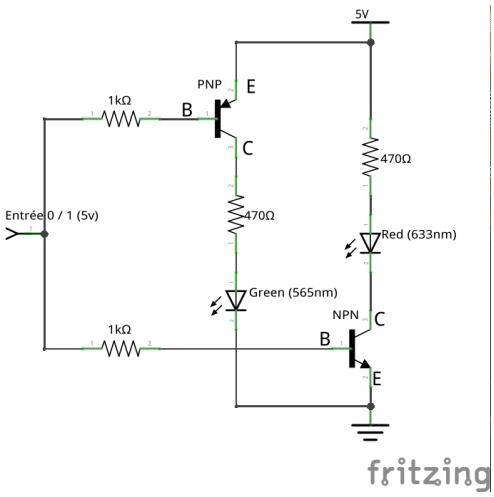


FIGURE II.3.3. – Un transistor PNP et un transistor NPN

La partie inférieure/droite du schéma nous est familière: un transistor NPN qui pilote une DEL rouge. Et voilà en haut à gauche notre transistor PNP. Ici on voit bien le sens des flèches sur les symboles pour les deux transistors: ils pointent dans la même direction : de +ve au -ve. L'émetteur PNP est +ve par rapport à son collecteur.
Alors, quelle est l'utilité de ce montage? Essayons de comprendre un peu…
Supposant que le niveau à l'entrée est à '1' (5v). Nous avons déjà vu que cela fait passer un courant à la base du transistor NPN, et la DEL s'allume. Mais la base du transistor PNP se trouve au même potentiel que son émetteur – et donc le transistor ne "s'allume" pas. Si l'entrée est tirée au niveau 0v, le NPN va s'arrêter de laisser passer le courant de la DEL rouge, mais maintenant, car V_{BE} devient -5 V (réfléchir), le PNP va se mettre en conduction – et la DEL verte s'allume.
Avec une seule sortie de l'Arduino, nous pilotons deux sorties exclusives. Une idée pour les feux de signalisation sur une machine ou pour sécuriser nos feux tricolores / piétons, par exemple…
Les règles d'utilisation et les calculs pour les transistors PNP sont les mêmes que pour les NPN: il suffit (je sais, c'est pas facile – c'est toujours écrit en Shakespeare) de chercher la fiche technique. Parfois les fabricants sont sympas et ils mettent sur la fiche l'équivalent ***complémentaire*** du transistor: cela aide à la recherche.

II.3.4.1. Transistors à Effet de Champ (FET)

À nouveau j'essaie de rester simple (mais il y aura un peu de maths simples, comme pour les transistors bipolaires).
Je vous ai déjà dit que la différence importante entre les FETs et les transistors bipolaires, c'est la présence d'une ***tension*** entre les bornes G et S qui contrôlent l'intensité qui traverse les FETs, là où nous avons vu que c'est ***l'intensité*** (entre B et E) pour les transistors bipolaires.
Les FETs ont aussi un autre différence: une très faible résistance interne en mode conduction (souvent moins d'un ohm); mais une résistance très élevée en mode bloquant. Deux paramètres faisant des FETs des candidats idéaux pour faire des interrupteurs pilotés. Quelques valeurs pour une FET de puissance, le IRFP260:

Standard Power MOSFET **IRFP 260**

V_{DSS} = 200 V
$I_{D(cont)}$ = 46 A
$R_{DS(on)}$ = 55 mΩ

N-Channel Enhancement Mode

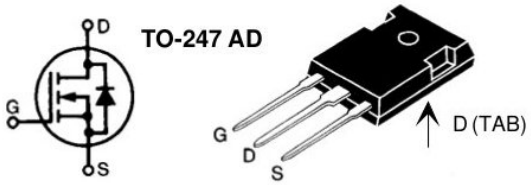


FIGURE II.3.4. – Le FET IRFP260

Des valeurs à faire rêver:
- $V_{DSMAX} = 200V$ (la tension maximale entre D et S) ;
- $I_{DMAX} = 46A$ (intensité maximale à travers D) ;
- $R_{DS(on)} = 55m$ (0,055Ω)! (la résistance en mode conduction)

Une autre valeur, tirée de la même fiche technique:
$V_{GS(thres)} = 2,0V_{min}, 4,0V_{max}$
Alors, c'est quoi? C'est tout simplement le seuil de tension à partir duquel le FET commence à entrer en conduction.
Alors, ça se branche comment? Ben, presque comme un transistor 'normal' :

FIGURE II.3.5. – Branchement d'un FET

Notre 'charge' (bobine[1], relais, moteur…) est montée entre la source d'alimentation et le FET.
Une résistance en entrée plutôt pour protéger la source (Arduino) en cas de court-circuit (car c'est une ***tension*** sur le gate qui fait fonctionner le FET - plus besoin de limiter le courant).
La résistance de 10KΩ pour les cas ou l'entrée reste 'flottante': c'est pour s'assurer que le FET s'arrête bien: le Gate est tiré vers 0v.
La source de tension dans ce schéma est de 5v, mais les données techniques nous disent qu'on peut aller jusqu'à 200v!
Notons que c'est la ***tension*** qui contrôle le passage de courant de G à S, et qu'il faut plus que 5v pour 'pousser' ce particular FET[2] jusqu'à passer les 46A – plutôt 10V. Les sorties Arduino nous 'fournissent' seulement 5V au niveau 'high'… Comment faire si on a vraiment besoin de faire un interrupteur de puissance $200V$ / $46A$ (cela fait un peu plus que 9kW quand même)?
Facile – vous ne voyez pas? Allez, réfléchissez un peu…
Et si on faisait 'piloter' le FET avec un transistor en mode interrupteur?:

FIGURE II.3.6. – Pilotage d'un FET par un bipolaire

Alors, à vous de travailler un peu:
Q: Quel est le type du transistor Q1?[3]
Q: Comment calculer la valeur pour R2?[4]
Q: Comment calculer la valeur pour R1?[5]
Q: Quel est l'inconvénient de ce montage (car il en a un)?[6]

II.3.4.1.0.1. Fonctionnement Quand la sortie de l'Arduino est à 5V, un courant traverse R1, le transistor Q1 est en conduction et le Gate du FET est tiré à 0V. Pas de conduction (courant S-D à travers le FET): la charge est donc éteinte.
Quand le niveau à l'entrée se trouve à 0V, Q1 n'est plus en conduction et le Gate est tiré vers 12V à travers R2: le FET est on conduction 'plein gaz'.
Réponses:
[1] Ex. une des deux / quatre bobines d'un moteur pas-a-pas [2] cad l'IRFP260. Il existent les FETs qui rentre en conduction totale avec VGS à 5V [3] Regarder le sens de la flèche: NPN [4] Il faut 'laisser passer' assez de courant pour bien ramener le gate du FET à 0V: disons 100mA [5] Exactement comme avant: chute de tension pour limiter le courant IBE [6] L'inversion des commandes: le FET est 'allumé' quand la sortie Arduino est à 0V
Il est, bien sûr, possible d'annuler cette 'inversion' en ajoutant un deuxième transistor devant Q1, mais cela coûte plus cher que d'inverser la logique de sortie de l'Arduino…
Merci à Glenn Smith pour ce cours!

II.3.5. Arduinomètre, troisième partie

II.3.5.1. Arduino thermomètre avec diode pour sonde

II.3.5.1.1. Troisième partie

Nous avons maintenant la base fonctionnelle pour créer plusieurs types d'appareils. Je vais vous donner quelques pistes, quelques idées de réalisations complètes, mais c'est l'opportunité pour vous de laisser votre imagination et votre créativité vous guider.
Coté Arduino on commence à être équipé pour attaquer la programmation. Avec l'appel et l'écriture des fonctions, et les *libraries* (bibliothèques) nous allons avancer à grands pas.

II.3.5.1.2. Couper le cordon

Jusqu'à maintenant les dialogues avec l'Arduino se faisaient par la liaison série (le cordon USB). Il est temps de s'affranchir de cette liaison, de couper le cordon et de rendre notre montage autonome. Deux soucis se présentent à nous: a) comment alimenter l'Arduinomètre? b) comment afficher la température?

II.3.5.1.2.1. Alimentation Si on coupe le cordon, l'Arduino n'est plus alimenté - car s'est notre ordinateur qui l'alimentait à travers la prise USB. Plusieurs solutions sont possibles.
Une simple c'est une petite alimentation secteur 5v équipée avec une prise USB (type chargeur de téléphone): brancher le câble USB bleu et hop! L'Arduino s'allume.
D'autres solutions:

— une alimentation universelle, à brancher sur la prise d'alimentation de l'Arduino. Dans ce cas veillez bien à ce que la tension en sortie de l'alimentation ne dépasse pas 12V;

— pile ou accumulateur: à partir d'une tension de 7V, jusqu'à 12V. À brancher aussi sur la prise d'alimentation de l'Arduino.

En tout cas l'Arduinomètre dans l'état actuel n'est pas du tout gourmand en électricité et la solution pile est très pratique. Une pile 9V fera l'affaire.

II.3.5.1.2.2. Je ne vois rien... Et oui, comment lire la température maintenant qu'il n'y a plus la liaison série? C'est maintenant que les choses deviennent intéressantes, car vous pouvez laisser libre cours à votre imagination pour un système d'affichage vraiment personnalisé. Regardons quelques solutions ensemble.

II.3.5.1.3. Affichage LCD

Les afficheurs LCD deviennent assez abordables, et ont l'intérêt d'être relativement normalisés. Le logiciel de développement de l'Arduino est fourni avec une bibliothèque standard, `Liquid Crystal`, qui prend en charge ces afficheurs en nous épargnant la complexité de leur protocole de communication. Avec un tel afficheur nous pouvions afficher au moins 2 lignes de 16 symboles: largement assez pour afficher la température! Voici un montage exemple sur notre platine d'expérimentation Fritzing :

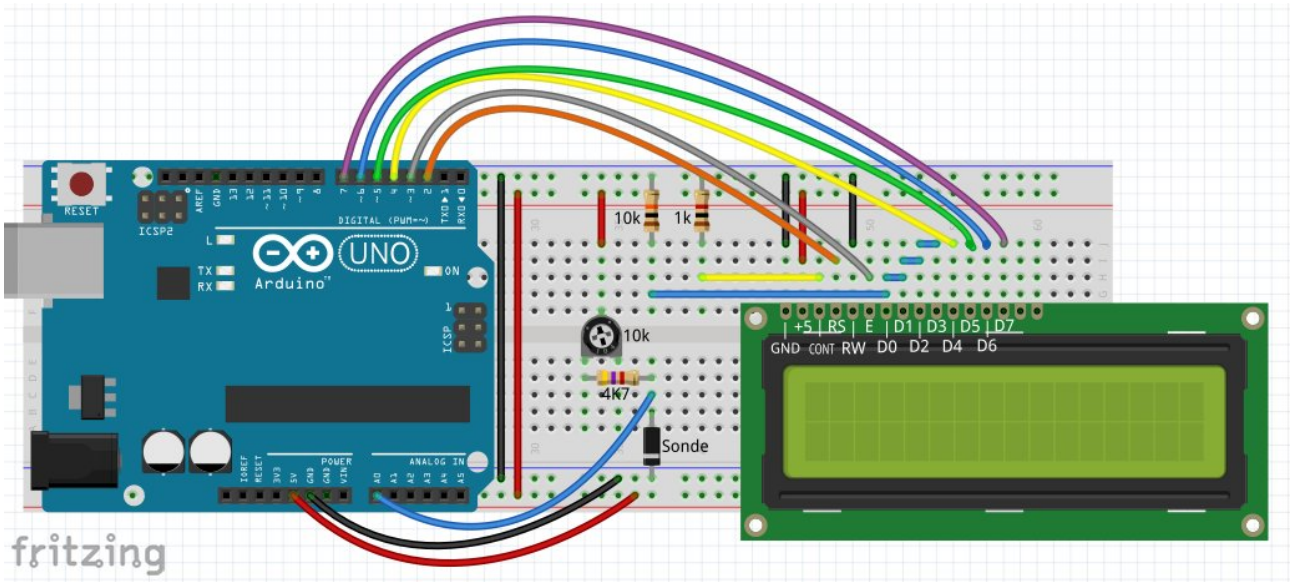


FIGURE II.3.7. – Montage de l'écran LCD

Schéma électronique de l'écran LCD

Si votre afficheur LCD n'est pas déjà équipé, il faut ajouter un jeu de broches pour faciliter le branchement sur la platine:

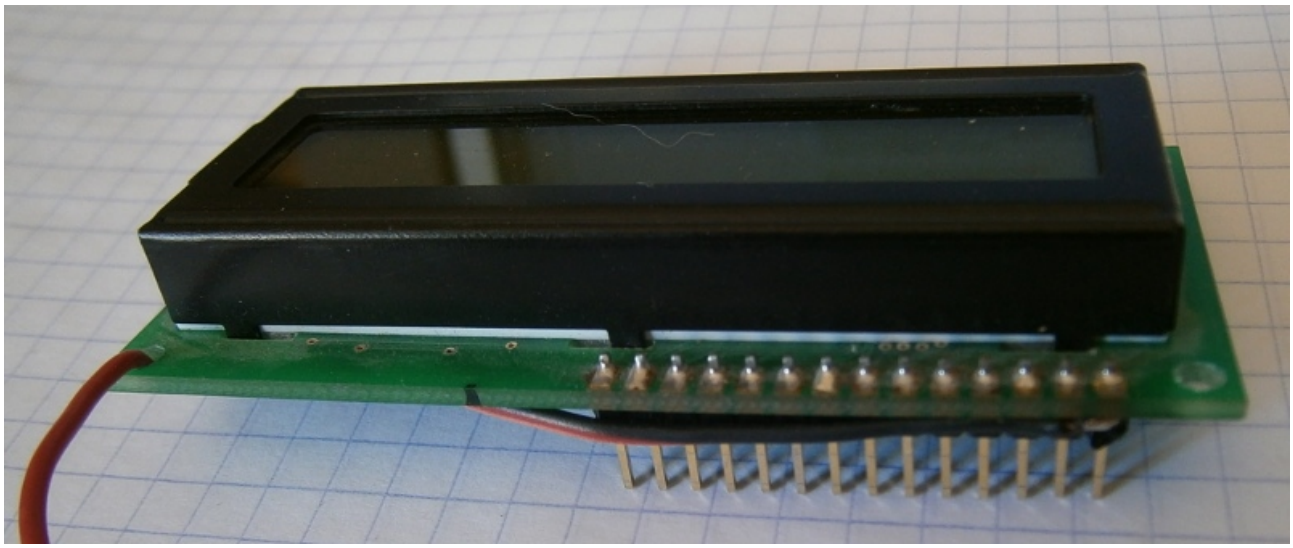


FIGURE II.3.8. – Un écran et des broches

Ça commence à faire beaucoup de fils à brancher, mais avec un peu *d'ordre et de méthode* (comme disait Hercule Poirot) ce n'est pas trop compliqué. Ce montage exemple est simplifié au maximum. Je n'ai pas branché les deux broches du rétro-éclairage à LEDs (broches 15 et 16). Pour la grande majorité des afficheurs vendus actuellement les résistances pour ces LEDs sont déjà montées dans l'afficheur - vous n'avez qu'à brancher le 0V et +5V. Si vous avez récupéré l'afficheur il faut trouver la documentation ou regarder de près afin de voir si les résistances sont présentes ou non. Notez que l'utilisation du rétro-éclairage n'est pas trop compatible avec une alimentation par petite pile 9v (consommation accrue).
Pour tester ce montage il va nous falloir un Sketch (programme). Voici un exemple:

```
/*
  Montage pour faire un thermomètre avec une diode pour sonde
  Troisième partie : thermomètre fonctionelle.
  La diode est montée en sens 'conduction' (cathode branche sur
    GND)
  avec une résistance fixe de 4.7k et une résistance variable de
  préférence de 10k en serie.

  Il faut d'abord régler le montage pour que le résultat de
    analogRead();
  donne 558 (environ 600mV) a 20°C (Sketch "Diode_Thermo_1") -OU-
  calibrer le montage avec deux températures connus.

  Ne plus toucher le potentiomètre après !

  Cette version avec afficheur LCD branché en mode 4-bits comme
    suite :

  LCD    Arduino    Libellé
  ================================================================
    1      -        0v / GND
    2      -        +5v
```

```
20      3       -       Contraste
21      4       2       RS (register select)
22      5       -       RW (read / write) Branché sur 0v
23      6       3       E  (enable)
24     11       4       D4 (données bit 4)
25     12       5       D5 (données bit 5)
26     13       6       D6 (données bit 6)
27     14       7       D7 (données bit 7)
28
29   Avril 2014 MOOC Fabrication Numérique
30
31 */
32 // Bibliothèques
33 #include <LiquidCrystal.h>
34
35 // Définition des broches
36 #define Diode A0  // forward biased diode w/10K R to Vcc
37 // Brochage LCD
38 #define LCD_rs 2
39 #define LCD_enable 3
40 #define LCD_d4 4
41 #define LCD_d5 5
42 #define LCD_d6 6
43 #define LCD_d7 7
44
45 // variables:
46 int DiodeValue = 0;    // for the diode
47 int Temperature = 0; // Brrrrr
48 float Result = 0.0;  // for calculations
49
50 // Déclarer notre afficheur LCD
51   LiquidCrystal lcd(LCD_rs, LCD_enable, LCD_d4, LCD_d5, LCD_d6,
       LCD_d7);
52
53 void setup() {
54   // Réference max. pour l'ADC = 1,1v
55   analogReference(INTERNAL);
56   // Démarrer l'affichage
57   lcd.begin(16, 2);
58   lcd.print("Arduinometre");
59 }
60
61 void loop() {
62   DiodeValue = analogRead(Diode); // Lire la valeur
63   Result = DiodeValue * 1.075; // Convertir en mV
64   // effacer la deuxième ligne
65   lcd.setCursor(0, 1);
66   lcd.print("                ");
67   lcd.setCursor(0, 1);
68   // afficher les valeurs
```

```
69    lcd.print(Result);
70    lcd.print("mV ");
71    Temperature = map(DiodeValue, 595, 409, 0, 100);  // Traduire
          valeur en degrés C
72    lcd.print("(");
73    lcd.print(Temperature);
74    lcd.write(0xdf);  // Le symbole "°"
75    lcd.print("C)");
76    delay(10000);  // Attente 10s
77  }
```

Listing 13 – Arduinomètre avec écran LCD

Vous avez peut-être remarqué que l'afficheur par défaut n'affiche que les caractères sans accents. Si vous voulez ajouter des symboles supplémentaires ("é" et "è", par exemple) il faut ajouter des appels à `createChar();` pour définir vos symboles. Pour plus d'information sur l'utilisation des fonctions fournies par la bibliothèque `LiquidCrystal`, et l'utilisation de `createChar();` se référer à la page chez "mon-club-elec" ***ici*** . J'ajouterai un article Wiki dédié à ce sujet ultérieurement.
Notre montage ressemble à un vrai thermomètre maintenant!

II.3.5.1.4. Avertisseur

Notre Arduinomètre est désormais autonome et nous affiche la température. Mais (surtout sans rétro-éclairage) il faut se pencher dessus pour voir la T°. Et si on ajoutait un avertissement par LED? LED verte = température dans la plage souhaitée; LED rouge = trop basse ou trop élevée. J'espère que je n'ai plus besoin de dessiner comment brancher les LEDs (avec les résistances en série, bien sûr). On peut utiliser les broches 8 et 9 de l'Arduino comme sorties pour les LEDs. Il va falloir ajouter quelques bouts de code supplémentaires dans le Sketch:

II.3.5.1.4.1. Déclarations :

```
1  #define LED_Rouge 8#define LED_Verte 9
2  // constantes:
3  const int seuilBas = 15;   // Seuil d'erreur bas
4  const int seuilHaute = 25; // Seuil d'erreur haut
```

II.3.5.1.4.2. Déclaration d'une procédure à nous (à ajouter ***après*** la } à la fin du `loop()` ou ***avant*** `setup()`)

```
1  void avertissement() {
2   if ((Temperature > seuilHaut) || (Temperature < seuilBas)) {
3      digitalWrite(LED_Rouge, HIGH);
4      digitalWrite(LED_Verte, LOW);
5    }
6    else {
7     digitalWrite(LED_Rouge, LOW);
```

```
   digitalWrite(LED_Verte, HIGH);
  }
}
```

II.3.5.1.4.3. Corps du code : (quelque part dans le `loop()`)

```
// T° bien ?
avertissement();
```

II.3.5.1.4.4. Exemple complet :

```
/*
  Montage pour faire un thermomètre avec une diode pour sonde
  Troisième partie : thermomètre fonctionelle.
  La diode est montée en sens 'conduction' (cathode branche sur
    GND)
  avec une résistance fixe de 4.7k et une résistance variable de
  préférence de 10k en serie.

  Il faut d'abord régler le montage pour que le résultat de
    analogRead();
  donne 558 (environ 600mV) a 20°C (Sketch "Diode_Thermo_1") -OU-
  calibrer le montage avec deux températures connus.

  Ne plus toucher le potentiomètre après !

  Cette version avec afficheur LCD branché en mode 4-bits comme
    suite :

  LCD    Arduino    Libellé
  ========================================================
    1      -        Ov / GND
    2      -        +5v
    3      -        Contraste
    4      2        RS (register select)
    5      -        RW (read / write) Branché sur 0v
    6      3        E  (enable)
   11      4        D4 (données bit 4)
   12      5        D5 (données bit 5)
   13      6        D6 (données bit 7)
   14      7        D7 (données bit 7)

  ... et deux LEDs pour avertissement.

  Avril 2014 MOOC Fabrication Numérique

```

```
*/
// Bibliothèques
#include <LiquidCrystal.h>

// Définition des broches
#define Diode A0  // forward biased diode w/10K R to Vcc
#define LED_Rouge 8
#define LED_Verte 9

// Brochage LCD
#define LCD_rs 2
#define LCD_enable 3
#define LCD_d4 4
#define LCD_d5 5
#define LCD_d6 6
#define LCD_d7 7

// constantes:
const int seuilBas = 15;  // Seuil d'erreur bas
const int seuilHaute = 25;  // Seuil d'erreur haute

// variables:
int DiodeValue = 0;    // for the diode
int Temperature = 0; // Brrrrr
float Result = 0.0;  // for calculations

// Déclarer notre afficheur LCD (et création de l'objet "lcd" de type LiquidCrystal)
  LiquidCrystal lcd(LCD_rs, LCD_enable, LCD_d4, LCD_d5, LCD_d6, LCD_d7);

void setup() {
  pinMode(LED_Rouge, OUTPUT);
  pinMode(LED_Verte, OUTPUT);
  digitalWrite(LED_Rouge, LOW);
  digitalWrite(LED_Verte, LOW);

  // Réference max. pour l'ADC = 1,1v
  analogReference(INTERNAL);  // Arduino UNO et autres

  // Démarrer l'affichage
  lcd.begin(16, 2);  // Obligatoire pour démarrer
  lcd.print("Arduinometre");
}

void loop() {
  DiodeValue = analogRead(Diode); // Lire la valeur
  Result = DiodeValue * 1.075; // Convertir en mV
  Temperature = map(DiodeValue, 595, 409, 0, 100);  // Convertir en degrés C
```

```
80    // T° bien ?
81    avertissement();
82
83    // effacer la deuxième ligne
84    lcd.setCursor(0, 1);
85    lcd.print("                ");
86    lcd.setCursor(0, 1);
87
88    // afficher les valeurs
89    lcd.print(Result);
90    lcd.print("mV ");
91    lcd.print("(");
92    lcd.print(Temperature);
93    lcd.write(0xdf);  // Le symbole "°"
94    lcd.print("C)");
95
96    delay(10000);  // Attente 10s
97  }
98
99  void avertissement() {
100   if ((Temperature > seuilHaute) || (Temperature < seuilBas)) {
101     digitalWrite(LED_Rouge, HIGH);
102     digitalWrite(LED_Verte, LOW);
103   }
104   else {
105   digitalWrite(LED_Rouge, LOW);
106   digitalWrite(LED_Verte, HIGH);
107   }
108 }
```

Listing 14 – Exemple compte de l'Arduinomètre

II.3.5.1.5. Idées

Encore quelques idées à essayer

- Utiliser une ligne de LEDs pour afficher la température (comme l'affichage du niveau du son sur certains magnétophones…(vous souvenez-vous des magnétophones ?);
- Ajout d'une deuxième diode sonde pour faire thermomètre intérieur / extérieur (mais lire l'article Wiki sur la sonde LM35: quelques informations sur la lecture de plusieurs sources analogiques simultanées) ;
- Au lieu d'un avertissement (ou en supplément), faire actionner une sortie numérique pour piloter un relais: l'Arduinomètre devient Arduinostat.
- faire un thermomètre à cadran: prendre un servomoteur, y fixer une aiguille et faire tourner l'aiguille sur une échelle graduée. Si votre servomoteur a une plage de fonctionnement limitée, utiliser la technique des seuils: réserver un espace 'rouge' 5° en bas et 5° en haut du cadran et utiliser le reste de la plage pour afficher la température 'normale';
- Vos idées ici…

Merci à Glenn Smith pour ce cours!

II.3.6. Travaux pratiques

II.3.6.0.1. TP à faire pour la semaine 9

À partir de la vidéo 3 de cette semaine, du cours et des ressources disponibles sur Internet, l'objectif de cette semaine est de modéliser avec Blender une variante de ce que Laurent vous décrit.
Nous vous invitons ensuite à montrer vos créations sur le forum, sur Twitter avec le hashtag #MoocFab , sur Facebook ou sur la communauté Google+ .
L'équipe du MOOC

II.4. Semaine 9 : Les imprimantes 3D

Introduction

Si les machines qui impriment des objets en trois dimensions ne sont pas récentes, l'attention portée à la version grand public de ces machines grandit depuis quelques années. Cette semaine, nous nous lançons dans la découverte des machines CNC avec un focus sur les imprimantes 3D. Nous verrons brièvement comment fonctionnent ces machines et quels outils existent pour passer d'un fichier informatique à un objet 3D.
Pour cela, nous sommes partis à Brest pour rencontrer Arthur et Stéphane qui commercialisent des imprimantes 3D et qui nous expliquent comment fonctionne une imprimante 3D un peu particulière: la RepRap.

II.4.1. L'impression 3D

II.4.1.1. PRINCIPES DE L'IMPRIMANTE 3D

Élément externe (Video) —
Consultez cet élément à l'adresse `https://geo.dailymotion.com/player.html?video=x2kh00w&`.

Merci d'avoir regardé cette première vidéo sur les principes de base de l'impression 3D.
Cette vidéo fait suite au cours de la semaine dernière où nous avons survolé les bases de la modélisation 3D avec Blender. Une fois votre modèle créé sous Blender, il ne reste plus qu'à l'exporter en `.stl`. Ce fichier peut ainsi être transformé en langage machine ou `.gcode` pour l'imprimer avec votre imprimante 3D ou celle de votre FabLab. C'est justement ce qu'on vous montre dans la vidéo ci-dessous. Aussi, il est fortement recommandé de lire les références proposées ici:

II.4.1.1.0.1. Références

- Une excellente vidéo sur l'impression 3D produite par Dimitri du FabLab de Lyon (également en dessous dans les annexes)
- Une autre vidéo très bien faite par FutureMag sur Arte (également en dessous dans les annexes)
- Guide d'utilisation de Slic3r par Gary Hodgson (traduit par Laurent Le Goff du TyFab)
- Manuel complet consacré à RepRap par l'équipe de FlossManuals

Merci à Arthur Wolf et Stéphane Philippe, adhérents du TyFab et fondateurs de Ipso Factio . Merci également aux équipes des Petits Débrouillards Bretagne pour nous avoir permis de tourner au TéléFab à Brest.

II.4.1.2. PASSER D'UN MODÈLE 3D À DU LANGAGE MACHINE

Élément externe (Video) —
Consultez cet élément à l'adresse https://geo.dailymotion.com/player.html?video=x2kgzzn&.

II.4.1.3. ANNEXE : L'IMPRESSION 3D

Élément externe (Video) —
Consultez cet élément à l'adresse https://www.youtube.com/embed/bH3nhwNF2CM?feature=oembed.

II.4.1.4. L'IMPRIMANTE 3D POUR TOUS - FUTUREMAG - ARTE

https://www.youtube.com/watch ?v=AWzjTxuhImQ

II.4.2. Les interruptions sur Arduino

II.4.2.1. Les interruptions

Vous avez remarqué qu'il n'est pas si évident que ça de détecter l'appui sur un bouton si au même moment on est en train de gérer la circulation des voitures et des trains! Si l'on pouvait simplement interrompre momentanément notre programme quand le bouton est appuyé...
Vous avez peut-être remarqué aussi que, dans les discussions autour des boutons poussoirs, parfois les mots "interruption" ou "interrupt" (anglais) ou "asynchrone" sont dits (ou parfois à peine chuchoté)!
Alors, on va regarder pourquoi certains experts répondent tout de suite "Hola, petit, va grandir un peu avant d'essayer ça", et que d'autres sourient sous cape.

II.4.2.1.1. C'est quoi ?

D'abord, il faut faire quelques petits rappels :

1. Le microcontrôleur, le coeur de l'Arduino, "tourne" à une vitesse phénoménale: souvent 4,000,000 instructions par seconde, et ne **s'arrête JAMAIS**;
2. Cela dit, il ne fait **qu'une chose à la fois**: à l'instant où il est en train d'éclairer une LED il ne peut pas lire en même temps l'état d'une entrée;
3. Dans ce monde de *vitesse* nous, pauvres humains, nous ressemblons à des escargots avec nos réactions de l'ordre de dizaines de millisecondes!

Alors, comment faire en sorte que, si quelque chose d'important se produit, on puisse arrêter la tâche en cours et intervenir avant qu'il soit trop tard?
Dans le monde numérique, on fait appel à des **interruptions**: un état signalé (provoqué par un événement à l'intérieur, ou par un signal extérieur) qui provoque une altération des séquences de traitements programmés.
Chaque type de processeur à ses propres façons de faire mais, en gros, voici ce qui se passe lors d'une interruption:

- Le programme en cours est suspendu ;
- Le processeur mémorise l'endroit où il se trouvait (l'instruction qu'il était en train de préparer);
- Il sauvegarde la valeur de certaines variables internes (sauvegarde d'état);
- Il commence à exécuter un programme spécifique au type d'interruption…
  - (Pendant ce temps-là, notre programme à nous n'avance pas.)
  - Quand le traitement de l'interruption est terminé:
- Le processeur récupère les variables sauvegardées;
- Il 'saute' à nouveau dans le programme d'origine, là où il en était.

Cela semble laborieux et lent mais à 4,000,000 instructions par seconde, cela ne prend même pas une milliseconde!
Une partie de cette séquence dépend du type de processeur mais **l'essentiel**, les instructions de traitement de l'événement, est sous notre contrôle: on peut lui dire quoi faire dans tel ou tel cas. Mais attention: si nous nous trompons dans notre code de traitement d'interruption, nos erreurs peuvent provoquer des symptômes assez difficiles à diagnostiquer. Vous voila prévenus…

II.4.2.1.2. Événement extérieur

Nous allons nous focaliser sur les événements extérieurs, que nous pouvons déclencher à notre guise. Sur l'Arduino UNO, deux broches sont équipées de détection d'interruption: les broches 2 (*INT0*) et 3 (*INT1*). Regardons ensemble comment on déclare que nous voulons traiter des interruptions sur l'une de ces deux broches:

```
1 attachInterrupt(interruption, fonction, mode);
```

- **interruption**: le numéro d'interruption à traiter (sur UNO: "0" = broche 2; ou "1" = broche 3)
- **fonction**: le nom de la fonction à appeler (pas de passage de paramètres)
- **mode**: spécifier dans quelles conditions l'interruption est à signaler: `LOW`, `CHANGE`, `RISING`, `FALLING`

II.4.2.1.2.1. Mode Je ne l'ai pas dit, mais on est en train de parler **d'événement externe**, donc les broches 2 et/ou 3 devaient êtres déclarées en `INPUT` ou `"INPUT-PULLUP"`, sinon on

va provoquer des interruptions nous même si une broche affectée au traitement d'interruption change d'état (une source de soucis, déjà). Alors, ces "modes", c'est quoi?

— `LOW`: provoquer une interruption quand la broche en question se trouve à l'état 0v;
— `CHANGE`: provoquer une interruption quand l'état de la broche en question change;
— `RISING`: provoquer une interruption quand la tension sur la broche en question monte (0v vers 5v);
— `FALLING`: provoquer une interruption quand la tension sur la broche en question diminue (5v vers 0v).

Les plus vigilants d'entre vous sont en train de se demander "Et *HIGH* alors ?". Alors, ce n'est disponible QUE sur l'Arduino Due - mais le plus souvent on utilise des résistances (internes ou externes) de PULLUP, et donc l'état HIGH est, généralement, l'état de repos…

II.4.2.1.2.2. Fonction Parfois la fonction est appelée **ISR** pour "Interrupt Service Routine" en anglais. Nous voilà au coeur du problème: quoi faire avec cette interruption? Évidemment tout dépend de la raison de l'interruption: la baignoire qui déborde, le téléphone qui sonne, une trame TCP/IP qui arrive. Je peux vous donner quelques conseils, cependant.

1. N'oubliez pas que tout le monde nous attend: dans cette fonction il faut faire le strict minimum, et le plus vite que possible;
2. Nous sommes pas chez nous! Gare à nos gros sabots qui écrasent des variables ou modifient l'état du système - utiliser uniquement des variables déclarées localement, ou déclarées en `volatile` - et ne touchez à rien d'autre sauf si vous savez vraiment ce que vous faites !
3. N'utilisez pas les instructions comme `delay()` ou les communications sur liaisons-séries (qui utilisent fréquemment des interruptions elles aussi). Si on veut voir l'état d'une variable à l'intérieur de notre fonction, sauvegardez sa valeur dans une autre variable (déclarée en globale et `volatile`) et l'afficher avec `lcd.print()` ou `serial.print()` depuis le programme principal.

II.4.2.1.2.3. Faire le strict minimum, et le plus vite que possible Par exemple, si notre interruption porte sur la détection d'événement physique, comme l'appui sur un bouton, n'essayez pas de traiter les cas provoqués par cet appui mais simplement rendre cette information disponible pour notre programme. Allez, je pense qu'un montage avec l'Arduino sera plus efficace que tout ce blabla.

II.4.2.1.3. Détection de bouton

Nous allons faire un montage très simple: un bouton (condensateur conseillé mais pas obligatoire):

Figure II.4.1. – Détection de bouton

Et un sketch aussi assez simple:

II.4.2.1.3.1. Déclarations :

```
#define boutonPin  2  // C'est l'entrée INT0
//  Pour qu'on puisse changer la valeur d'une variable depuis une fonction de
// gestion d'interruption, il faut que la variable soit déclarée "volatile"
volatile boolean changed = false;
```

II.4.2.1.3.2. Notre fonction ISR à nous : (à ajouter *après* la `}` à la fin du `loop()` ou *avant* `setup()`)

```
void doContact() {
   changed = true;
   // Difficile de faire + court !
}
```

II.4.2.1.3.3. Activer l'interruption INT0, attacher notre ISR : (dans `setup()`)

```
attachInterrupt(0, doContact, FALLING);
```

II.4.2.1.3.4. Et la boucle `loop()`

```
void loop(){
    if ( changed ) {
        changed = false;
        Serial.println("Ha ! Vous avez touché le bouton");
    }
    delay(1000);
}
```

Notez bien que nous ne nous soucions plus du bouton dans notre code principal. Si la variable `changed` reste à l'état "faux", la boucle ne fait qu'attendre! Ni vu, ni connu, notre fonction `doContact` a modifié la valeur de `changed` à l'instant où nous avons appuyé sur le bouton. Le délai avant que le message s'affiche est tout simplement le `delay(1000)` qui continue à décompter.

II.4.2.1.3.5. Exemple complet :

```
/*
  Exemple pour démontrer l'utilisation des interruptions.

```

```
4    Bouton poussoir, connecté entre GND et la broche 2 (INT0).
5
6    Résistance PULLUP activé pour cette broche.  Condensateur de
       100nF
7    en parallèle avec le bouton recommandé.
8
9    Glenn Smith.   MOOC Fabrication Numérique, mai 2014
10
11 */
12
13 #define boutonPin  2  // C'est l'entrée INT0
14
15 //  Pour qu'on puisse changer la valeur d'une variable depuis une
       fonction de
16 // gestion d'interruption, il faut que la variable soit déclarée
       "volatile"
17 volatile boolean changed = false;
18
19 // Notre fonction de traitement d'interruption.  Le strict
       minimum, souvenez-vous.
20 void doContact() {
21   changed = true;
22   // Difficile de faire + court
23 }
24
25 void setup() {
26   // Broche en entrée avec résistance de pull-up
27   pinMode(boutonPin, INPUT_PULLUP);
28
29   // Attacher notre fonction ISR, détection de flanc descendant
         (5v vers 0v)
30   attachInterrupt(0, doContact, FALLING);
31
32   // Montrer que nous sommes la
33   Serial.begin (115200);
34   Serial.println("Hello");
35 }
36
37 void loop(){
38   if ( changed ) {
39     changed = false;
40     Serial.println("Ha ! Vous avez touché le bouton");
41   }
42   delay(1000);
43   // Vous pouvez ajouter du code ici pour faire clignoter un led,
         etc.
44 }
```

II.4.2.1.4. Suite

Dans le Wiki je vais publier un article utilisant cette technique pour 'dompter' un bouton rotatif - un must pour augmenter les possibilités d'interface humain / Arduino. Pour ceux qui n'ont pas cet article dans leur kit, la même technique est aussi expliquée avec des boutons "classiques". Bonne semaine à vous.

Merci à Glenn Smith pour ce cours!

II.4.3. Travaux pratiques

II.4.3.0.1. TP (1/2) À FAIRE POUR LA SEMAINE 10

Cette semaine, nous revenons à nos histoires de feu avec une barrière et un compteur de voiture. Grâce au cours de Glenn la semaine passée, le fonctionnement des écrans LCD ne devrait plus avoir de secrets pour vous. Voici donc les consignes:
Le montage à réaliser devra comporter:

- Un servomoteur qui jouera le rôle de barrière;
- Un écran LCD qui viendra afficher le nombre de voitures passées depuis que l'Arduino est branché;
- Un bouton pour demander l'ouverture de la barrière (et qui viendra incrémenter le compteur de voitures);
- Un feu bicolore qui passera au vert lorsque la barrière sera complètement ouverte.

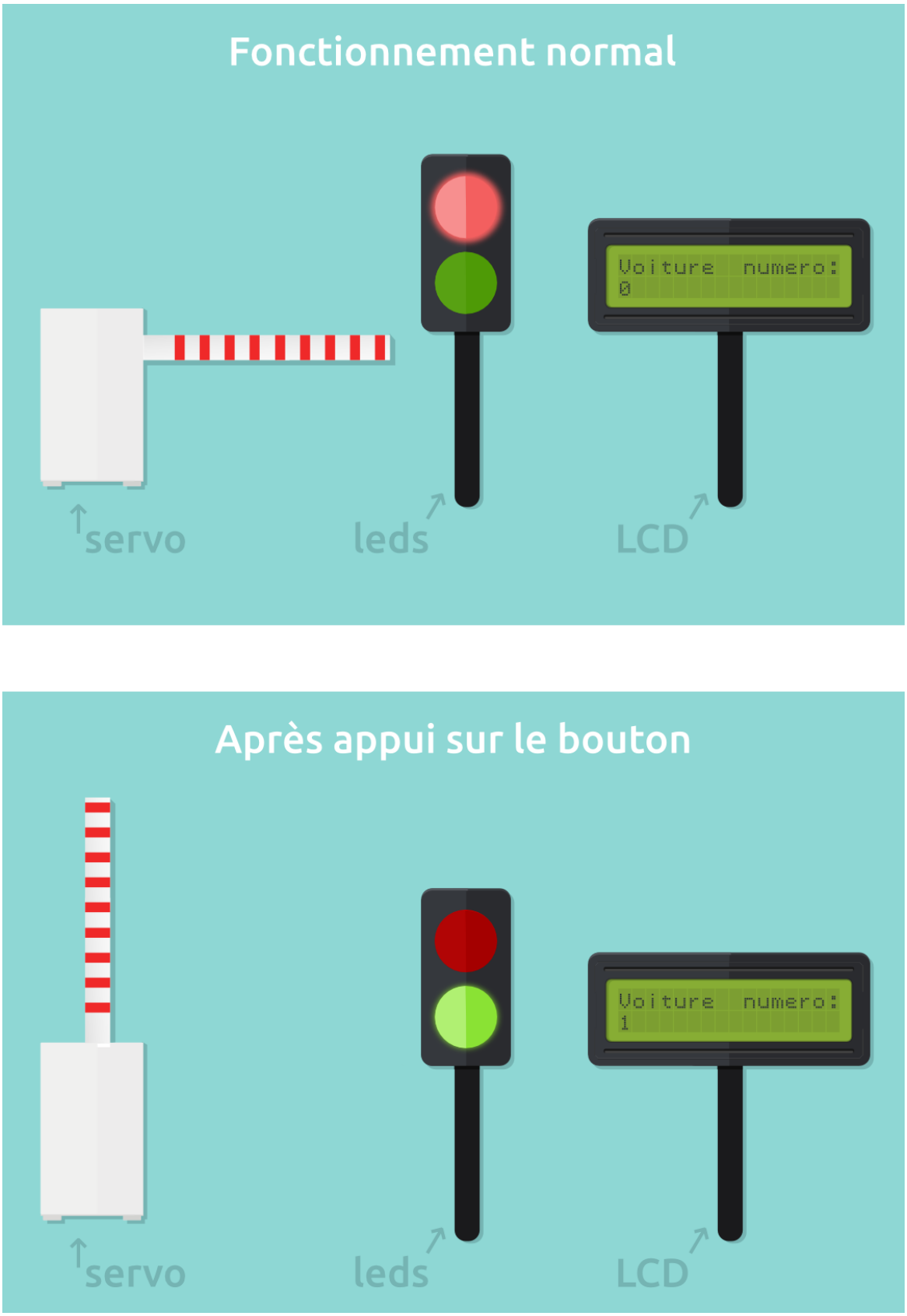


FIGURE II.4.2. – Fonctionnement de l'exercice

Le scénario sera le suivant:
Le fonctionnement normal est un feu allumé au rouge, une barrière fermée (0°) et un afficheur LCD qui affiche sur la première ligne "Voiture numéro:" puis sur la seconde ligne le nombre de

fois où le bouton a été appuyé depuis le démarrage de l'Arduino. Le fonctionnement normal est interrompu par l'appui sur un bouton poussoir.
Si l'appui du bouton est détecté, alors la barrière (actionnée par le servomoteur) se relève doucement. Lorsque la barrière est à la verticale (90°), le feu vert s'allume pendant 5 secondes pendant lesquelles la barrière reste ouverte (90°). Après les 5 secondes, le feu repasse au rouge, la barrière redescend doucement et on vient ajouter un au compteur affiché sur la seconde ligne de l'écran LCD. Enfin, le fonctionnement normal reprend.
Impératif: Votre code devra contenir une fonction appelée `mouvementBarriere()` prenant deux paramètres: la position de départ et la position d'arrivée. Cette fonction devra être capable de lever ou baisser la barrière en fonction de la valeur de ces paramètres.
Aussi, nous souhaitons recevoir le message "Voiture numéro:" suivi du nombre d'appuis sur le bouton dans le moniteur série lorsque l'appui a été détecté.

II.4.3.0.1.1. Quelques indices

Vous aurez besoin de mobiliser toutes les compétences vues ces dernières semaines pour réaliser ce TP:

- L'utilisation de boucles `for` ou `while`;
- L'utilisation d'entrée et de sorties numériques;
- Utilisation de librairies et d'un servomoteur;
- Utilisation des instructions `Serial`;
- Déclaration d'une fonction;

II.4.3.0.1.2. Quelques conseils

- Avertissez-nous sur le fil de discussion ci-dessous si les consignes ne vous semblent pas claires;
- N'allez pas regarder la solution sur Internet sinon il n'y a pas de fun;
- Prenez toujours les hypothèses qui vous arrangent ;
- Seules les notions abordées dans le cours et sur cette page sont nécessaires pour mener à bien ce TP;
- Il n'y a pas une mais plusieurs solutions à chaque problème. La meilleure est celle que vous comprenez!

Bon courage et bonne semaine,
L'équipe du MOOC

II.4.3.0.2. TP (2/2) complémentaire

Grâce à Slic3r, générez le `.gcode` de votre modèle 3D d'avion conçu la semaine dernière avec Blender. N'hésitez pas ensuite a partager sur le forum ou sur Twitter avec le hashtag #MoocFab ou sur la communauté Google+ .

II.5. Semaine 10 : Les fraiseuses numériques

Introduction

Nous continuons cette semaine notre découverte des machines utilisées dans les Fablabs avec une vidéo sur les fraiseuses numériques. Contrairement aux imprimantes 3D qui viennent ajouter du plastique pour créer des objets, les fraiseuses numériques permettent de soustraire de la matière pour usiner des pièces mécaniques, des circuits imprimés et beaucoup d'autres choses décrites dans la vidéo!

II.5.1. Corrigé du TP : Feu bicolore, barrière et LCD

II.5.1.0.1. Corrigé du TP : Feu bicolore, barrière et LCD

Voici la correction du TP de la semaine dernière qui reprend des éléments du TP Feu bicolore et barrière et du cours sur les écrans LCD .

II.5.1.0.1.1. Code Voici une des solutions possibles pour répondre au problème dans l'état actuel de nos connaissances:

```
/*
  Feu bicolore, barrière et LCD

 TP de la semaine 9 du MOOC "La Fabrication Numérique"

 Le montage :
 * Une LED rouge sur la broche 8 en série avec une résistance de 220Ω
 * Une LED verte sur la broche 9 en série avec une résistance de 220Ω

 * Un servomoteur branché sur les broches 10, +5V et GND

 * Bouton poussoir branché sur la broche 7 depuis +5V
 * Une résistance de 1KΩ branchée sur la broche 2 depuis GND
 * Un écran LCD branché sur les broches 2, 3, 4, 5, 11, 12

 créé le 9 Mai 2014
 par Baptiste Gaultier

 Ce code est en CC0 1.0 Universal
```

```
20
21  */
22
23 #include <Servo.h>
24 #include <LiquidCrystal.h>
25
26 Servo servo;  // création de l'objet servo issu
27               // du moule Servo
28
29 LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
30
31
32 // Initialisation des constantes
33 const int bouton = 7;
34
35 const int ledRouge =  8;
36 const int ledVerte =  9;
37
38 // Déclaration des variables :
39 int etatBouton = 0;
40 int nombreDeVoitures = 0;
41
42 // le code dans cette fonction est exécuté une fois au début
43 void setup() {
44   // on souhaite communiquer avec l'ordinateur
45   Serial.begin(9600);
46
47   // Initialisation avec nombres de colones et lignes
48   lcd.begin(16, 2);
49   // Afficher sur la ligne du haut le message :
50   lcd.print("Voiture numero:");
51
52   // indique que les broches des LED
53   // sont des sorties :
54   pinMode(ledRouge, OUTPUT);
55   pinMode(ledVerte, OUTPUT);
56
57   // indique que la broche bouton est une entrée :
58   pinMode(bouton, INPUT);
59
60   // on accroche notre servomoteur branché sur la broche 9
61   servo.attach(10);
62
63   // allume le feu rouge
64   digitalWrite(ledRouge, HIGH);
65
66   // positionne la barrière horizontalement
67   servo.write(0);
68 }
69
```

```
70 // le code dans cette fonction est exécuté en boucle
71 void loop(){
72   // read the state of the pushbutton value:
73   etatBouton = digitalRead(bouton);
74 
75   // si le bouton est appuyé
76   if (etatBouton == HIGH) {
77     // alors on envoie un message sur le moniteur série
78      Serial.print("Bouton appuye");
79      nombreDeVoitures = nombreDeVoitures +1;
80     // puis on remonte la barrière de 90°
81     mouvementBarriere(0, 90);
82 
83     // puis on allume le feu vert durant 5 secondes
84     digitalWrite(ledRouge, LOW);
85     digitalWrite(ledVerte, HIGH);
86     delay(5000);
87 
88     // et on repasse au rouge
89     digitalWrite(ledVerte, LOW);
90     digitalWrite(ledRouge, HIGH);
91 
92     // enfin, on redescend la barrière
93     mouvementBarriere(90, 0);
94 
95   }
96   lcd.setCursor(0, 2);
97   lcd.print(nombreDeVoitures);
98 }
99 
100 void mouvementBarriere(int debut, int fin) {
101   int pos = debut;
102   if(debut < fin) {
103     for(pos = debut; pos < fin; pos++) {
104       servo.write(pos);
105       delay(15);
106     }
107   }
108   else {
109     for(pos = debut; pos>=fin; pos--) {
110       servo.write(pos);
111       delay(15);
112     }
113   }
114 }
```

Listing 15 – Une solution au TP

II.5.1.0.1.2. Montage

FIGURE II.5.1. – Montage - Feu bicolore, barrière et LCD

II.5.1.0.1.3. Schéma

FIGURE II.5.2. – Schéma - Feu bicolore, barrière et LCD

Pour réaliser ce montage, vous avez besoin de:

- Un Arduino;
- Une platine de prototypage;
- Un câble USB;
- Une résistance de 10kΩ;
- Deux résistances de 220Ω;
- Des fils de prototypage;
- Une photorésistance;
- Un servomoteur;
- Un bouton poussoir;
- Une LED rouge;
- Une LED verte;
- Une écran LCD;
- Du temps

Notre correction s'arrête là. Pour des éclaircissements, nous vous invitons à poser vos questions sur le forum et à reparcourir le cours.
On vous laisse avec le montage un peu fouilli de Baptiste:

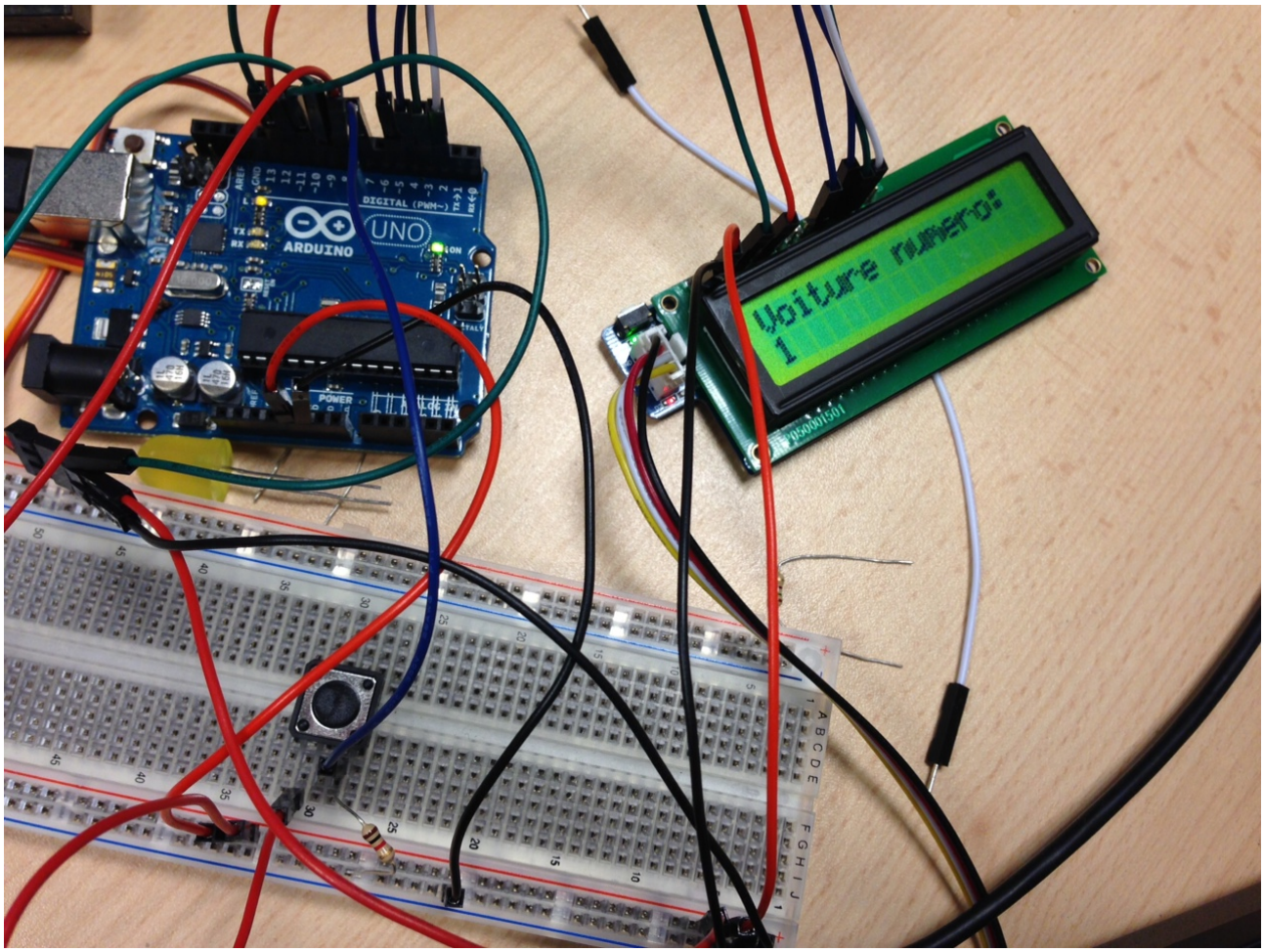


FIGURE II.5.3. – Montage de Baptiste

Ainsi qu'un affichage sur l'écran LCD rétroéclairé de ylc:

FIGURE II.5.4. – Aperçu de l'écran

II.5.2. Les fraiseuses numériques

ÉLÉMENT EXTERNE (VIDEO) —
Consultez cet élément à l'adresse https://geo.dailymotion.com/player.html?video=x2kgzwx&.

Merci d'avoir regardé cette vidéo sur les fraiseuses numériques. Cette vidéo décrit brièvement les principes de fonctionnement de ces machines.
Comme il existe de très nombreux logiciels pour passer d'un fichier informatique aux instructions machine (le fameux `.gcode` dont nous parlions la semaine dernière), nous n'avons pas souhaité rentrer dans la partie logicielle des fraiseuses. Nous vous invitons plutôt à visiter un FabLab qui dispose d'une machine de ce type et de regarder la vidéo ci-dessous produite par Dimitri du FabLab de Lyon .

II.5.2.0.0.1. Références

— Une excellente vidéo sur les fraiseuses (youtube) produite par Dimitri du FabLab de Lyon \

Merci à Arthur Wolf et Stéphane Philippe, adhérents du TyFab et fondateurs de Ipso Factio

. Merci également aux équipes des Petits Débrouillards Bretagne pour nous avoir permis de tourner au TéléFab à Brest.

II.5.2.0.0.2. La fraiseuse numérique en 3 parties par Dimitri

Élément externe (Video) —
Consultez cet élément à l'adresse https://www.youtube.com/embed/81QJWwqnRjY?feature=oembed.

II.5.3. Arduino et I²C

II.5.3.1. Le bus I²C (Inter Integrated Circuit)

II.5.3.1.1. Les bits and bytes (and wires)

Commercialisée à la fin des années 80 par Philips, cette norme de communication "inter-circuits-intégrés" a été mise à jour deux fois durant les années 90 et est devenue un standard non seulement dans l'industrie mais aussi pour les amateurs. Souvent poussée beaucoup plus loin que l'idée d'origine: un système de branchements et un protocole de communication destiné aux liaisons courtes et entre circuits intégrés, elle est désormais utilisée comme interface pour les périphériques de tous genres et c'est même le point de départ pour d'autres normes de communication série (le bus CAN, par exemple). Moi-même j'ai utilisé le I²C 'tel quel' pour la communication entre modules de gestion sur une moto de compétition.
Côté branchements le bus I²C séduit car, en dehors de l'alimentation et le retour 0v, il n'y a besoin que de deux conducteurs: les données, appelées SDA, et une horloge appelée SCL.
Le protocole de communication est relativement simple mais est conçu de sorte que plusieurs maîtres et plusieurs esclaves peuvent cohabiter sur le même bus sans soucis de collision. Eh oui, il est malheureusement impossible d'aborder ce sujet sans parler de choses très "époque coloniale": les maîtres et les esclaves.

II.5.3.1.1.1. Maîtres et esclaves Un maitre dans ce contexte est celui qui donne des ordres et, surtout, celui gère l'horloge - qui cadence le temps. D'habitude c'est le microcontrôleur. Un esclave est représenté par tout autre circuit ou périphérique connecté sur le bus. Avec une seule ligne de données (SDA) la communication est ce qu'on appelle half-duplex: la communication se passe dans un sens à la fois. Regardons la connectique d'un peu plus près:

II.5.3.1.1.2. Connectique I²C Voici comment on branche les périphériques sur le bus. Pour fonctionner, bien sûr, il faut qu'il y ait aussi un retour (GND) commun pour tout le monde.

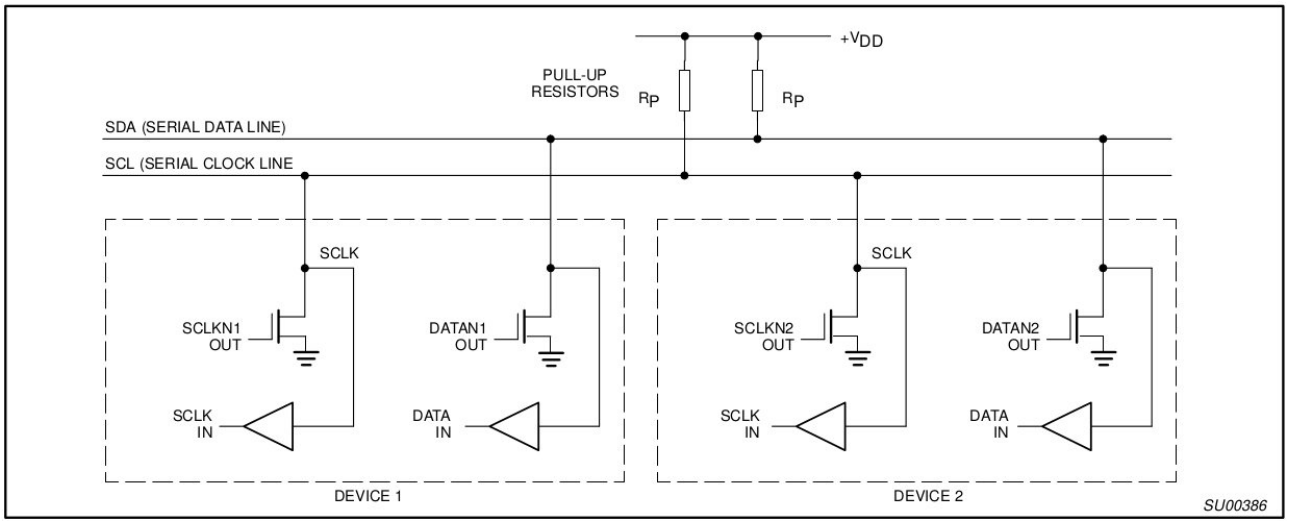


FIGURE II.5.5. – Branchements I2C

(image: source datasheet Philips)
Toutes les connexions SDA et toutes les connexions SCL sont reliées entre elles. Pour l'ensemble des circuits, on ajoute une résistance 'pull-up' sur la ligne SDA et une sur la ligne SCL. La valeur des deux résistances aide à optimiser le bus pour une meilleure performance, mais dans la plupart des cas des valeurs entre 4k7 et 10k donnent de bons résultats. Oui, parfois le bus fonctionne sans ces deux résistances, mais leur présence est TRÈS importante. Ne lésinez pas sur vos montages. Alors, si tout le monde est branché ensemble, comment démêler la communication? Comment sait-on qui dit quoi?

II.5.3.1.1.3. C'est moi le maître Le protocole est défini de sorte qu'à un moment donné il n'y ait qu'un maître, et c'est lui qui dicte sa loi pendant la durée de son 'mandat'. À partir du moment où un maître 'prend le bus', il est seul maître à bord tant qu'il ne le relâche pas. Chaque esclave est codé avec une adresse, et tous écoutent afin de savoir si leur numéro est appelé. Voici une photo d'un dialogue typique I2C:

FIGURE II.5.6. – Des trames de dialogue I2C

Lisons de gauche à droite: d'abord la condition de **démarrage** (Start): un maître prend le contrôle du bus en tirant la ligne de données à 0v pendant que tout le monde dort (toutes les sorties sont tirées par défaut à 5v par les deux résistances). C'est annoté sur cette trace par le point vert. Pour communiquer, le maître fait cycler la ligne d'horloge SCL. Chaque bit à transmettre ou à recevoir est validé sur les fronts montants du signal SCL. A partir

de l'instant "Start", les esclaves écoutent pour savoir si leur adresse est annoncée. À la fin de l'adresse, un bit indique si le maître va envoyer des données à l'esclave, ou si l'esclave doit envoyer les données. Comme avec un talkie-walkie ou la phraséo VHF marine… Dans l'exemple ici, l'adresse annoncée est 0x68 (héxadécimal) ou 104 (décimal). Le drapeau lecture/écriture est à "0", ce qui veut dire que c'est le maître qui va parler à l'esclave à l'adresse 0x68. Si l'esclave est bien à l'écoute, et prêt à entendre les bonnes paroles, il signale sa présence en tirant la ligne SDA à 0v pendant le 9ème cycle. Le maître peut continuer à lui envoyer les données. La fin du "mandat", la condition **terminé** (Stop), est signalée par le relâchement de SDA après que SCL soit stabilisé à son niveau haut. On la voit ici repérée par le point rouge. Dans cet exemple, le maître envoie deux octets de valeur zéro à l'esclave avant de terminer le message.

II.5.3.1.1.4. Vous avez dit barbant ? Eh oui, c'est assez laborieux tout cela! Heureusement il y a Findu… non, je veux dire qu'heureusement nous avons déjà une bibliothèque prévue pour les interfaces 2-fils: **Wire.h** Nous n'avons plus qu'à lui donner l'adresse de l'esclave à qui on veut parler, et c'est la bibliothèque qui s'occupe du reste.

II.5.3.1.2. Un exemple concret

Pour expérimenter avec l'I²C, rien de mieux que de prendre un circuit intégré typique et de le faire parler! J'ai choisi pour vous le DS1307, une horloge / calendrier / mémoire. Vous pouvez l'acheter complètement nu, une puce DIL-8 qui se branche directement sur une platine d'expérimentation, ou pré-monté sur un circuit imprimé (appelé souvent un *breakout*), souvent avec une pile de sauvegarde 3v. Voici les deux bêtes côte à côte:

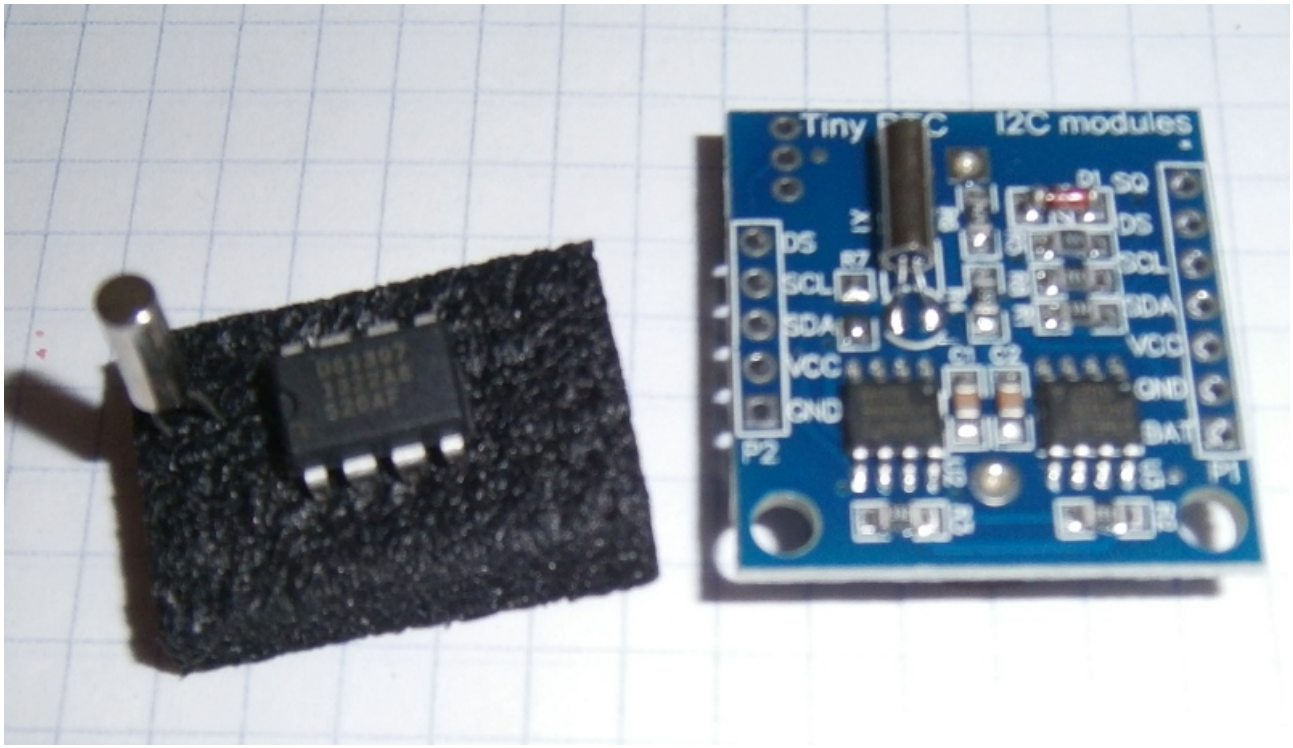


Figure II.5.7. – Le DS1307

(désolé pour la mise au point…)
À côté du DS1307 'tout nu' dans son lit douillet j'ai mis le quartz de 32KHz qu'il faut utiliser pour le faire fonctionner. Sur le breakout on distingue (à peine, je sais) une deuxième puce:

c'est une EEPROM qui dialogue aussi avec l'I²C. L'emplacement pour la pile est sur l'autre face de la platine. Les branchements VCC, GND, SDA et SCL sont clairement repérés sur la platine, il n'y a qu'à les relier à l'Arduino car la platine est aussi équipée des deux résistances de pull-up (un souci, d'ailleurs, si on veut mettre plusieurs platines ensemble sur le même bus…). Voici le câblage sur la platine d'expérimentation:

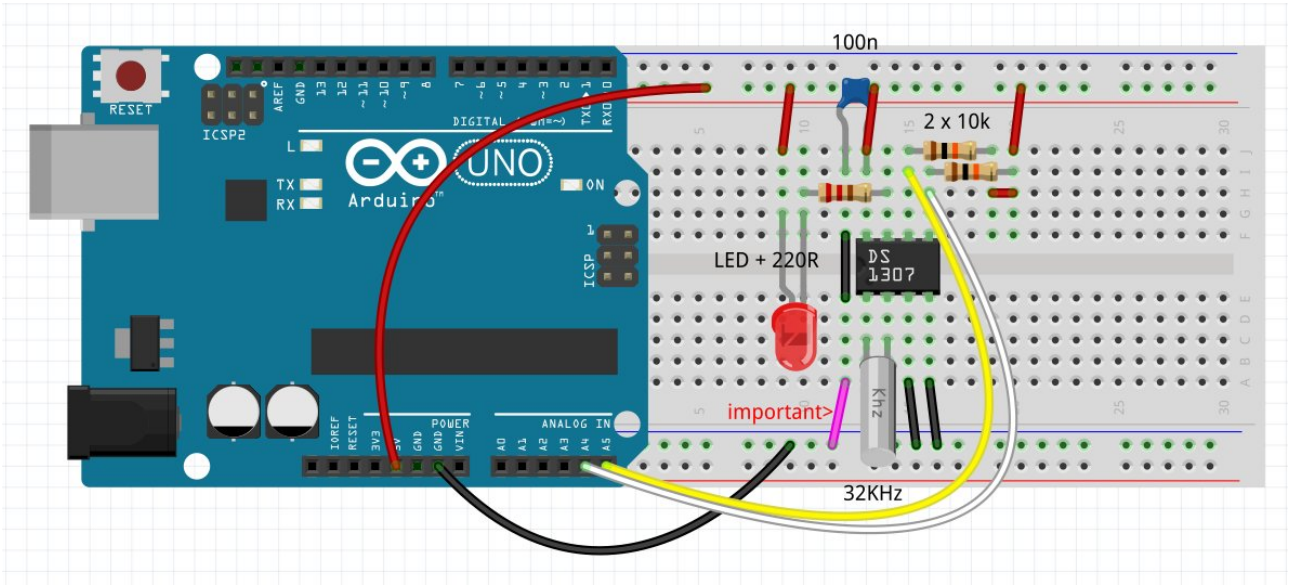


FIGURE II.5.8. – Montage du DS1307

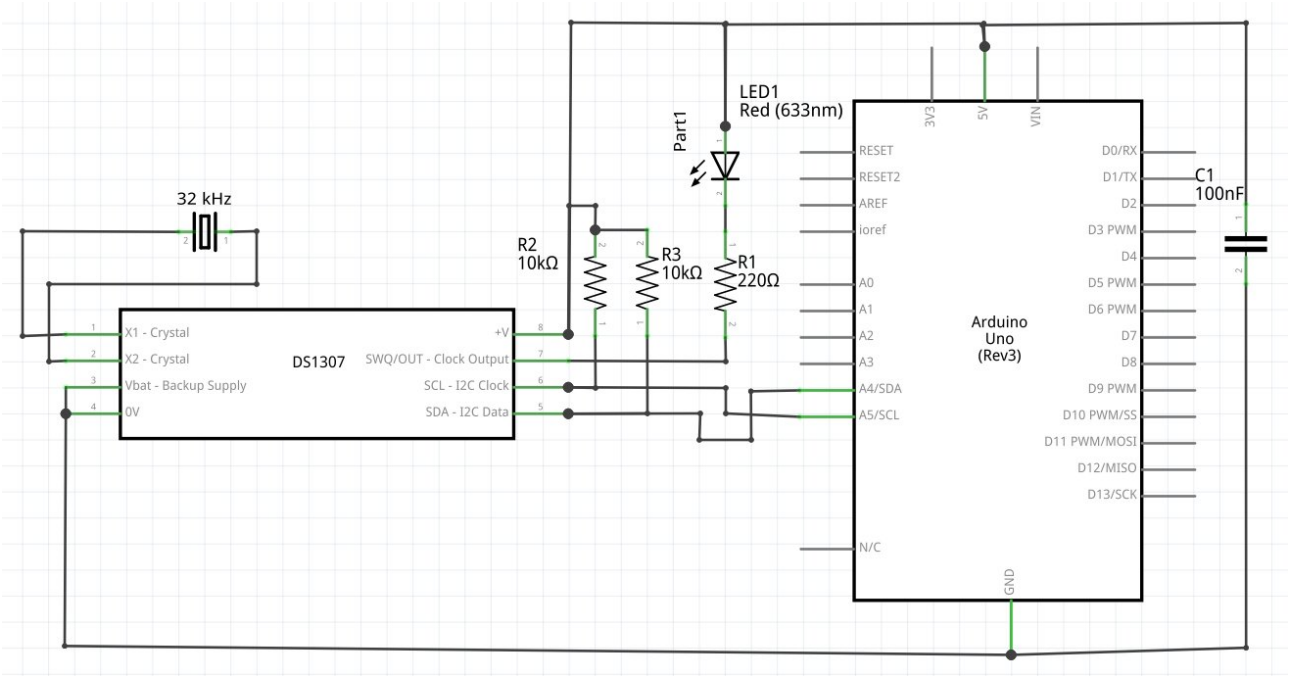


FIGURE II.5.9. – Schéma électronique du câblage du DS1307

Pas de soucis particuliers. Sur la platine vous remarquerez un fil noté "important". Non, ce n'est pas la couleur qui est importante, mais le fait de placer un fil de masse sur la rangée à côté du quartz. Cela aide à la stabilité des oscillations et ainsi à la précision de l'horloge. J'ai remarqué que le DS1307 était TRÈS sensible aux parasites, et aussi très sensible à la précision du quartz.

II.5.3.1.2.1. Le Sketch La première ligne de notre Sketch est la déclaration de la bibliothèque :

```
1 #include <Wire.h>
```

Ensuite, il faut déclarer l'adresse de notre esclave. Pour connaître les adresses des circuits intégrés, il est **obligatoire** d'aller consulter la documentation technique. Pour le DS1307 je l'ai fait pour vous: son adresse est 0x68. (Vous avez déjà vu la notation "0xnn" qui exprime un nombre en hexadécimale, non ?) Déclarer l'adresse avec un `#define` ou un `const` au choix.

```
1 #define  addr_ds1307 0x68
```

Pour initialiser l'environnement, c'est exactement comme pour la liaison série:

```
1 Wire.begin();
```

Et l'on peut sauter tout de suite dans la communication:

```
1 Wire.beginTransmission(addr_ds1307);
2 Wire.write(0); // adresse du registre
3 Wire.write(0); // octet à écrire
4 Wire.endTransmission();
```

Mais, un instant… On ne sait même pas comment dialoguer avec cette bête! Il faut regarder encore dans la documentation. Le principe de la plupart des circuits compatibles I²C est que les fonctionnalités et paramètres sont accessibles comme étant des *registres*, ou des locations de mémoire, à l'intérieur du circuit. Le DS1307 n'est pas une exception. Pour les fonctions d'horloge / calendrier, 8 registres de 0x00 à 0x07 nous intéressent:

Table 2. Timekeeper Registers

ADDRESS	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0	FUNCTION	RANGE
00h	CH	10 Seconds			Seconds				Seconds	00–59
01h	0	10 Minutes			Minutes				Minutes	00–59
02h	0	12	10 Hour	10 Hour	Hours				Hours	1–12 +AM/PM 00–23
		24	PM/ AM							
03h	0	0	0	0	0	DAY			Day	01–07
04h	0	0	10 Date		Date				Date	01–31
05h	0	0	0	10 Month	Month				Month	01–12
06h	10 Year				Year				Year	00–99
07h	OUT	0	0	SQWE	0	0	RS1	RS0	Control	—
08h–3Fh									RAM 56 x 8	00h–FFh

0 = Always reads back as 0.

FIGURE II.5.10. – Registres du DS1307

Les adresses 0x00 et 0x07 servent de registres de configuration. Les autres servent à initialiser la

date et l'heure et, une fois l'horloge démarrée, pour lire la date / heure actuelle. Ce qui n'est pas clairement annoncé dans ces informations, c'est que les valeurs sont stockées en **BCD**: "décimale codée en binaire". C'est quoi? Prenons l'exemple de la valeur 65. En binaire 'normale', 65 = B01000001 ou 0x41. En BCD on fait en sorte que chaque demi-octet représente un chiffre de 0 à 9, pour faire une gamme de valeurs de 0 à 99. Dans ce système, 65 devient B01100101 ou 0x65.
Pour accéder aux registres, il faut faire comme suit:

- Annoncer l'adresse du circuit en mode écriture;
- Envoyer l'adresse du registre auquel accéder;
- Envoyer l'octet de données à stocker dans le registre.

Presque tous les circuits augmentent d'un le "register address" automatiquement après chaque écriture ou lecture: si l'on veut envoyer plusieurs octets, il suffit de les envoyer dans l'ordre des registres. Puis, comme nous l'avons vu tout à l'heure, il faut signaler la fin de transmission. Si maintenant on regarde à nouveau le dernier morceau de code que j'ai affiché, c'est peut-être un peu plus clair, non?
La lecture des registres est assez simple aussi. Le protocole I²C nous demande de suivre cette séquence:

- Annoncer l'adresse du circuit en mode **écriture**;
- Envoyer l'adresse du registre à accéder;
- Annoncer l'adresse du circuit en mode **lecture**;
- Annoncer combien d'octets on souhaite lire;
- Lire les octets un à un;
- Terminer la transmission.

On procède avec les commandes suivantes:

```
Wire.beginTransmission(addr_ds1307);
Wire.write(0); // adresse du registre de départ
Wire.endTransmission();
// Demande mode lecture, 7 octets
Wire.requestFrom(addr_ds1307, 7);

for (i=0; i<7; i++) {
    time[i] = bcd2bin(Wire.read());
}

Wire.endTransmission();
```

Je vous laisse découvrir toutes les merveilles du Sketch de cette semaine ci-dessous, mais je voulais aussi vous montrer quelques astuces qui aident à la réalisation de ce logiciel.

Contenu masqué n°6

II.5.3.1.2.2. Conversions BCD Deux petites fonctions assez compactes pour convertir des valeurs en BCD et du BCD en 'vrai':

```
/* Conversion d'une valeur BCD en décimale */
static uint8_t bcd2bin (uint8_t val) {
    return val - 6 * (val >> 4);
}

/* Conversion d'une valeur décimale en BCD */
static uint8_t bin2bcd (uint8_t val) {
    return val + 6 * (val / 10);
}
```

Listing 16 – Convertisseur BCD <-> Decimale

Si on prendre notre exemple de tout à l'heure du numéro 65, la fonction **bin2bcd** divise la valeur par 10, la multiplie par 6 et ajoute la valeur d'origine. Cela donne: $6 \times 6 + 65 = 101$. En hexadécimale cela donne 0x65!
Regardons la fonction inverse. L'opération **(val » 4)** fait déplacer les bits de l'octet par 4 places vers la droite. C'est comme si on prenait le chiffre des dizaines. Et donc cela donne $101 - (6 \times 6) = 65$. CQFD!

II.5.3.1.2.3. Tableaux 2D François, alias *fb251*, nous a documenté une première utilisation des tableaux avec son Sketch hyper puissant. Je vais vous montrer une autre utilisation des tableaux, les tables de conversion. Voici un exemple:

```
/* Tableaux de conversion des jours */
const char jours[7][10] = {
    "Dimanche",
    "Lundi",
    "Mardi",
    "Mercredi",
    "Jeudi",
    "Vendredi",
    "Samedi"
};
```

Listing 17 – Tableau de conversion des jours

Nous avons vu les tableaux à une dimension, comme char message[21]. Ce qui est déclaré dans l'exemple c'est un tableau à deux dimensions: on peut l'imaginer comme 7 lignes de 10 colonnes. La déclaration fait aussi l'initialisation de chaque ligne avec les noms des jours de la semaine. De ce fait, si on accède à jours[3] nous allons tomber sur le séquence de caractères **"Mercredi"**. Nous avons construit un tableaux de conversion entre une valeur numérique et un message associé. Vous allez voir dans le Sketch comment cette astuce facilite la tâche de décodage des données stockées dans les registres du DS1307.
Si vous avez des projets qui demandent une horloge fiable et la temporisation de périodes assez longues (des minutes ou heures), je vous conseille un autre circuit - le PCF8583. Un peu plus difficile à dénicher, un peu plus cher, mais moins sensible aux variations de quartz (il peut même être piloté par les 50Hz du secteur). Il fournit des fonctions de réveil par interruption afin de vous libérer de tout souci de temporisation.

Passez une bonne semaine,

Glenn Smith mai 2014

II.5.4. Travaux pratiques

II.5.4.0.1. TP à faire pour la semaine 10

Comme nous avons pu le voir dans les semaines précédentes, le port USB permet d'échanger des informations entre Arduino et l'ordinateur à l'aide de la librairie `Serial`. L'objectif de ce TP est d'utiliser le port série pour envoyer depuis Arduino des valeurs récupérées depuis un capteur (potentiomètre, photorésistance ou autre) et de recevoir et traiter cette information grâce à un logiciel appelé Processing .

FIGURE II.5.11. – Bienvenue à Processing

Ce logiciel (qui ressemble beaucoup au logiciel Arduino en terme de langage et d'interface utilisateur) va nous permettre de recevoir les caractères envoyés par Arduino pour ensuite les traiter. Processing dispose ensuite de nombreuses librairies pour afficher de l'image, jouer du son…
Un article sur le wiki (annexe) détaille son utilisation et un cours est disponible sur OpenClassRoom pour installer le logiciel.
Une fois Processing installé, vous n'aurez pas à taper de code puisque nous vous donnons le programme Processing qui va recevoir les caractères et les traiter. Une fois lancé, le programme suivant affichera une fenêtre plus ou moins blanche selon la valeur envoyée par Arduino:

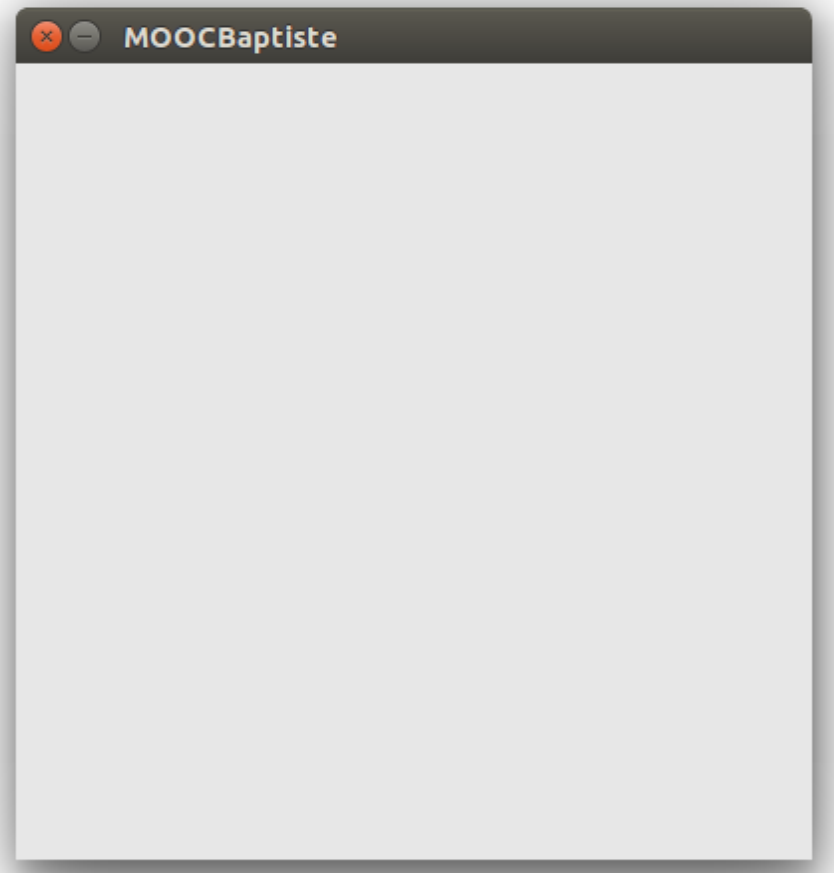


FIGURE II.5.12. – Exemple de résultat

Voici le programme Processing à copier-coller dans votre logiciel:

```
import processing.serial.*;

float valeurArduino = 0; // valeur envoyée par le capteur Arduino

Serial myPort;

void setup() {
  // création d'une fenêtre de 400×400 pixels
  size(400, 400);

  // initialisation du port série pour communiquer avec Arduino
  myPort = new Serial(this, Serial.list()[0], 9600);
  myPort.bufferUntil('\n');
}
```

```
15
16 void draw() {
17   // set the background color with the color values:
18   background(valeurArduino);
19 }
20
21 void serialEvent(Serial myPort) {
22   // récupérer la valeur envoyée par Arduino
23   String inString = myPort.readStringUntil('\n');
24
25   if (inString != null) {
26     // supprimer les potentiels espaces et stocker la valeur
27     // reçue dans la variable valeurArduino
28     valeurArduino = float(trim(inString));
29     valeurArduino = map(valeurArduino, 0, 1023, 0, 255);
30   }
31 }
```

II.5.4.0.1.1. Votre mission (si vous l'acceptez...) Est d'écrire un programme pour Arduino qui viendra lire la valeur d'un capteur analogique (comme une photorésistance ou un potentiomètre) branché sur la broche `A0` et qui va ensuite envoyé cette valeur en utilisant la fonction `Serial.println( )` qui, rappelons-le, permet d'envoyer du texte vers l'ordinateur.

II.5.4.0.1.2. Quelques indices Vous aurez besoin de mobiliser toutes les compétences vues ces dernières semaines pour réaliser ce TP:

- L'utilisation d'une entrée analogique;
- Utilisation de la librairie `Serial`.

II.5.4.0.1.3. Quelques conseils

- Avertissez-nous sur le fil de discussion ci-dessous si les consignes ne vous semblent pas claires;
- N'allez pas regarder la solution sur Internet sinon il n'y a pas de fun;
- Prenez toujours les hypothèses qui vous arrangent 🍊 ;
- Seules les notions abordées dans le cours et sur cette page sont nécessaires pour mener à bien ce TP;
- Il n'y a pas une mais plusieurs solutions à chaque problème. La meilleure est celle que vous comprenez!

Bon courage et bonne semaine,
L'équipe du MOOC

Contenu masqué

Contenu masqué n°6

```
/*
  Démonstration des dialogues I²C avec une DS1307 RTC

  N'oubliez pas les deux résistances "pull-up" si la platine
  'breakout' n'est pas équipé.

  Glenn Smith  mai 2014
  MOOC Fabrication Numérique
*/
#include <stdarg.h>  // pour le fonction "format"
#include <Wire.h>  // Pour le I²C

/* L'adresse de chaque périperique est documenté dans sa fiche
    technique */
#define  addr_ds1307 0x68  // Adresse du DS1307 sur bus I²C

/* Tableaux de conversion des jours */
const char jours[7][10] = {
    "Dimanche",
    "Lundi",
    "Mardi",
    "Mercredi",
    "Jeudi",
    "Vendredi",
    "Samedi"
};

/* Tableaux de conversion des mois */
const char mois[12][10] = {
    "janvier",
    "fevrier",
    "mars",
    "avril",
    "mai",
    "juin",
    "juillet",
    "aout",
    "septembre",
    "octobre",
    "novembre",
    "decembre"
};

/*
  Vous pouvez corrigé les valeurs ici pour regler date et heure
```

```
*/
//               Secs, Mins, Hrs, Jour, Date, Mois, (Année - 2000)
byte  time[7] = {0,    0,    13,  1,    25,   5,    14};

char tempString[32]; // pour le message formaté

void format(char *fmt, ... ){
    va_list args;
    va_start (args, fmt );  // allocation d'espace en mémoire
    vsnprintf(tempString, 32, fmt, args);  // formattage du
        message dans tempString
    va_end (args);  // liberation des ressources.
}

/* Conversion d'une valeur BCD en décimale */
static uint8_t bcd2bin (uint8_t val) {
    return val - 6 * (val >> 4);
}

/* Conversion d'une valeur décimale en BCD */
static uint8_t bin2bcd (uint8_t val) {
    return val + 6 * (val / 10);
}

void start_timer() {
    // Configurer la sortie pour 1Hz
    Wire.beginTransmission(addr_ds1307);
    Wire.write(0x07);  // addresse du registre
    Wire.write(0x10);  // Byte to write
    Wire.endTransmission();
    // Effacer le contenu du registre 0.  Ceci pour but de
    // démarrer l'horloge.
    Wire.beginTransmission(addr_ds1307);
    Wire.write(0);  // addresse du registre
    Wire.write(0);  // octet à écrire
    Wire.endTransmission();
}

void regler_horloge(byte newtime[7]){
    byte i;

    Wire.beginTransmission(addr_ds1307);
    Wire.write(0);  // addresse du registre de départ
    for (i=0; i<7; i++) {  // envoyer les 7 octets un-a-un
        Wire.write(bin2bcd(newtime[i]));
    }
    Wire.endTransmission();
}

void  lire_horloge() {
```

```
94      byte i;
95
96      Wire.beginTransmission(addr_ds1307);
97      Wire.write(0);  // adresse du registre de départ
98      Wire.endTransmission();
99      // Demande mode lecture, 7 octets
100     Wire.requestFrom(addr_ds1307, 7);
101     // lire les octets dans l'ordre
102     for (i=0; i<7; i++) {
103         time[i] = bcd2bin(Wire.read());
104     }
105     Wire.endTransmission();
106 }
107
108 // Afficher date et heure sur la liaison série
109 void afficher_lheure () {
110     Serial.print(jours[time[3]-1]);  // le jour
111     Serial.print(" ");
112     Serial.print(time[4]);  // la date du mois
113     Serial.print(" ");
114     Serial.print(mois[time[5]-1]); // le mois
115     Serial.print(" ");
116     Serial.print(time[6]+2000);  // l'année
117     format("  %02u:%02u:%02u", time[2],  time[1],  time[0]);
118     Serial.println(tempString);  // et l'heure hh:mm:ss
119 }
120
121 void setup()
122 {
123     Wire.begin(); // Initialiser l'environnement I²C
124     Serial.begin(9600);
125     Serial.println("Depart");
126     start_timer(); // Démarrer le horloge
127     Serial.println("Horloge OK");
128     //  demande_heure
129     regler_horloge(time);
130     Serial.println("Date et heure OK");
131 }
132
133 void loop()
134 {
135     lire_horloge();
136     afficher_lheure();
137     Serial.println("Attente...");
138     delay(5000);
139 }
```

Listing 18 – Démonstration des dialogues I²C avec une DS1307 RTC

Retourner au texte.

II.6. Semaine 11 : Design d'objets

Introduction

Bienvenue dans la semaine 11. Cette semaine, la designer Bérengère Amiot nous présente un cours consacré au design d'objets. Elle nous offre un rapide tour d'horizon de ce qu'est le design et comment celui-ci peut être pris en compte lors de la création d'un objet.
Cette semaine également, Glenn vous propose un excellent cours sur le Design Électronique .
Enfin, Laurent revient cette semaine avec une vidéo Blender consacrée à la modélisation pour l'impression 3D.
Bonne semaine!

II.6.1. Modélisation 3D pour l'impression 3D

Pour continuer sur la modélisation 3D, Laurent vous propose une vidéo consacrée à Blender pour les imprimantes 3D. Accrochez-vous, ça va vite!

Élément externe (Video) —
Consultez cet élément à l'adresse https://geo.dailymotion.com/player.html?video=x2kgzzk&.

II.6.2. Corrigé du TP Arduino et Processing

II.6.2.0.1. Corrigé du TP Arduino et Processing

Voici la correction du TP de la semaine dernière qui reprend des éléments du cours sur les capteurs numériques et du wiki sur Processing .

Contenu masqué n°7

II.6.2.0.1.1. Montage

FIGURE II.6.1. – Le montage

II.6.2.0.1.2. Schéma

FIGURE II.6.2. – TP9 schéma

Pour réaliser ce montage, vous avez besoin de:
- Un Arduino
- Une platine de prototypage
- Un câble USB branché à votre ordinateur
- Un bouton poussoir
- Des fils de prototypage
- Du temps

Notre correction s'arrête là. Pour des éclaircissements, nous vous invitons à poser vos questions sur le forum et à reparcourir le cours.
On vous laisse avec le montage de Baptiste:

FIGURE II.6.3. – Le résultat

II.6.3. Design d'objets

ÉLÉMENT EXTERNE (VIDEO) —
Consultez cet élément à l'adresse https://geo.dailymotion.com/player.html?video=x2kgzum&.

Merci d'avoir regardé cette vidéo sur les bases du design.

Si vous avez envie d'approfondir vos recherches sur la question, Bérengère vous a concocté une petite bibliographie. Il y a des sources récentes et d'autres plus anciennes. Nous vous invitons à vous rendre dans votre bibliothèque de quartier pour trouver les plus anciens ouvrage.

II.6.3.0.0.1. Bibliographie

— Gaston Bachelard, La poétique de l'espace, Paris, Presse universitaire de France, 1972.
— Barry Bergdoll (dir.), From Inspiration to Innovation Nature Design, Zurich, Lars Muller Publishers, 2007.
— Lucius Burckhardt, Le design au-delà du visible, Paris, Centre Pompidou, 1991.
— Marie-haude Caraës et Chloé Heyraud, Jardin et design, Barcelone, Cité du design-Actes Sud, 2010.
— Antoine Fenoglio et Frédéric Lecourt, L'objet du design, St Just la Pendue, Bibliothèque nationale de France, 2009.
— Hella Jongerius, Misfit, Londre, Phaidon, 2010.
— Gérard Laizé et Frédéric Loeb, Se nourrir de la nécessité à la convivialité, Paris, Distill et Via, 2010.
— Philippe Louguet, Le design au cœur des paradoxes d'un monde en mutation, hors-série Intramuros, janvier 2009.
— Victor Papanek, Design pour un monde réel, Paris, Mercure de France, 1974.
— Georges Perrec, Espèce d'espace, Paris, Galilée, 1974.
— Chantal Prod'Hom (dir.), Nature en kit, Lausanne, Infolio edition, 2009.
— Constance Rubini (dir.), Dessiner le design, Paris, Les Arts Décoratifs, 2009.
— Gilbert Simondon, Du mode d'existence des objets techniques, Paris, Aubier, 1989.
— John Thackara, In the bubble: de la complexité au design durable, St Etienne, Cité du design, 2008.
— Yves Zarka, Le monde émergent, Paris, Armand Colin, 2010.

Aussi, vous trouverez ci-dessous des sites sélectionnés par Bérengère plus pour s'aérer l'esprit lorsqu'elle travaille sur un nouveau un projet ou juste pour y flâner. Si vous souhaitez faire partager des sites que vous aimez, n'hésitez pas à les indiquer dans la discussion ci-dessous!

II.6.3.0.0.2. Webographie

http://ffffound.com/
http://boingboing.net/
http://mocoloco.com/
http://www.informationisbeautiful.net/

II.6.3.0.0.3. Sites dédié au design

http://www.lelieududesign.com/\
http://www.designenbretagne.com/
http://www.architonic.com/
http://www.core77.com/
http://lovelypackage.com/
http://graphism.fr/

Merci à Bérengère Amiot pour ce cours

II.6.4. Les suites de design électroniques

Prenons un peu de temps aujourd'hui pour partir à la découverte de quelques logiciels pouvant être utiles à tout bricoleur faisant des circuits électroniques: les suites de création de PCB!

Je vous en ai sélectionné quelques-unes, toutes gratuites et allant du niveau débutant jusqu'à une compétence plus professionnelle.

II.6.4.0.0.1. Fritzing

FIGURE II.6.4. – Fritzing

On commence doucement avec un logiciel auquel sont déjà habitués certains hobbyistes: Fritzing
!
Cet outil possède un avantage assez rare, il dispose d'un rendu type "dessin" très appréciable pour faire des schémas simples et visuels pour des débutants. Il permet ainsi de voir "quel fil va où ?!". C'est une fonction assez unique dans le monde des outils d'électronique (que les autres logiciels présentés dans ce chapitre ne proposeront pas).
Un autre atout est sa prise en main aisée et rapide même pour un néophyte. Enfin, il est assez complet en terme de fonctions pour prétendre pouvoir réaliser des schémas électroniques et des typons.
Mais (car il y a un mais), tout cela vient à un prix. En effet, le côté obscur du *user-friendly* est bien sûr qu'il devient plus délicat de faire des choses un peu plus poussées. La bibliothèque de composants de base n'est pas très fournie (très orientée Arduino) et l'outil de schéma électronique est perfectible (fil qui ne s'aligne pas toujours proprement, pas de label, etc.).
Ce logiciel est donc très bien pour faire des schémas simples ou visuels, mais sera vite limité pour quelqu'un voulant faire quelque chose de plus avancé.
Enfin, il est bon de savoir que le programme est multi-plateforme (pratique !) et *open source*.
N'ayant pas testé la fonction de création de PCB, je ne jugerai pas cette dernière.

II.6.4.0.0.2. Kicad

FIGURE II.6.5. – Kicad

Commençons tout de suite à parler de ce qui le démarque des autres: c'est un outil français!
Voilà, fin du chauvinisme, parlons un peu du logiciel… Voici un lien vers la page de ce dernier: http://www.kicad-pcb.org/
Tout d'abord, sachez qu'il a été développé pour être fourni de manière gratuite dès le début. Plutôt un bon point et du coup il est facile de trouver de l'aide dessus sur Internet.
Son interface est similaire à celle que l'on peut trouver sur d'autres programmes du même calibre. Là on parle bien de logiciel pour personnes un peu plus habituées à l'électronique puisqu'il n'y a plus de glisser-déposer de dessins sur une feuille. Ici c'est du schéma électronique, du vrai, et ce sera comme ça aussi pour les autres logiciels dessous.

En plus du schéma électronique, on peut bien sûr réaliser un PCB (typon). Ce dernier peut d'ailleurs être visualisé en 3D si les modèles des composants sont fournis.
Je ne sais pas trop pourquoi, je n'ai jamais trop accroché à ce soft. L'interface est pourtant similaire à celle de ses confrères et les fonctionnalités aussi, mais je n'ai pas trop d'idées sur ce qui me freine… Je ne pourrai donc pas trop vous parler plus en détail! Là encore, c'est un logiciel multi-plateforme.

II.6.4.0.0.3. DesignSpark

FIGURE II.6.6. – Design Spark

Au suivant! DesignSpark (“l'étincelle du design” ou “l'étincelle de la création” si l'on voulait traduire). Derrière ce nom éloquent se cache un programme codé par les ingénieurs de chez RadioSpares. Oui, je parle bien du fournisseur de composants. En effet, RS a décidé de proposer son propre outil et de fonder une communauté autour afin de fédérer des utilisateurs et probablement ainsi étendre son marché. Et ça marche! Le logiciel n'est pas désagréable à utiliser (demande un peu de prise en main car les raccourcis clavier sont différents des autres logiciels) et est plutôt clair et fonctionnel. La bibliothèque de composants de base est correctement fournie et, gros point positif, le logiciel embarque un outil de création de composant plutôt pas mal faite du tout et facile à utiliser. Il devient ainsi simple de créer un composant depuis une datasheet. Comme pour le précédent, on retrouve une vue 3D si le composant possède un modèle. Sinon rien de plus à redire, il respecte les mêmes codes et conventions que ses concurrents.
Un point noir cependant pour moi qui travaille sous Linux: il n'est disponible que pour Windows et l'équipe n'a apparemment pas prévu de portage. Dommage à l'heure où une majeure partie des bidouilleurs/bricoleurs se retrouve sous des systèmes comme Ubuntu.

II.6.4.0.0.4. Eagle

FIGURE II.6.7. – Eagle

Passons au dernier programme de cet article Eagle . Ce logiciel est à l'origine dédié au marché des professionnels. Il est donc nécessaire d'avoir une licence (chère) pour l'utiliser à volonté. Cependant, devant l'ampleur du phénomène “Maker”, l'éditeur a décidé de proposer une version gratuite et sans limite dans le temps ou dans les fonctionnalités. Ou presque. Les seules limites sont représentées par l'utilisation d'une seule feuille par schéma et on ne peut faire que des typons de deux couches (ce qui est la limite de faisabilité pour un process “fait main”). On ne peut aussi que créer un PCB de 100mm par 80mm.

Au-delà de ces limites, on se retrouve avec un logiciel pleinement fonctionnel, professionnel et assez bien fourni. On retrouve les mêmes fonctions que sur les deux précédents (schéma, typon, 3D…).
Point intéressant, c'est ce programme qui est utilisé par Arduino et LadyAda. On retrouve donc de nombreux schémas électroniques et typons fournis par ces derniers que les bricoleurs peuvent ensuite reprendre pour leur propre utilisation.
Enfin, il est multi-plateforme ou presque. Il se sert d'une sorte d'émulation Windows pour fonctionner sous Linux, mais ce n'est pas vraiment handicapant.

Il existe bien sûr plein d'autres logiciels et je ne peux pas faire une liste exhaustive ne pouvant pas tous les tester et n'étant pas non plus électronicien de métier, je ne saurais faire une analyse vraiment poussée de chacun de ces logiciels.
Le mieux encore est de tester et de découvrir ce qui vous convient le plus. Cependant, pour des "grands débutants" j'aurais tendance à conseiller Fritzing si vous l'utilisez **avec rigueur**. En effet, rien de pire que de lire un schéma dont les fils partent dans tous les sens.
Pour les personnes moins effrayées par les symboles de l'électronique, je vous encourage à vous tourner vers des logiciels comme Eagle dans sa version gratuite. Ce dernier est en effet très utilisé dans le monde professionnel et il est (très) fréquent de trouver des librairies d'utilisateur Arduino et autres outils DIY faits par des gens compétents.

II.6.5. Fabrication Numérique

II.6.5.1. La Fabrication Numérique

II.6.5.1.1. Fabriquer une platine électronique

Au fil des semaines, nous avons vu les bases pour la création des machines à contrôle numérique et nous avons aussi appris quelques notions de conception en 2D et 3D. Dans le monde (un peu parallèle) de l'électronique, la fraiseuse nous apporte un sacré coup de main pour la réalisation des platines. Mais avant d'en arriver à la platine, je voulais vous montrer un petit aperçu des outils et techniques de conception numérique.

II.6.5.1.2. Numérique, mais pas automatique

Comme les antibiotiques, la réalisation électronique n'a rien d'automatique. Pour passer d'une idée jusqu'à l'objet définitif il faut passer par un certain nombre d'étapes. Avec des logiciels professionnels, certaines tâches sont facilitées mais il faut savoir piloter ces différentes étapes.
Voici comment on peut se représenter les choses:

1. J'ai une idée, je fais un croquis sur la nappe de table chez mon meilleur ami (en espérant que la nappe soit en papier, sinon pour déchirer le bout et rapporter le croquis à la maison ça va être dur…)
2. Je réalise un schéma électronique, et je fais un prototype ou une maquette sur une platine d'expérimentation
3. Je crée le programme (Sketch) pour faire fonctionner la maquette et je teste l'ensemble
4. (Probablement quelques itérations des étapes 2 et 3 pour arriver au résultat voulu)
5. Content du résultat, je crée une platine (circuit imprimé) électronique pour faire une version pérenne (ou pour la fabriquer en plusieurs exemplaires, donner les sources en "open source" pour les autres…)

6. Beaucoup de personnes en voudraient, il faut donc fabriquer plusieurs platines afin d'organiser un atelier ensemble

Nous avons vu les étapes 2 et 3 en partie: Fritzing et le simulateur 123D nous permettent de concevoir un montage avec de l'électronique et les éléments intelligents Arduino. Il est même possible de créer des fichiers numériques pour la fabrication de platine avec Fritzing. Pour les réalisations simples, c'est peut-être déjà suffisant. Dès que le montage devient un peu complexe, il faut chercher des outils un peu plus performants.

II.6.5.1.3. Mon ami KiCad

KiCad est un logiciel libre, open source et peut s'adresser aux amateurs et professionnels de l'électronique. Le site officiel est ici . Son niveau et son utilisation sont très proches d'Eagle - une référence pour beaucoup d'électroniciens. Non seulement les outils sont performants, mais KiCad est aussi livré avec des bibliothèques de composants très complets. Comme pour les morceaux de logiciels pour l'Arduino, on trouve beaucoup de fichiers contribués sur le web (symbols, modules, images 3D) facilement intégrables. Des éditeurs permettent la création ou modification de composants non standards ou de récupération. C'est devenu mon ami car, même si l'apprentissage prend un peu de temps, ce logiciel me fait gagner BEAUCOUP de temps et me permet la réalisation de A à Z en numérique.

II.6.5.1.4. Le schéma

Pour vous faire une démonstration, je vais vous montrer un projet que j'ai réalisé avec notre ami Eskimon: un shield télémètre à ultrasons pour Arduino. Allez, on va se jeter à l'eau tout de suite: voici le schéma dans eeschema de KiCad:

Figure II.6.8. – Un shield télémètre à ultrasons sous Kicad

Le bloc "ARDUINO_SHIELD" en bas à gauche est un exemple de module fourni par une tierce personne. Vous verrez que les broches correspondent avec les broches de notre cher Arduino UNO. Bon, l'important ici n'est pas trop le contenu du schéma, mais comment on y parvient! Sur un schéma électronique comme celui-ci, l'emplacement et l'orientation des composants ne sont pas importants - tout est histoire de préférences et conventions. Il faut cependant être sûr que les interconnexions soient correctes ! Quand le schéma est terminé, il y a trois étapes à franchir afin de rendre notre dessin "intelligible" pour des machines numériques:

1. Donner une référence unique à chaque composant. Exemple: R1, R2…
2. Créer un fichier d'interconnexions: dans le jargon il est appelé ***Netlist*** File. Exemple: "dire que la broche 5v de l'Arduino est cablé sur l'alimentation 5v";
3. Créer un fichier de correspondance des symboles avec un module réel pour chaque composant. Exemple: "dire que le LM35 (en haut à gauche) est en fait un boitier TO92".

La première étape est très simple: il suffit de cliquer sur une icône (la 22ème en haut) et laisser le logiciel faire son oeuvre. La deuxième

— le Netlist - est aussi simple: c'est la 21ème icône! À partir du fichier ainsi créé, on peut aborder la troisième étape, la correspondance des symboles avec leurs vrais composants. C'est un petit outil qui s'appelle **CVpcb**, lancé par la 14ème icône, qui nous aide dans cette tâche.

FIGURE II.6.9. – Correspondance symbole <-> composant

Ce logiciel nous présente, à gauche, la liste qui a été créée à partir des étapes 1 et 2 et nous propose une liste de modules à partir des bibliothèques standards. De plus, il propose également des bibliothèques additionnelles. L'exemple ici c'est le module Arduino Shield: on voit bien la silhouette bien connue de l'Arduino UNO, avec les pastilles de connexion qui correspondent aux broches réelles. Ce n'est pas évident sur cette image, mais le module est modélisé aux dimensions exactes. Les correspondances sont ainsi créées en quelques clics dans ce logiciel. À la fin, comme toujours, il faut sauvegarder notre travail, dans un fichier de composants.

II.6.5.1.5. La platine

Maintenant, nous sommes prêts pour dessiner notre platine! Voilà un autre outil: PCBNew (lancé par un clic sur la 15ème icône).
Ce logiciel nous propose un canevas vide… Il faut lui indiquer le nom de notre fichier NetList et puis PAF ! - nous avons ce qu'on appelle en Anglais un rats-nest (nid de rats):

FIGURE II.6.10. – Le rats-nest

Le logiciel connait maintenant la forme, les branchements et les interconnexions des composants - mais il ne sait ni où ni dans quelle orientation on veut les disposer. Ils sont laissés en tas et c'est à nous de diriger les opérations. C'est un jeu de cliquer-tirer qui peut durer plusieurs heures… En tirant les composants, les interconnexions restent toujours évidentes avec les fils blancs élastiques qui suivent les mouvements. Il faut faire en sorte que le moins possible de ces fils se croisent - c'est un art - un métier même!
Au bout d'un certain temps on peut arriver à quelque chose comme ceci (eh oui, il en avait du monde dans le tas !):

FIGURE II.6.11. – Les composants en position

(Je dois avouer que j'ai supprimé l'affichage de la plupart des liaisons "fils blanc", car on ne voyait vraiment rien…)
Maintenant, il faut faire les pistes en dur. Pour une platine "fait maison", on essaie de le faire uniquement sur la face arrière: un vrai défi! Après encore des jeux avec la souris, il est possible d'arriver à un résultat comme celui-ci:

FIGURE II.6.12. – Un routage qui tient (presque) sur une seule face

Tout ce qui est en rouge, c'est le cuivre sur la face arrière de la platine, vu par "transparence". Les quelques traces de couleur verte sont sur la face avant: pour notre platine "fait maison", elles seront matérialisées par des liaisons par des fils (appelés *straps* ou *jumpers* en anglais). Notez le curseur en forme de croix en haut à gauche du dessin: c'est un élément d'une importance fondamentale: c'est le point d'origine, le repère "zéro" pour calculer la position des trous pour les composants. Vous verrez bientôt pourquoi il est en haut, et non en bas (réfléchissez bien: vous trouverez la réponse vous-même).
Tout ce que nous avons vu jusqu'à présent n'est pas vraiment nouveau, c'est la façon traditionnelle de fabriquer des platines. On prend la partie de l'image en rouge, on imprime (en noir) un calque et l'on fabrique la platine par photo-gravure avec plein de produits chimiques plus ou moins nocifs pour nous et pour l'environnement…

II.6.5.1.6. Fabrication numérique

Eh oui, et c'est même le titre de ce MOOC! Avec une fraiseuse numérique et les bons outils, il est possible de préparer la même platine sans avoir à sortir les gants et lunettes de protection. Il y a deux processus à gérer: 1) la "gravure" des traces; 2) le perçage des trous. Dans l'industrie, il existe des normes pour créer des fichiers de commande pour ces étapes: pour la partie gravure c'est un fichier GERBER; pour les trous c'est un fichier EXCELLON. Les deux fichiers sont générés par KICAD à partir du logiciel PCBNew. Regardons un instant les paramètres pour la génération du fichier EXCELLON:

FIGURE II.6.13. – Le menu de gravure

(Désolé: je travaille avec KiCad en mode anglais…)
Notez deux choses en bas de ce menu: le "Drill origin" (référence zéro), et l'option "Mirror y axis" (inverser l'axe "Y"). La première, c'est pour que les coordonnées dans le fichier soient référencées à partir du curseur que nous avons vu sur l'image. Et la deuxième, primordiale, veut dire qu'il faut 'retourner' l'image… Car notre platine, pour la faire usiner par une fraiseuse, doit montrer sa face cachée: on va usiner le DOS de la platine et non le devant. Posée ainsi à

l'envers, les traces et les positions des trous seront inversées par rapport à l'image donnée par PCBNew.
OK, nous avons nos fichiers, passons aux choses sérieuses…

II.6.5.1.7. GCODE

Non, désolé, on n'est pas encore en train de faire des copeaux… Quelques petites étapes sont encore nécessaires avant de démarrer la fraiseuse. Nous avons vu dans le cours que les imprimantes 3D et les fraiseuses (au moins, les modèles que nous trouvons chez nous ou dans les Fablabs/Makerspaces, etc.) fonctionnent à partir des fichiers en GCODE. Il faut traduire nos fichiers industriels en GCODE. Roulement de tambours: entre en scène un autre logiciel libre, open et, surtout, génial: FlatCam.
Flatcam nous permet de prendre des fichiers industriels et de les convertir en GCODE pour nos machines à nous. En passant, FlatCam nous permet de faire plein d'autres choses: corriger les dimensions; faire inverser "l'image" si le logiciel de création n'en est pas capable; déplacer l'origine; ajouter des commandes ou paramètres spécifiques pour la machine. Voici un aperçu de FlatCam.

FIGURE II.6.14. – FlatCam au travail

II.6.5.1.8. Gravure à l'anglaise

Et non, ce n'est pas (uniquement) parce que votre humble animateur est de descendance britannique: la technique que nous allons utiliser pour créer les pistes s'appelle bien ainsi! L'idée est de demander à la fraiseuse numérique de nous 'découper' les traces en usinant les contours.
Avec une petite fraise en forme de "V", il est possible de découper autour de chaque piste pour l'isoler des voisins. Pour générer le fichier GCODE correspondant, il faut maîtriser un certain nombre de paramètres spécifiques à la machine et la matière (vitesse de coupe, profondeur, largeur de recouvrement, etc.) Mais déjà - comme dans le film Apollo 13 - "Bip, Houston, nous avons un problème… bip".
Regarder l'image des traces dans la fenêtre de droite: 10 points pour la première personne qui me dit ce qui cloche… Non, ce n'est pas le fait que l'image soit double car ça, c'est volontaire: ma platine de télémètre a été conçue pour que je puisse fabriquer deux platines à partir d'une plaque cuivrée standard de 160x100mm. Non, il y a un autre problème. Réfléchir… Mais oui! Ce n'est pas dans le bon sens! Il faut inverser l'image pour que, une fois usinée et retournée, notre face cuivrée devienne le DOS de notre shield. Un clic sur le menu 'Tools' de FlatCam et nous avons une option 'Outil pour platines double-face'. Cet outil nous permet de faire l'inversion de l'image, et nous donne le choix de faire l'inversion par rapport à l'axe "X" ou l'axe "Y".
Voici le menu, et le résultat:

FIGURE II.6.15. – Le menu de Flatcam

FIGURE II.6.16. – La prévision de gravure

L'axe de "miroir" c'est l'axe "X", et l'origine, ou point zéro, c'est le point 0,0. Maintenant, l'image est dans le bon sens pour l'usinage et le point 0,0 - en bas à gauche - c'est aussi le même point où se trouvait notre curseur de référence pour le fichier de perçage! Les traces rouges indiquent le parcours de l'outil pour découper les traces. Quelques clics supplémentaires et nous avons un fichier tout chaud en GCODE pour donner à manger à notre fraiseuse.
Et voici ce que notre fraiseuse peut faire avec:

ÉLÉMENT EXTERNE (VIDEO) —
Consultez cet élément à l'adresse https://geo.dailymotion.com/player.html?video=x2h7bvr&.

II.6.5.1.9. Des p'tit trous

Dernière ligne droite! Les traces sont découpées, il ne reste que les trous à percer. Avec FlatCam le procédé est pareil: on ouvre le fichier EXCELLON et nous avons plein de choix à faire... Est-ce que nous allons percer tous les trous d'un coup? (car certains trous sont plus grands que d'autres) À quelle vitesse? À quelle profondeur? etc. Notre JO ou un copain qui a déjà fait le tour de la question peut nous renseigner, et FlatCam nous montre une image comme celle-ci:

FIGURE II.6.17. – Itinéraire de perçage

Ici, j'ai fait le choix de faire TOUS les trous avec un foret (ou mèche si vous préférez) de 0,8mm de diamètre. Même si j'ai quelques trous à élargir (pour les vis, pour certaines broches qui

font plutôt 1mm, etc.), le fait d'avoir déjà un avant-trou me permet de rapidement faire les bons trous, aux bon endroits, à la main. Un sacré gain de temps. Quelques clics, un fichier GCODE, et à nouveau la fraiseuse partie pour un tour !
Voici le perçage…

Élément externe (Video) —
Consultez cet élément à l'adresse https://geo.dailymotion.com/player.html?video=x2h7c5o&.

… et le produit fini

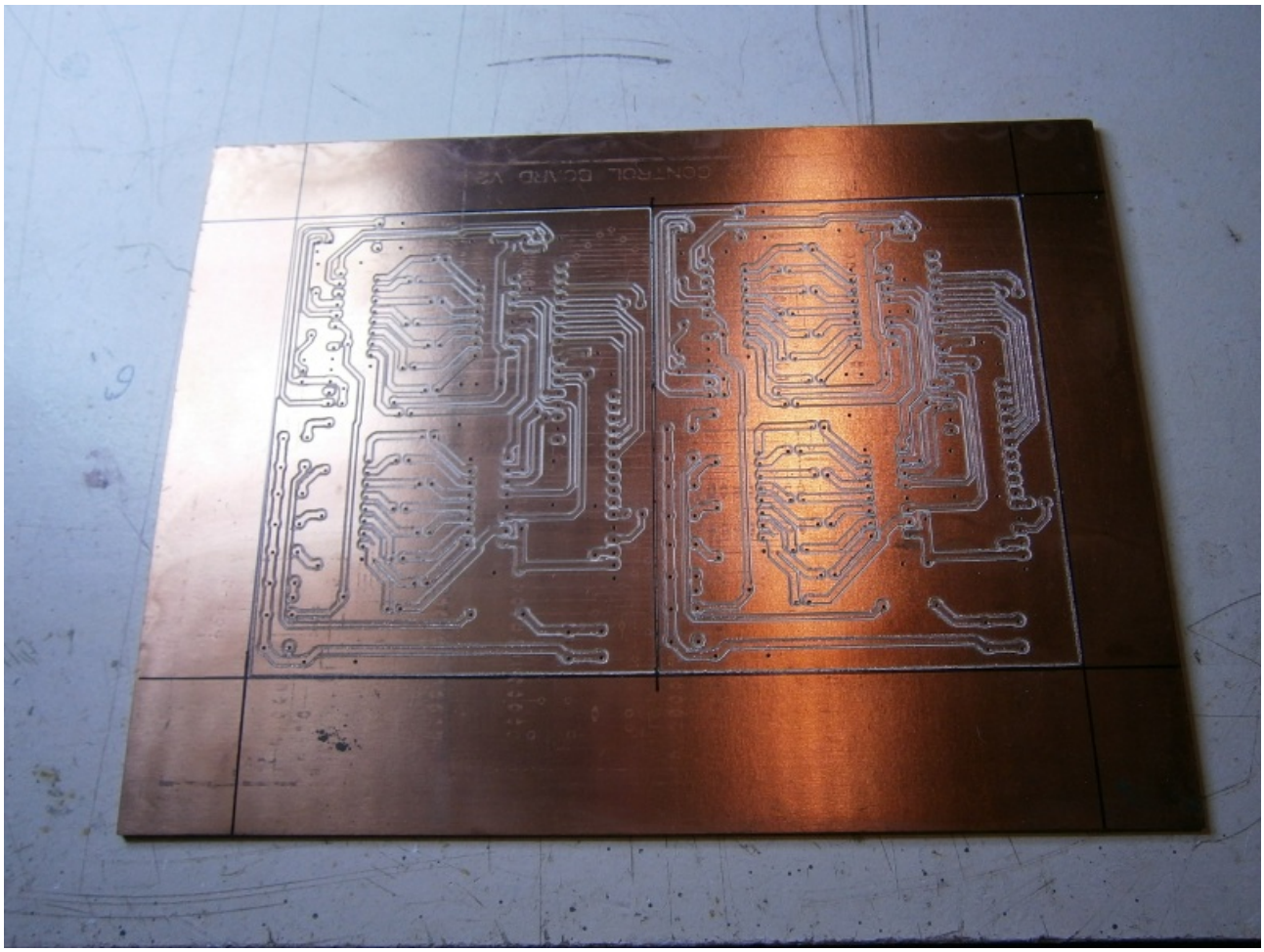


Figure II.6.18. – Le shield, gravé, percé, prêt à être assemble!

À noter: entre l'étape de "gravure" et l'étape de perçage, il ne faut pas éteindre ou dérégler la fraiseuse: sinon on perd la référence du point "zéro" et les trous seront décalés ! Là, c'est l'expérience qui vous parle - regardez bien l'image!

Glenn Smith février 2015

Contenu masqué

Contenu masqué n°7

II.6.5.1.9.1. Code Arduino

```
/*
  Feu Bicolore et Processing

 TP de la semaine 10 du MOOC "La Fabrication Numérique"

 Le montage :
 * Un bouton poussoir branché sur la broche 2

 créé le 20 Mai 2014
 par Baptiste Gaultier

 Ce code est en CC0 1.0 Universal

 https://www.france-universite-numerique-mooc.fr/courses/MinesTelecom/04002/Trimestre_1_2014/about

 */

const int boutonPin = 2;  // C'est l'entrée INT0

//  Pour qu'on puisse changer la valeur d'une variable depuis une fonction de
// gestion d'interruption, il faut que la variable soit déclarée "volatile"
volatile boolean changed = false;

// Notre fonction de traitement d'interruption.  Le strict minimum, souvenez-vous.
void doContact() {
  changed = true;
  // Difficile de faire + court
}

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

  // Broche en entrée avec résistance de pull-up
  pinMode(boutonPin, INPUT_PULLUP);

  // Attacher notre fonction ISR, détection de flanc descendant (5v vers 0v)
  attachInterrupt(0, doContact, FALLING);
}

```

```
40 void loop(){
41   if ( changed ) {
42     changed = false;
43     Serial.println("bouton");
44   }
45   delay(2000);
46 }
```

II.6.5.1.9.2. Code Processing

```
1  import processing.serial.*;
2
3  // déclaration des variables
4  PFont f;
5  PImage red;
6  PImage green;
7
8  // Port série
9  Serial myPort;
10
11 // Stocke le nombre de voiture
12 int cars = 0;
13
14 final int DISPLAY_DURATION = 2000; // 2000 ms = 2 secondes
15 int startTime; // stocke le temps de la réception d'un texte sur
       le port série
16
17 boolean buttonPressed = false;
18
19 void setup() {
20   // Taille fenêtre
21   size(1024, 768);
22
23   // Choisir une police de caractère LCD
24   f = createFont("LCDDot TR Regular", 24);
25   textFont(f, 50);
26
27   // Charger les images
28   red = loadImage("red.png");
29   green = loadImage("green.png");
30
31   imageMode(CENTER);
32
33   // initialisation du port série pour communiquer avec Arduino
34   myPort = new Serial(this, Serial.list()[0], 9600);
35   myPort.bufferUntil('\n');
36 }
37
```

```
38 void draw() {
39   // récupérer la valeur envoyée par Arduino
40   String inString = myPort.readStringUntil('\n');
41
42   // si on reçoit qq chose, alors on lance la fonction :
43   if (inString != null)
44     ArduinoButtonPressed();
45
46   if (buttonPressed)
47   {
48     image(green, width/2, height/2);
49
50     if (millis() - startTime > DISPLAY_DURATION)
51       buttonPressed = false;
52   }
53   else
54     image(red, width/2, height/2);
55
56   text("Voiture numero :", 510, 310);
57   text(cars, 510, 340);
58
59 }
60
61 void ArduinoButtonPressed()
62 {
63   // le bouton a été appuyé
64   buttonPressed = true;
65
66   // incrémenter le compteur de voiture
67   cars++;
68   println("Voiture numero : " + cars);
69
70   // vous souvenir de quand le bouton est appuyé
71   startTime = millis();
72 }
```

Retourner au texte.

Troisième partie

Internet of Things

III.1. Semaine 12 : Internet des Objets (1/2)

Introduction

Cette semaine, nous nous intéressons au futur d'internet. Rien que ça!
Après les ordinateurs, les smartphones, les tablettes... Internet s'infiltre dans nos objets: maison connectée, vêtements connectés, voiture connectée, lunettes connectées et ce n'est que le début.
Avec ce module, nous souhaitons proposer une initiation aux technologies de ce domaine en plein explosion avec des cours, des exercices et des travaux pratiques qui serviront de base à vos futurs montages. Pour vous présenter tout cela, c'est le FabLab de Lannion qui a réalisé les vidéos de cette semaine.
Aussi, Glenn a préparé un cours complet pour les curieux qui voudraient se passer d'Arduino pour leurs montages.
On vous laisse découvrir tout ça car cette semaine va être dense!
Bonne semaine!

III.1.1. Arduino sans l'Arduino

III.1.1.0.1. Libérer l'Arduino sans perdre notre oeuvre

Ce qui est bien avec l'Arduino, c'est qu'il est très facile d'essayer un concept, tester une idée et même créer un prototype de projet. Jumelé avec la platine d'expérimentation et soutenu par les bibliothèques officielles et contribuées même l'UNO devient une base de développement assez puissante pour un grand nombre de projets. Le souci - surtout si, comme moi, vous avez beaucoup d'idées à tester - c'est qu'il faut démonter chaque expérimentation avant d'en commencer une autre. Cela me rappelle mon enfance et les heures passées à jouer avec le mécano: je n'appréciais pas du tout de démonter la dernière création afin d'avoir des pièces disponibles pour la suivante. Parfois on a tout simplement envie de garder notre création - surtout quand c'est quelque chose de ludique et/ou d'utile.
Nous allons regarder ensemble les étapes, et les outils, nécessaires pour créer un montage permanent à partir de notre prototype fonctionnel construit avec l'Arduino. Je ne parlerai que de la version **ATMEGA328**, le coeur de l'Arduino UNO.
Ce module d'instruction va être encore plus dense que d'habitude: j'anime un cours similaire (plateformes PIC et Arduino) sur plusieurs jours! Donc on s'accroche et en avant!

III.1.1.0.1.1. Point de départ Le point de départ est un microcontrôleur **ATMEGA328.** Pour les montages "Do It Ourselves" (fait nous-mêmes), la version P-DIP est la plus pratique. Il s'agit d'une puce avec 28 pattes. On peut l'insérer (délicatement car les pattes sont fragiles) directement sur une platine d'expérimentation. Comme ici:

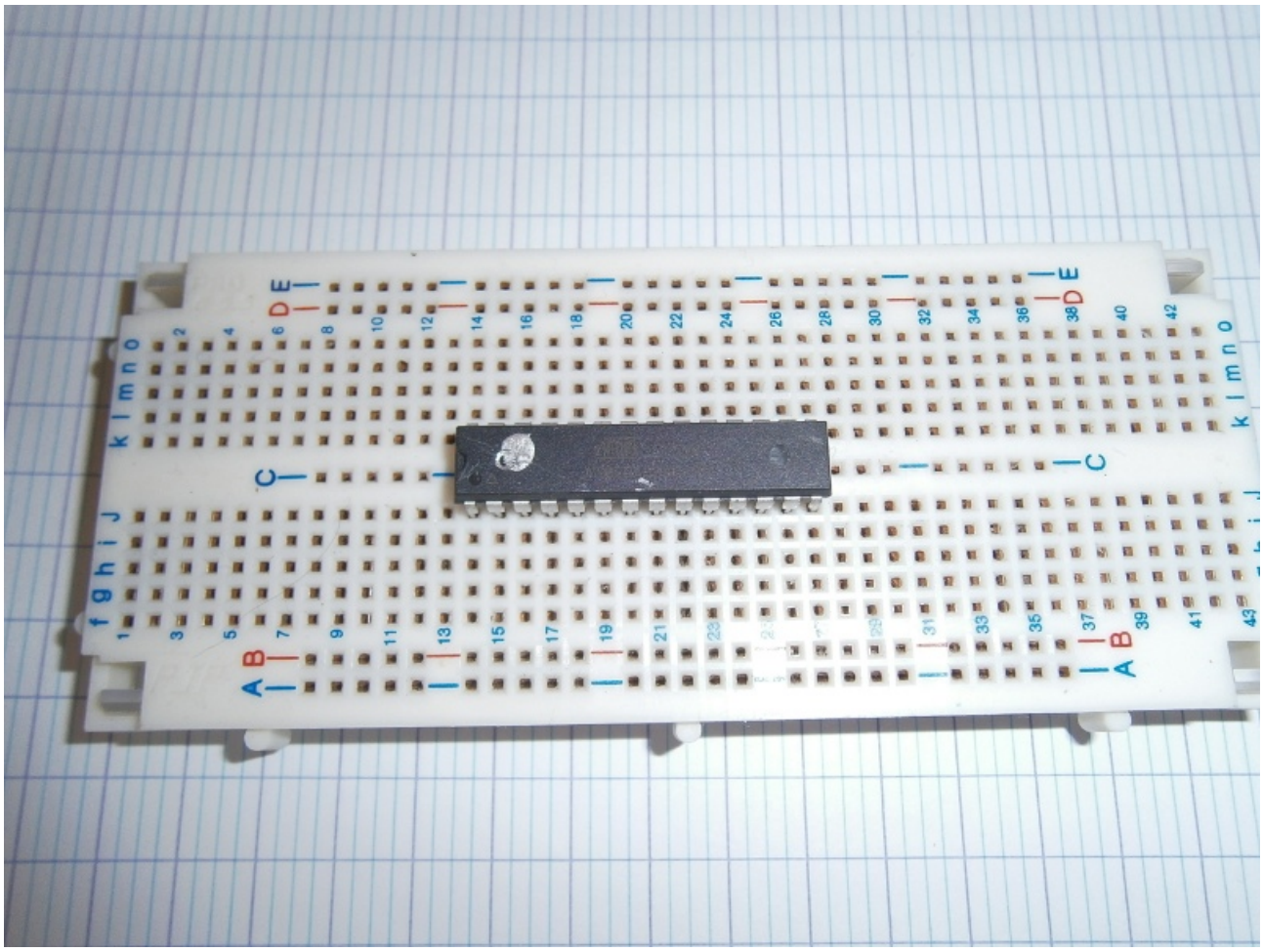


FIGURE III.1.1. – L'Atmega328 seule sur une breadboard

Il est possible de faire fonctionner ce microcontrôleur sans ajouter d'autres composants au tour, mais pour un maximum de compatibilité avec l'environnement Arduino UNO il est conseillé d'ajouter un quartz 16MHz et ses deux condensateurs. Mais je m'avance déjà trop… Car notre microcontrôleur est livré complètement vide. Comme un enfant nouveau-né il ne sait ni parler, ni comprendre ce qu'on lui dit. Pour être en mesure de téléverser un Sketch Arduino, cette puce a besoin d'une séance de rattrapage - un peu comme dans matrix - pour qu'il comprenne le langage Arduino. Dans le jargon parfois un peu barbare de l'informatique on parle de ***flashage***. L'idée c'est d'installer un petit logiciel permanent dans le microcontrôleur, lui permettant de dialoguer avec un autre ordinateur via sa liaison série. Il existe des appareils spécifiques pour cette tâche (on parle de ICSP ou *In-Circuit-Serial-Programming*) mais notre Arduino polyvalent peut aussi nous servir de 'flasheur'. Voici donc la première étape de notre parcours: préparer le microcontrôleur.

III.1.1.0.2. Flashage : Avec ou sans quartz ?

C'est une question importante qu'il faut se poser tout de suite car le logiciel que nous allons télécharger est différent pour les deux cas. Sans le quartz, il est possible de faire fonctionner ce microcontrôleur avec son horloge interne qui est réglée à l'usine pour tourner à 8MHz. L'Arduino UNO est équipé d'un quartz de 16MHz, donc deux fois plus rapide. Mais le quartz n'est pas gratuit: non seulement il a un prix, mais aussi il prend de la place et, surtout, deux lignes d'entrée/sortie. Regardons l'image suivante:

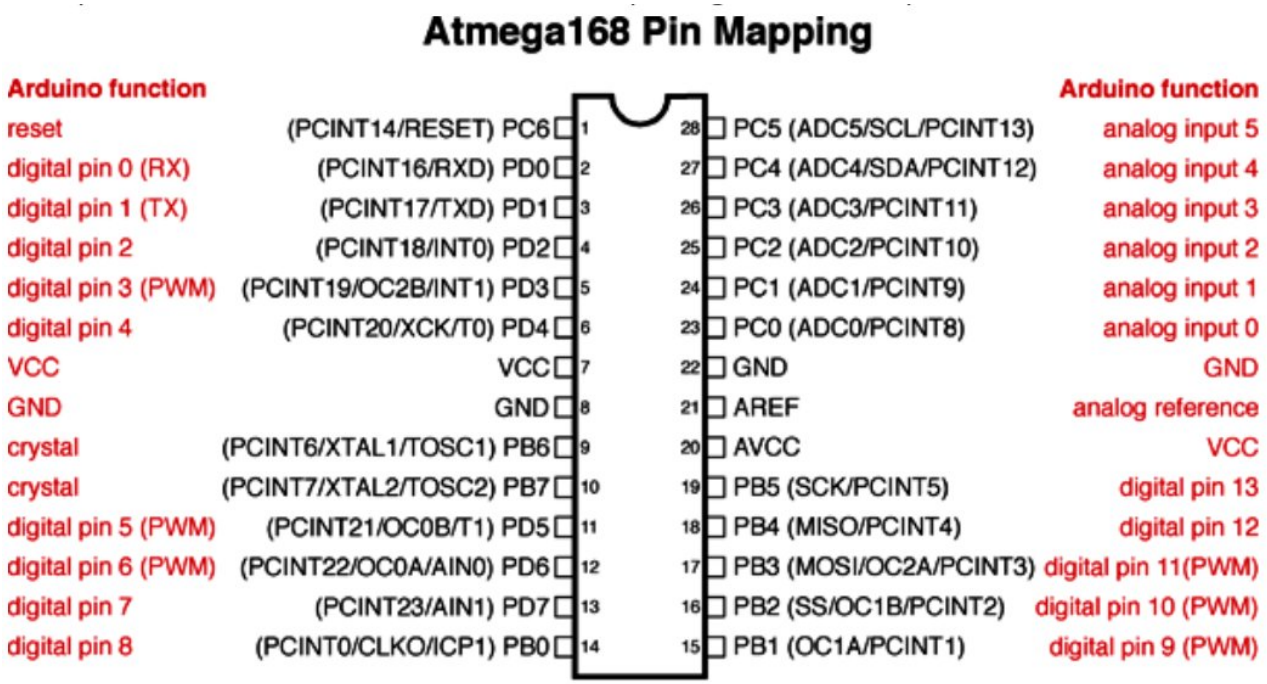


FIGURE III.1.2. – Definition des broches de l'Atmega328

Cette image, qu'il faut garder précieusement à coté de vous pendant ce projet, nous montre les branchements de l'ATMEGA328 et les équivalences Arduino UNO. Notez que l'ATMEGA168 et l'ATMEGA328 partagent la même configuration d'entrées et sorties. Si on regarde les pattes 9 et 10 on voit que, sur l'Arduino, elles sont connectées sur le quartz (crystal). Mais ce sont des broches d'entrée sortie comme les autres. Si nous n'avons pas besoin de faire fonctionner le microcontrôleur à 16MHz (et c'est rarement le cas), nous pouvons "récupérer" ces deux broches pour autre chose. Dès que nous avons pris notre décision, préparons le bloc opératoire pour le flashage.

III.1.1.0.2.1. Version 16MHz (avec quartz) Il faut faire un montage comme ceci[1]:

FIGURE III.1.3. – Branchement du microcontrôleur

Les branchements au microcontrôleur sont des plus simples: l'alimentation en +5v et 0v (les deux côtés, notez bien), le quartz avec ses deux condensateurs, et une résistance de "pull-up" pour une fonction de remise à zéro (reset) pilotée. La liaison de communication est une liaison série similaire à l'interface I²C que nous avons étudiée la semaine dernière. L'idée de ce montage est d'utiliser l'Arduino comme programmateur. Voici comment procéder:

1. Quand tout est bien câblé, ouvrir l'environnement Arduino et chercher un Sketch qui s'appelle "***ArduinoISP***" sous **Exemples**.
2. Téléverser ce Sketch dans l'Arduino.
3. Allez dans le menu **Outils** (Tools) et chercher l'option **Programmateur**, puis sélectionner "***Arduino as ISP***".
4. Toujours dans le menu **Outils**, cliquer sur l'option "***Graver la séquence d'initialisation***". Les LEDs Rx et TX de l'Arduino vont clignoter pendant plusieurs secondes puis, si tout va bien, le message "*Gravure de la séquence d'initialisation terminée*" sera affiché.

5. Le microcontrôleur est prêt pour le service. Notez bien qu'il ne fonctionnera QUE si le quartz et les deux condensateurs sont branchés. Vous pouvez maintenant procéder à l'étape *Premier essai*.

III.1.1.0.2.2. Version 8MHz (sans quartz) Avant de vous montrer le câblage pour le programmateur, il faut que je vous explique quelques détails. Pour la version 16MHz, dès que le microcontrôleur est 'flashé' il se comporte *exactement* comme un Arduino UNO - car les paramètres de fonctionnement sont les mêmes. Pour la version 8MHz, par contre, certains paramètres devraient changer. La configuration de l'horloge interne et l'attribution des broches 9 et 10, pour commencer. Non seulement le code d'initialisation est différent, mais pour l'environnement de développement cette bestiole n'est plus un Arduino UNO - et donc la compilation et le téléversement des Sketchs sont aussi modifiés. Heureusement, l'environnement est relativement facile à personnaliser. Nous avons besoin de deux nouveaux fichiers, à installer une fois pour toute, pour que notre 'mutant' soit accepté parmi les plateformes Arduino. Le premier fichier contient les paramètres pour les étapes de flashage, et de téléversement. Le deuxième fichier contient les définitions pour le microcontrôleur sans quartz et avec les deux broches d'entrée/sortie supplémentaires. Voici comment les installer:

— Téléchargez ce fichier Arduino_breadboard.zip , et le décompressez dans un dossier nommé "hardware" dans votre dossier principal "sketchbook". Voici nos deux fichiers. Le fichier *boards.txt* va rester là où il se trouve, l'autre fichier est à installer dans l'environnement Arduino.
— Cherchez le dossier principal de votre installation Arduino. Vous aurez besoin des droits d'administrateur car ce dossier est probablement sous /usr/share, par exemple, sous linux. Sous windows, il me semble (car je n'ai pas une version windows chez moi) que c'est sous \Program Files…
— Descendez l'arborescence de ce dossier jusqu'à *…/hardware/arduino/variants*. Dans le dossier **variants** il faut créer un nouveau dossier "***breadboard***" (sans majuscules).
— Déplacer le fichier *pins_arduino.h* que vous avez décompressé tout à l'heure dans ce nouveau dossier *breadboard*. (sous linux : vérifier les droits de lecture…)
— Ça-y-est! Il ne reste qu'à fermer et à redémarrer le logiciel Arduino.

Pour vérifier que les modifications ont étés prises en compte, regarder dans le menu **Outils / Type de carte**: à la fin de la liste vous devriez voir "*ATMEGA328 on a breadboard (8MHz…*"
Voici le câblage pour programmer notre microcontrôleur 8MHz:

FIGURE III.1.4. – Câblage pour programmer un microcontrôleur 8MHz

Ben, c'est pareil que l'autre - sauf qu'il n'y a plus le quartz! Ah si, la résistance a aussi disparu… Il vaut mieux la mettre, cette résistance, si possible. Voici la procédure de flashage:

1. Quand tout est bien câblé, ouvrir l'environnement Arduino et cherchez un Sketch qui s'appelle "***ArduinoISP***" sous **Exemples**.
2. Téléversez ce Sketch dans l'Arduino.
3. Allez dans le menu **Outils** (Tools) et chercher l'option **Programmateur**, puis sélectionnez "***Arduino as ISP***".
4. Maintenant, et c'est très important, il faut changer le type de carte: sélectionnez "**ATMEGA328 on a breadboard (8MHz…**"

5. Toujours dans le menu **Outils**, cliquez sur l'option "***Graver la séquence d'initialisation***". Les LEDs Rx et TX de l'Arduino vont clignoter pendant plusieurs secondes puis, si tout va bien, le message "*Gravure de la séquence d'initialisation terminée*" sera affiché.
6. Le microcontrôleur est prêt pour le service.

III.1.1.0.3. Premier essai

Nous voilà avec un microcontrôleur fraîchement flashé (on dirait une publicité pour dentifrice). Comment vérifier si ça marche? Il faut téléverser un programme dedans! Mais comment? La sortie de notre PC de développement c'est un câble USB. Le microcontrôleur tout nu n'a rien pour communiquer. Là encore, deux solutions sont possibles:

— Modifier notre Arduino pour que notre microcontrôleur prenne la place du 'vrai';
— Utiliser une interface USB/Série .

III.1.1.0.3.1. Téléversement avec l'Arduino Je ne vous conseille pas cette opération si l'Arduino que vous avez devant vous n'est pas le vôtre! Car on va enlever le microcontrôleur! Et j'ai déjà dit que les broches sont fragiles… Si vous êtes dans un Fablab ou autre endroit génial de ce genre, demander à votre GO (gentil organisateur) s'il n'a pas une interface série à vous prêter.
Deuxième avertissement: les composants électroniques sont TRES sensibles aux décharges électrostatiques. Ne travaillez pas avec l'électronique si vos vêtements sont en fibres synthétiques et que vous entendez des crépitements quand vous bougez! Le microcontrôleur sorti de la carte est très vulnérable: il faut l'insérer dans de la mousse conductrice (mousse noire) ou, à défaut, dans un morceau de papier alu replié plusieurs fois.
Les avertissements derrière nous, voila l'étape un peu critique: il faut enlever le ATMEGA328 de la carte Arduino. Le souci c'est qu'il faut soulever la puce sans en tordre les pattes. Procédez avec précaution, sans précipitation. Dans un Fablab, ils auront peut-être le vrai outil (extracteur), mais chez nous il faut chercher un tournevis fin, genre tournevis de joaillier. Avant d'enlever le microcontrôleur, marquez d'un point de blanc correcteur ou similaire afin qu'on ne le confonde pas avec notre microcontrôleur à nous… En commençant par le côté opposé aux connecteurs d'alimentation / USB, passez le tournevis à plat entre la puce et son support - sur toute la longueur de la puce. Soulevez un peu (1 à 2mm). Sortir le tournevis et maintenant procédez de même de l'autre coté. Continuez ainsi jusqu'à la libération de la puce. Redressez (doucement) les éventuelles pattes tordues, et placer le microcontrôleur dans la mousse conductrice.
Deux possibilités se présentent à nous: on peut câbler l'interface entre l'Arduino désormais "brain dead" et notre platine d'expérimentation, ou on peut insérer notre microcontrôleur sur l'Arduino à la place du 'vrai'. Cela vaut aussi pour la version 'mutante' à 8MHz. Sachant que si on prend la 2ème option, il faut refaire l'étape stressante de enlèvement de puce une deuxième fois! Voici le schéma de câblage pour l'option "platine":

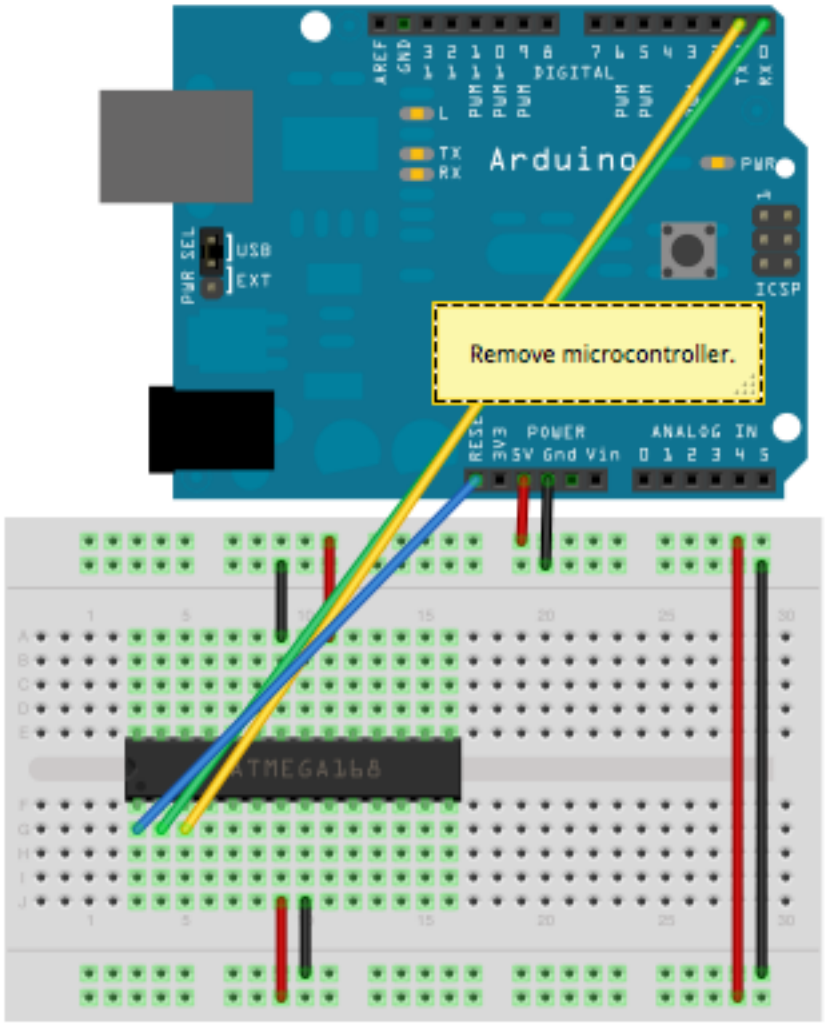


FIGURE III.1.5. – Câblage platine

C'est assez simple, et peut-être plus sûr que de risquer d'endommager les pattes de notre microcontrôleur…
Ouvrir l'environnement Arduino. Si nous travaillons avec le microcontrôleur à 8MHz, il ne faut pas oublier de changer le type de carte dans le menu Outils. Téléversez le Sketch 'Blink' (Exemples / 01.Basics / Blink). Si vous avez inséré votre microcontrôleur sur la carte Arduino vous devez voir la LED sur la broche 13 clignoter. YES! Ça marche!
Et sur la platine? Mais oui, il faut ajouter la LED (et sa résistance!) - mais sur quelle broche? Allez, la réponse est plus haut. Je vous ai même dit qu'il fallait la garder proche de vous…. Oui, l'image de "pin mapping". Sur la carte Arduino, la LED est branchée sur le connecteur 13. Cherchez "digital pin 13" sur l'image et vous verrez que cela correspond à la broche 19 du microcontrôleur[2]. Il suffit de brancher notre LED entre la broche 19 et 0v/GND. Magique! Aujourd'hui Blink, demain le monde.

III.1.1.0.3.2. Téléversement à l'aide d'une interface série Plusieurs sortes d'interfaces USB / Série existent, elles appartiennent à deux grosses catégories: les 'simples', ne comportant pas d'autres signaux que Rx et Tx, et les autres - plus évoluées. Certaines sont même pré-configurées pour fonctionner exactement comme l'interface intégrée dans l'Arduino: c'est la solution la plus simple (mais, forcement, la plus chère). Les versions 'simples' fonctionnent bien mais

pour ceux qui ont connu les Arduinos d'antan… ils vont retrouver leur jeunesse car il faut 'resetter' le microcontrôleur manuellement afin de lancer le téléversement. Les interfaces plus évoluées, équipées d'une broche DTR ou carrément une ligne 'reset', souvent appelées FTDI, nous facilitent la tâche.
Voici le câblage pour la version 'simple':

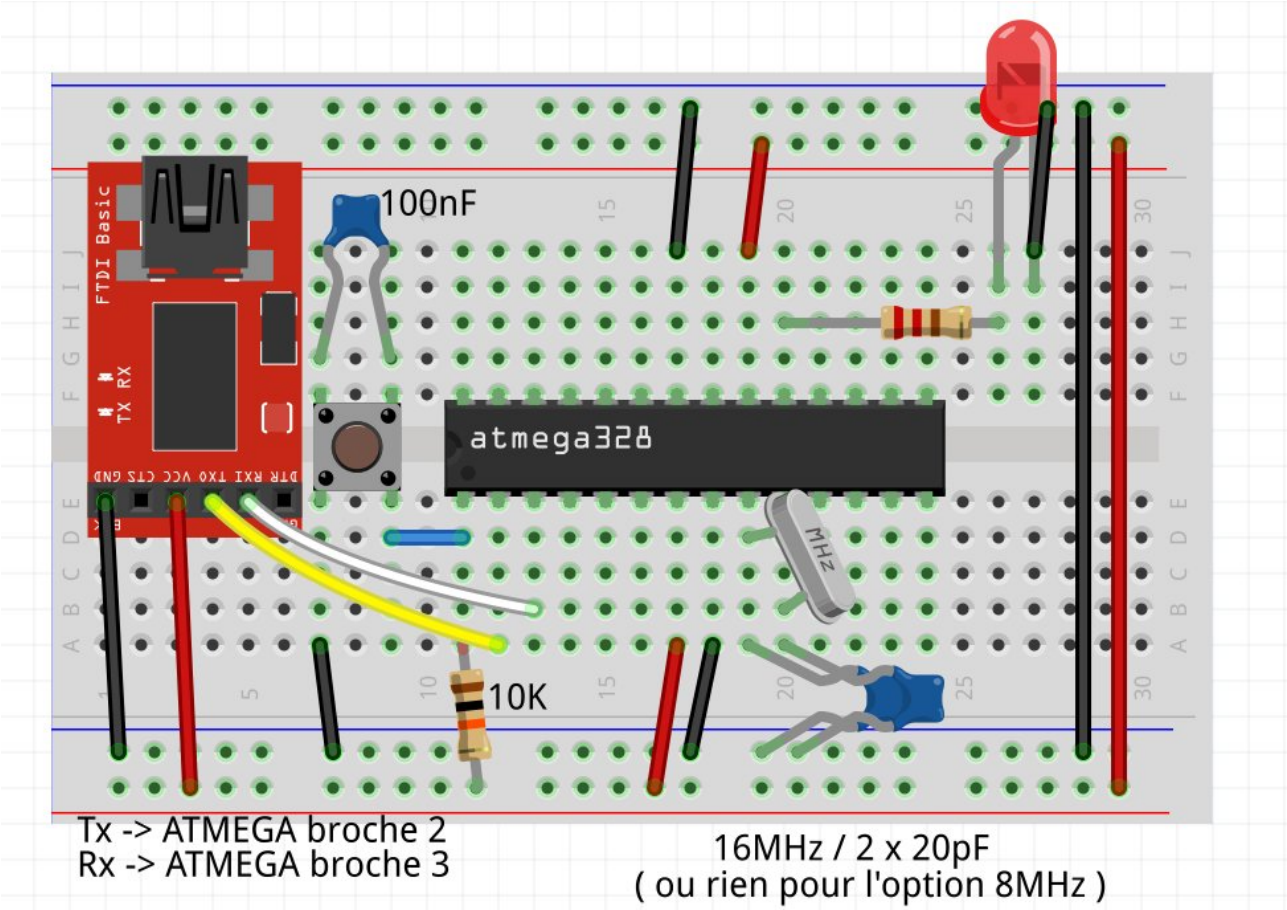


FIGURE III.1.6. – Câblage simple

Le bouton câblé sur la broche 1 est là pour faire 'reset'. C'est un coup de main à trouver, mais quand on lance le téléversement il faut avoir le doigt sur le bouton et les yeux rivés sur le programme Arduino : dès l'instant où la compilation est terminée (la taille de notre Sketch est affichée) il faut appuyer sur le bouton 'reset'. Si tout va bien le programme est chargé dans le microcontrôleur. Essayez avec le Sketch Blink.
Si vous avez une interface USB / Série évoluée, avec une sortie DTR, voici comment la câbler:

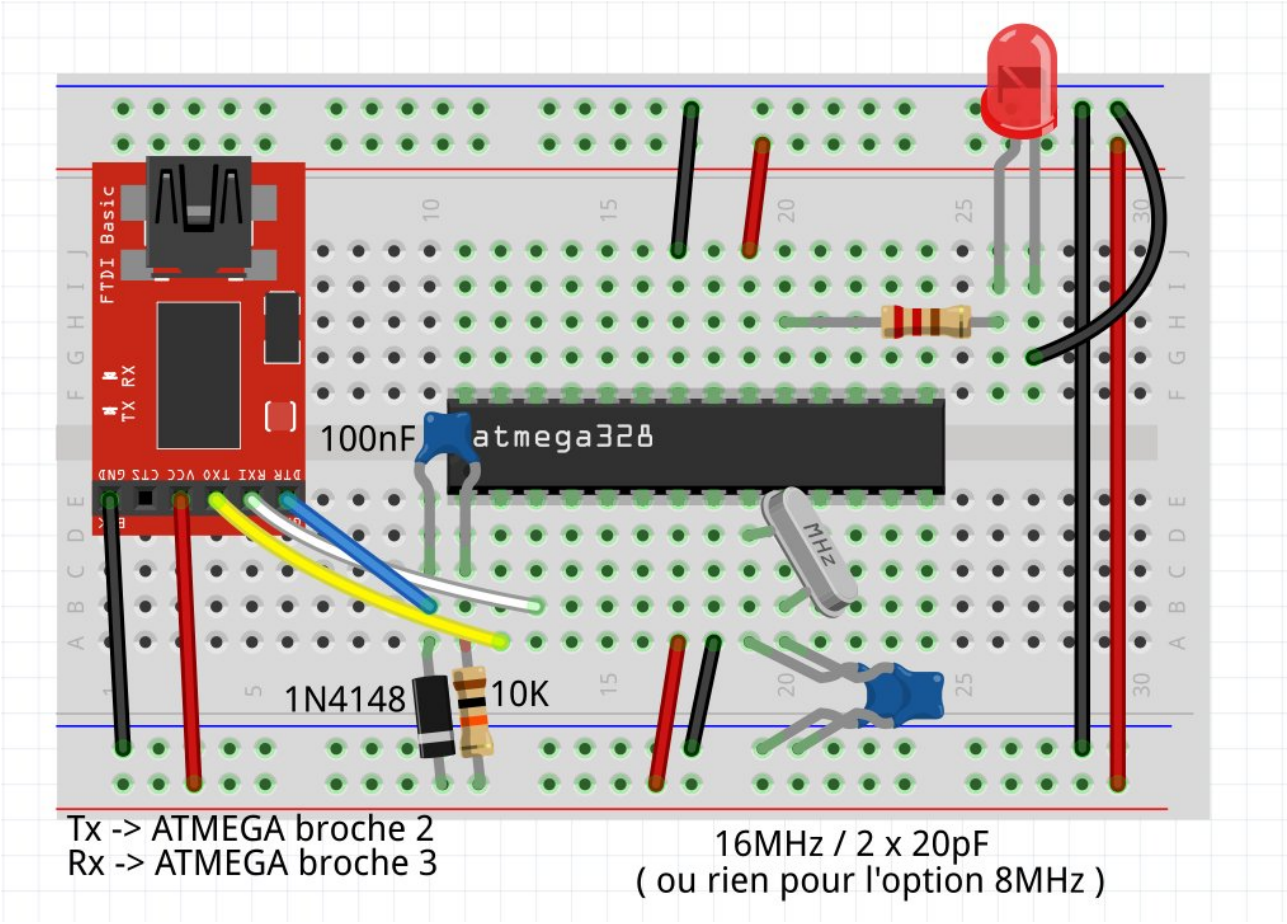


FIGURE III.1.7. – Câblage USB/Série évoluée avec sortie DTR

Les plus avertis d'entre vous remarqueront le montage condensateur/diode sur la broche 1. J'ai parlé de ce montage dans le Wiki intitulé "Diodes et condensateurs, suite et fin". C'est le circuit de détection d'un flanc descendant. La diode joue un rôle important: empêcher la tension sur la broche 1 de dépasser de trop les +5v maxi autorisés. Avec ce montage, le reset pour le téléversement est automatique, et ce montage se comporte exactement comme un vrai Arduino.

III.1.1.0.4. Alimentation

Avec l'option liaison série (avec adaptateur ou avec la carte Arduino) l'alimentation en 5v de montage est assurée. Mais si le but est de faire un appareil autonome (comme l'Arduinomètre), il faut résoudre le problème de l'alimentation. Comme toujours, nous avons plusieurs solutions. Je vais vous parler de deux solutions similaires à base de régulateurs de tension. Mais, d'abord, quelques rappels.
Le microcontrôleur a besoin d'une alimentation stable à environ 5v. Les variations de tension sont néfastes car ils perturbent le bon fonctionnement du processeur. Malheureusement 5v sont difficiles à fournir avec des piles et la tension fournie par les piles n'est pas très stable. Donc, au lieu d'essayer de faire une pile qui fournit exactement 5v, on utilise une pile d'une tension supérieure et on réduit la tension à 5v. Mais comment? Les premiers chapitres du cours d'électronique nous ont montré qu'une résistance peut servir pour faire chuter la tension. Il suffit d'avoir une intensité stable et on peut calculer la résistance nécessaire pour obtenir la chute de tension voulue. Si vous avez suivi le Wiki sur les diodes Zener vous aurez vu qu'il est difficile d'avoir une intensité stable avec un montage microcontrôleur: le processeur en lui-même

ne consomme pas beaucoup - 10mA en tout. Mais si on allume des LEDs à 20mA chacune on peut avoir une intensité qui varie du simple au quadruple dans le temps: impossible de calculer une valeur de résistance dans ces conditions-là. En plus, nous avons aussi vu que cette chute de tension à travers une résistance est un gaspillage d'énergie car ça chauffe la résistance (ce qu'on appelle l'effet Joule). C'est pourquoi le régulateur de tension est vraiment la seul possibilité pour nos montages.

III.1.1.0.4.1. Le 7805 Avec le NE555, le 7805 fait vraiment partie des vénérables ancêtres de l'électronique moderne. Même si ce régulateur existe en une multitude de formats, et que d'autres types de régulateurs existent avec de meilleures spécifications, il reste indémodable! Deux versions de ce régulateur sont intéressantes pour nous: le 7805CV en format TO220 (format transistor de puissance) qui peut gérer une intensité d'un ampère maximum; et le 78L05 en boîtier TO92 (format transistor normal) pour les intensités de 100mA maximales. Les deux versions TO92 et TO220 coûtent environ 0,50€ pièce. Le brochage:

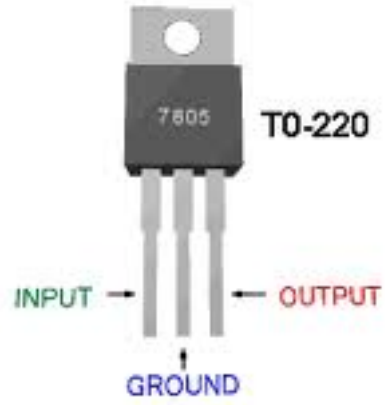


FIGURE III.1.8. – Brochage TO220

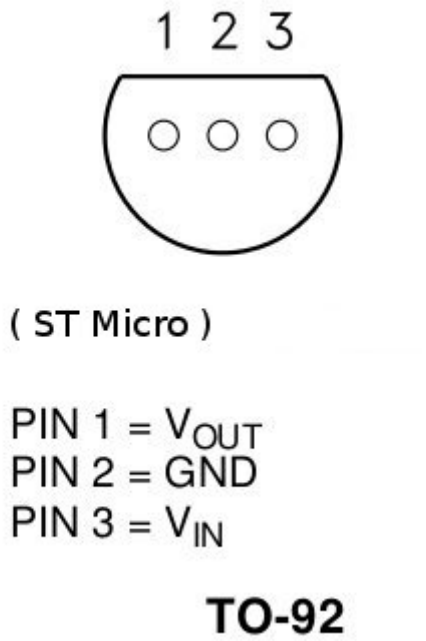


FIGURE III.1.9. – Brochage TO92

Et le schéma de câblage:

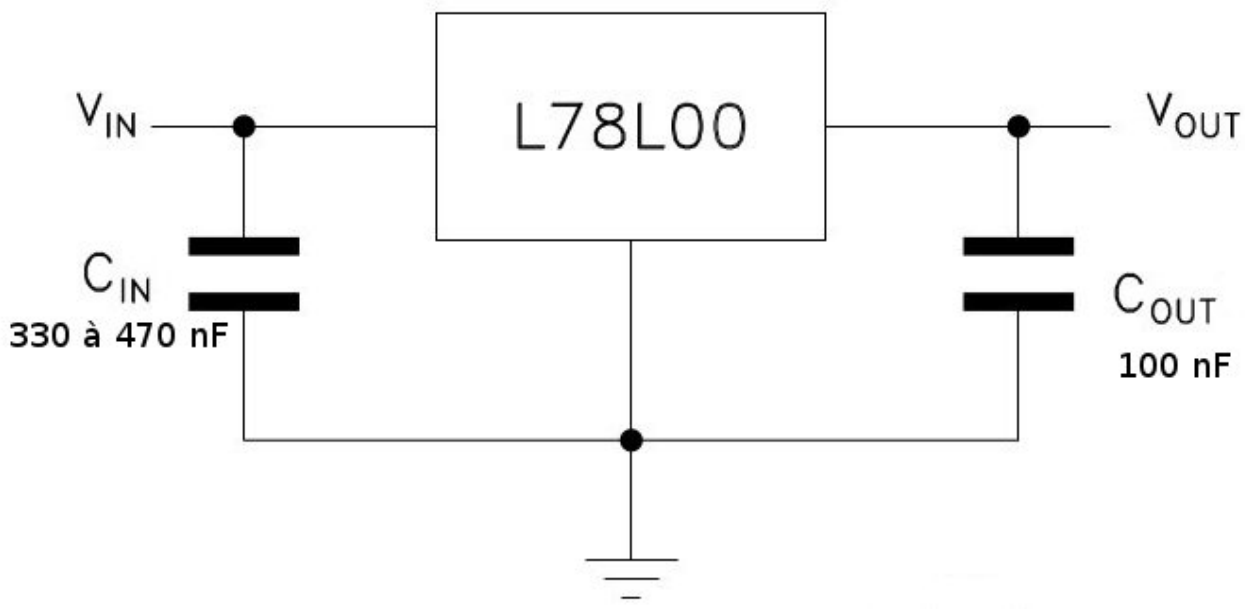


FIGURE III.1.10. – Schéma de câblage

Les deux condensateurs devraient se trouver le plus près possible du régulateur. Pour que les régulateurs 78xx fonctionnent correctement, la tension d'entrée devrait être supérieure d'au moins 2V de plus que la tension de sortie. Il faut veiller au refroidissement, surtout avec le TO220 aux intensités approchant 1A.
Il existe aussi un équivalent en version 'faible chute de tension', le LM2940, par exemple. Les versions 'faible chute de tension' se prêtent bien aux montages avec alimentation à pile. Ils ont besoin, par contre, d'un condensateur de forte valeur (22uF minimum) à leur sortie. Attention car certains de ces régulateurs n'ont pas le même brochage (pinout) que le 7805 - consulter la documentation technique!

III.1.1.0.4.2. Les platines On trouve aussi les petites platines régulateurs - d'une taille de quelques mm² - qui peuvent gérer jusqu'à 1A avec une assez faible chute de tension. Leur utilisation est identique: une entrée, une connexion 0v et une sortie régulée. Les prix varient de 1€ à plusieurs Euros.

FIGURE III.1.11. – platine régulateur

III.1.1.0.5. Et maintenant ?

Nous avons vu beaucoup de choses, et il est possible que nous ayons perdu de vue notre objectif: sauvegarder un montage ou un prototype - ou carrément le transformer en produit fini. Les étapes parcourues jusqu'ici sont les étapes préliminaires nécessaires pour la préparation d'un microcontrôleur ATMEGA328, et les montages possibles pour la suite : la programmation avec l'environnement Arduino. Pour terminer je voulais prendre un exemple concret: mon Arduinomètre avec afficheur LCD. C'est un montage encore simple, mais suffisamment complet pour nous fournir de bonnes bases pour des projets plus ambitieux. Voici le montage d'origine, avec Arduino et platine:

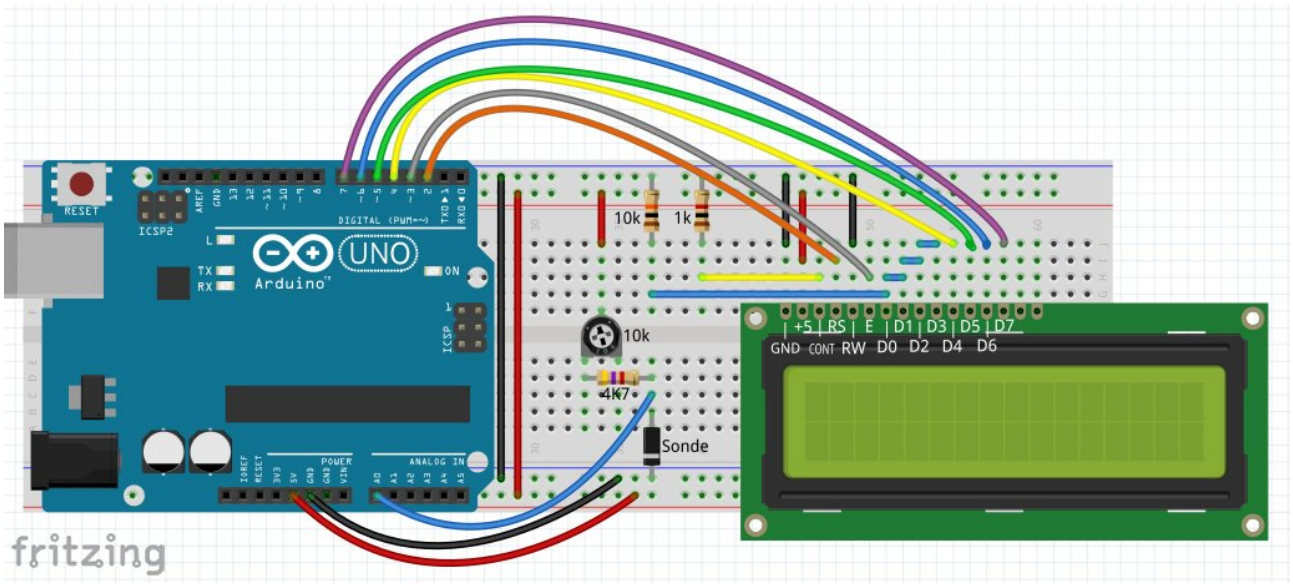


FIGURE III.1.12. – Montage d'origine

Si, armé de mon image "pin mapping" je dessine le schéma qui correspond aux branchements sur le ATMEGA328, cela donne ceci:

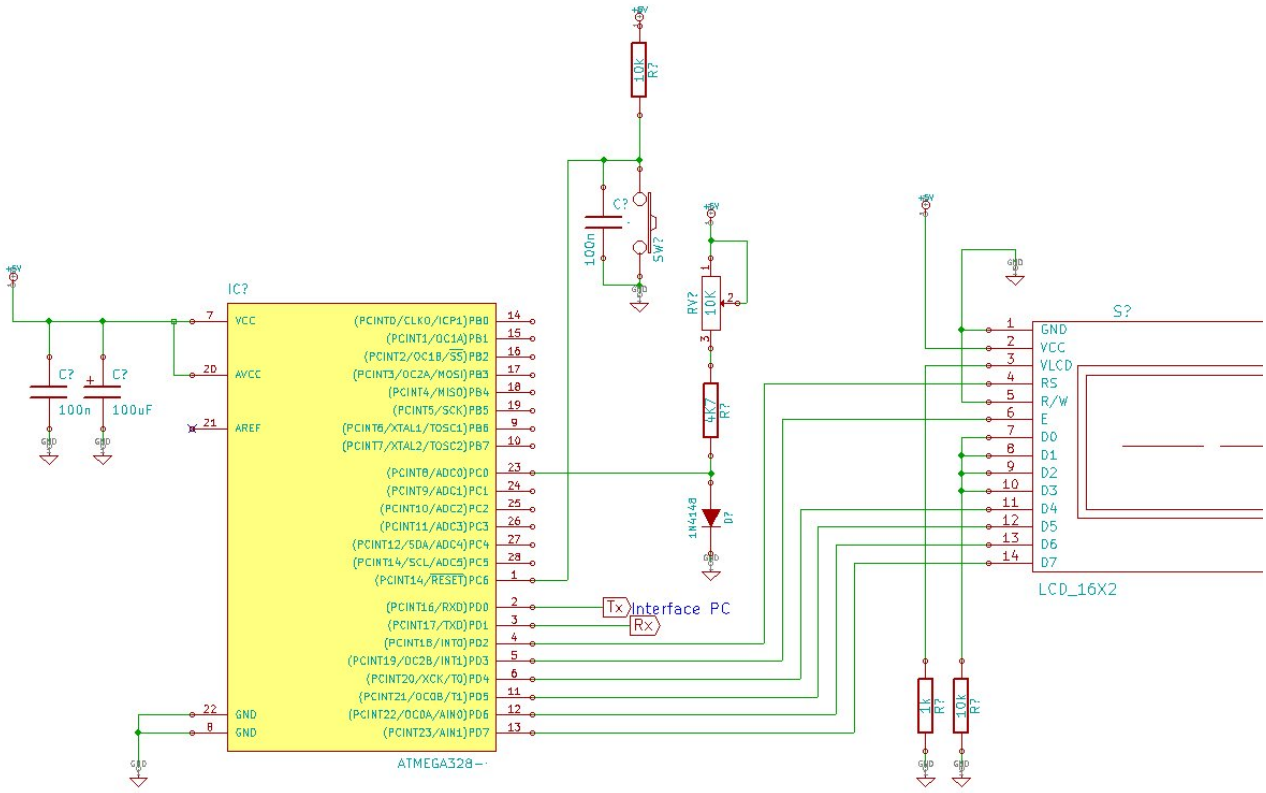


FIGURE III.1.13. – Branchement du microcontrôleur

Et le montage avec Fritzing:

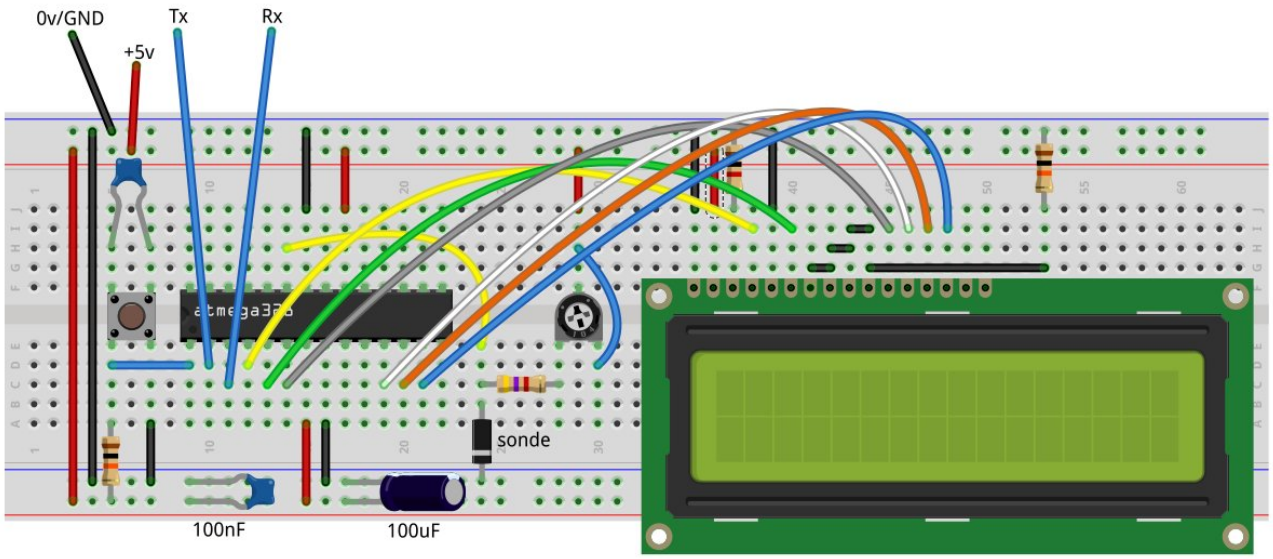


FIGURE III.1.14. – Montage du microcontrôleur

C'est pas si difficile, n'est-ce pas?
Mais maintenant je veux créer une platine permanente: un circuit imprimé. A la maison, il est possible de faire des platines simple-face de bonne qualité avec des moyens modestes. Nous n'avons pas parlé de ce sujet, mais tant qu'on a Fritzing ouvert, voici ce que cela donne:

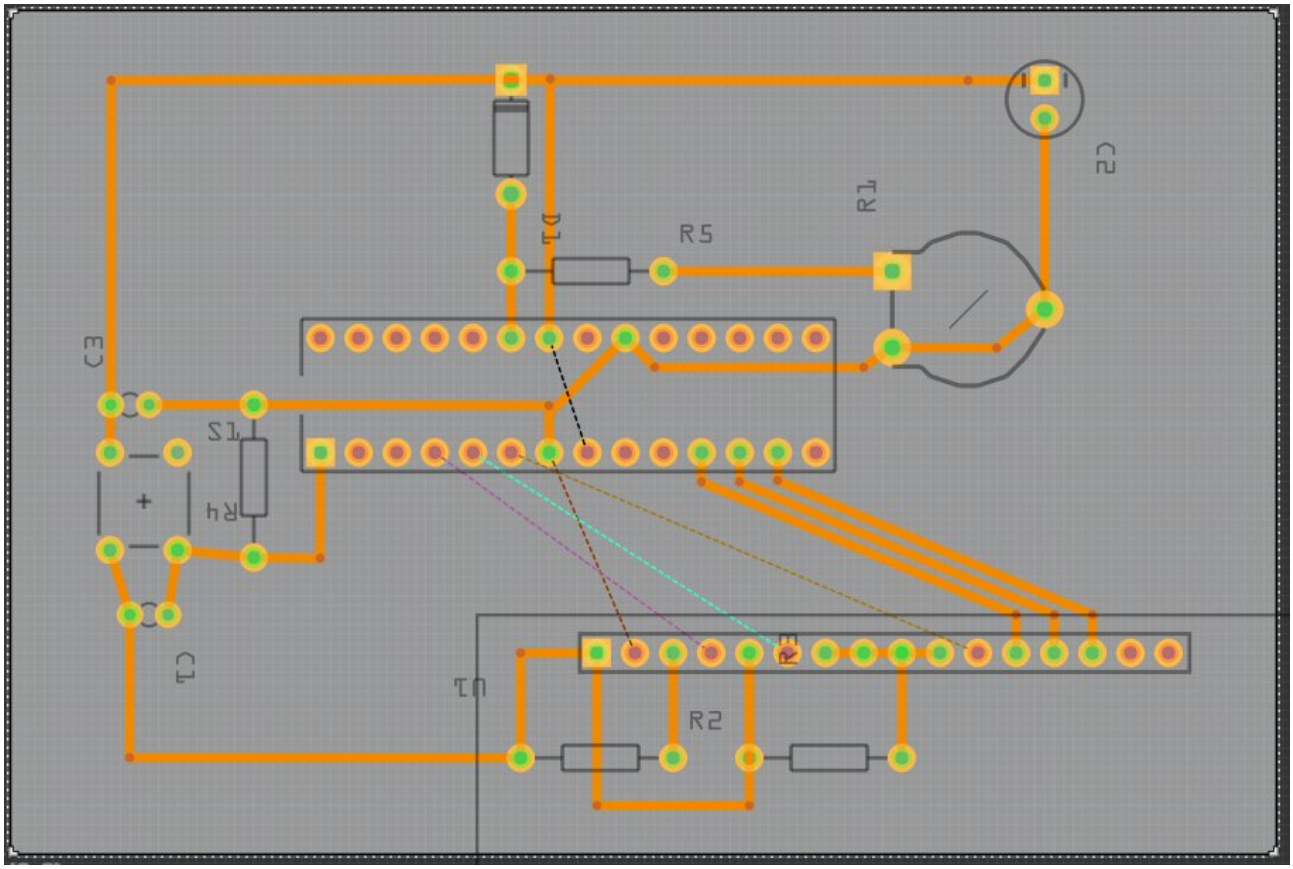


FIGURE III.1.15. – Interface de routage de Fritzing

Ici, on commence à voir le souci pour cette dernière étape. J'ai déjà placé le plupart des composants aux endroits adaptés, mais j'ai plusieurs pistes qui se croisent. (Regardez les lignes pointillées). Pour ce montage ce n'est pas très méchant et il est toujours possible de poser des 'cavaliers': des fils qui sont posés comme s'ils était des composants et qui servent à 'sauter' des pistes. Mais d'autres astuces sont aussi possibles. Il n'y a pas grand chose de branché sur notre microcontrôleur: un grand nombre de ces broches ne sont pas connectées. Si on change les définitions de brochage dans notre Sketch, il est tout à fait possible de faire en sorte que les pistes ne se croisent plus.
Je vous laisse le fichier Fritzing ici et le Sketch de l'Arduinomètre d'origine ici
— vous pouvez jouer avec les deux pour trouver une solution.

III.1.1.0.6. Bénéfices et inconvénients de la version 8MHz

Dernier petit mot au sujet de la version 8MHz. Nous avons vu que l'inconvénient majeur c'est qu'il faut ajouter des fichiers de paramètres dans l'environnement Arduino. Cela n'est pas trop contraignant, mais il faut garder les deux fichiers avec vous, sur une clef USB par exemple, si jamais vous partez en "voyage" avec votre montage - dans un Fablab, par exemple: car sans les deux fichiers installés, impossible de compiler ou téléverser des Sketchs dans le microcontrôleur 8MHz. L'autre inconvénient est que le processeur tourne plus lentement. Mais, franchement, il est quand-même rare que cette différence soit vraiment marquante car, dans la grande majorité des cas, notre programme reste dans les boucles d'attente: soit les `delay()` que nous avons codées nous-même, soit les attentes pour l'afficheur LCD, pour la liaison série, pour le I²C et j'en passe…

Et les bénéfices, alors? En voici quelques-uns
- C'est moins cher (le prix du quartz) ;
- Moins de place prise sur la platine ou sur le circuit imprimé ;
- DEUX ENTREES / SORTIES EN PLUS! La broche 9 devient "digital pin 20" (entrée ou sortie au choix) ; La broche 10 devient "digital pin 21" (entrée ou sortie au choix). [Pour les puristes il s'agit des E/S PORTB6 et PORTB7, qui n'existent pas sur l'Arduino UNO]
- Plus de flexibilité pour les montages.

III.1.1.0.7. Voilà !

Nous avons fait un grand chemin ensemble. J'espère qu'à travers les différents modules au fil des semaines, je vous ai donné le goût de vouloir expérimenter, apprendre et faire vos propres réalisations. Passez un bon été et peut-être à bientôt dans un Fablab ou une autre animation.

Glenn Smith juin 2014

III.1.2. Internet des objets

III.1.2.0.1. Internet des objets

Élément externe (Video) —
Consultez cet élément à l'adresse https://geo.dailymotion.com/player.html?video=x2kgzyn&.

Merci d'avoir regardé cette vidéo. Celle-ci constitue une brève introduction à l'internet des objets et son application avec Arduino. Nous vous invitons maintenant à réaliser votre premier montage connecté. Aussi, il est (vraiment) recommandé de lire les références proposées ici:

III.1.2.0.1.1. Références

- Documentation de la librairie Ethernet par Xavier Hinault
- Cours sur l'architecture d'Internet

Merci au FabLab Lannion pour ce cours

III.1.3. Travaux pratiques

III.1.3.0.1. TP à faire pour la semaine 13

Ça se corse cette semaine!

1. Sur la platine c'est un ATMEGA168, mais le brochage est identique au ATMEGA328.
2. Les adeptes de Fritzing auront une bonne surprise: le microcontrôleur ATMEGA328 dans la bibliothèque de Fritzing est 'codé' avec les correspondances des broches Arduino. Essayez: passez la souris sur la broche 19 et vous verrez...

À partir des fonctions vues dans la vidéo et de vos connaissances acquises lors des précédentes semaines, l'objectif de ce TP sera d'allumer ou éteindre les leds à partir d'un navigateur internet.
Vous devrez connecter votre arduino en filaire sur votre box via le shield Ethernet:

- Depuis un browser sur votre PC vous enverrez une requête http de la forme `http://192.168.1.x?led=on` ou `http://192.168.1.x?led=off` permettant respectivement d'allumer ou d'éteindre la LED
- Pré-requis : partir d'un TP précédent comme base pour l'allumage d'un LED, brancher le shield Ethernet, intégrer la librairie Ethernet (beaucoup d'infos ici).

III.1.3.0.1.1. Quelques indices Vous aurez besoin de mobiliser toutes les compétences vues ces dernières semaines pour réaliser ce TP:

- L'utilisation d'une sorties numériques
- L'utilisation de la librairie Ethernet et de l'exemple WebServer .
- N'oubliez pas de modifier les adresses MAC et IP.
- Vous pourrez avoir besoin de la librairie `String.h` et notamment des fonctions: `concat()` et `substring()`

La correction (montage et code) sera donnée la semaine prochaine!
Bon courage (car il va en falloir) et bonne semaine,
L'équipe du MOOC

III.2. Semaine 13 : Internet des Objets (2/2)

Introduction

Toutes les bonnes choses ont une fin... Et oui, après 13 semaines intenses: l'aventure "La Fabrication Numérique" touche à sa fin. On ne se quitte pas encore puisque nous vous avons encore réservé du contenu sur l'Internet des Objets cette semaine. Laurent Toutain de Télécom Bretagne nous expose les bases des protocoles réseaux en nous parlant de container de bananes 🍊

Je profite de cette dernière semaine pour remercier l'équipe pédagogique qui a fait tout le boulot:

- Glenn: notre génie de l'électronique à la motivation infinie. Il a énormément apporté dans le forum et les cours! Merci merci merci!
- Simon (Eskimon): pointure de l'Arduino, il est venu compléter, enrichir cette seconde édition avec une pédagogie sans faille. je vous invite à dévorer son site qui est une mine d'or. MERCI!
- Laurent Mattlé: notre modeleur fou qui a carrément révolutionné la partie modélisation 3D. Ces vidéos sont parmi les plus regardées et viennent de dépasser les 10 000 vues! Merciiiiiiii!

Et on continue avec les acteurs de ce MOOC: vous!

III.2.1. Corrigé du TP 10 : Pilotage de LED par Internet

Voici la correction du TP de la semaine dernière qui reprend des éléments du cours sur l'internet des objets .

III.2.1.0.0.1. Code Arduino

Contenu masqué n°8

Ce code Arduino est disponible ici .

III.2.1.0.0.2. Montage

FIGURE III.2.1. – Montage - Correction du TP 10

III.2.1.0.0.3. Schéma

FIGURE III.2.2. – Schéma - Correction du TP 10

Pour réaliser ce montage, vous avez besoin de:
- Un Arduino
- Une platine de prototypage
- Un câble USB branché à votre ordinateur
- Une LED de votre couleur préférée
- Des fils de prototypage
- Une résistance de 220Ω
- Du temps

Notre correction s'arrête là. Pour des éclaircissements, nous vous invitons à poser vos questions sur le forum et à reparcourir le cours.

III.2.2. Internet des objets

III.2.2.0.1. Arduino et Internet

ÉLÉMENT EXTERNE (VIDEO) —
Consultez cet élément à l'adresse https://geo.dailymotion.com/player.html?video=x2kgzps&.

Merci d'avoir regardé cette vidéo. Celle-ci vous permettra, nous l'espérons de mieux comprendre ce qui se passe lorsqu'un Arduino équipé d'un shield Ethernet est branché sur une box Internet. Nous vous invitons maintenant à réaliser votre second montage connecté. Aussi, il est (vraiment) recommandé de lire les références proposées ici:

III.2.2.0.1.1. Références

- Documentation de la librairie Ethernet par Xavier Hinault
- Cours sur l'architecture d'Internet

Merci à Laurent Toutain pour ce cours

III.2.3. Qu'est ce qu'une API ?

III.2.3.0.1. Récupérer un flux RSS avec Arduino

Nous vous en parlions la semaine passée: Internet se propage dans nos objets. Cela permet aux équipements autour de nous de récupérer toute sorte d'informations comme les conditions météos, les actualités, l'état du trafic routier…
À la différence de l'homme, les machines n'ont pas besoin d'avoir des pages avec des images, des menus, une police agréable à lire, une structure de page… Ce qui compte, c'est l'information!
Nous souhaitons cette semaine vous montrer comment les machines peuvent récupérer des données météo à l'aide d'une API (une interface de récupération de données dediée aux machines).
Grâce à l'API de Yahoo Weather Forecast , nous allons pouvoir récupérer la température en degré celsius de pratiquement n'importe quelle ville du globe. Les données récupérées serviront pour changer la couleur d'une LED RGB sur une gamme de couleur représentant la température. En d'autres termes, nous allons faire un thermomètre lumineux!
Les données sont fournies au format XML (Extensible Markup Language) qui est un langage de balisage, il est couramment utilisé afin de structurer des données.
L'URL du flux XML que nous allons utiliser est la suivante http://weather.yahooapis.com/forecastrss ?w=619163&u=c
Elle est composée du chemin d'accès à l'API Yahoo http://weather.yahooapis.com/forecastrss puis comporte ensuite deux arguments séparés par le caractère &:

- **w=619163** correspond à l'ID de la ville sur laquelle on souhaite se baser (ici, Rennes), pour rechercher l'ID d'une ville, cliquez-ici
- **u=c** est relatif aux unités utilisées, ici le degré celsius à l'instar du fahrenheit

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
        <rss version="2.0" xmlns:yweather="http://xml.weather.yahoo.com/ns/rss/1.0" xmlns:geo="http://www.w3.org/2003/01/geo/wgs84_pos#">
            <channel>

<title>Yahoo! Weather - Rennes, FR</title>
<link>http://us.rd.yahoo.com/dailynews/rss/weather/Rennes__FR/*http://weather.yahoo.com/forecast/FRXX0114_f.html</link>
<description>Yahoo! Weather for Rennes, FR</description>
<language>en-us</language>
<lastBuildDate>Tue, 17 Jun 2014 11:00 am CEST</lastBuildDate>
<ttl>60</ttl>
<yweather:location city="Rennes" region=""   country="France"/>
<yweather:units temperature="F" distance="mi" pressure="in" speed="mph"/>
<yweather:wind chill="61"   direction="30"   speed="15" />
<yweather:atmosphere humidity="68"  visibility="6.21"  pressure="30.24"  rising="0" />
<yweather:astronomy sunrise="6:05 am"   sunset="10:08 pm"/>
<image>
<title>Yahoo! Weather</title>
<width>142</width>
<height>18</height>
<link>http://weather.yahoo.com</link>
<url>http://l.yimg.com/a/i/brand/purplelogo//uh/us/news-wea.gif</url>
</image>
<item>
<title>Conditions for Rennes, FR at 11:00 am CEST</title>
<geo:lat>48.11</geo:lat>
<geo:long>-1.67</geo:long>
<link>http://us.rd.yahoo.com/dailynews/rss/weather/Rennes__FR/*http://weather.yahoo.com/forecast/FRXX0114_f.html</link>
<pubDate>Tue, 17 Jun 2014 11:00 am CEST</pubDate>
<yweather:condition  text="Cloudy"  code="26"  temp="61"  date="Tue, 17 Jun 2014 11:00 am CEST" />
<description><![CDATA[...]]></description>
<yweather:forecast day="Tue" date="17 Jun 2014" low="50" high="70" text="AM Clouds/PM Sun" code="30" />
<yweather:forecast day="Wed" date="18 Jun 2014" low="52" high="74" text="Partly Cloudy" code="30" />
<yweather:forecast day="Thu" date="19 Jun 2014" low="51" high="75" text="Partly Cloudy" code="30" />
<yweather:forecast day="Fri" date="20 Jun 2014" low="53" high="77" text="Mostly Sunny" code="34" />
<yweather:forecast day="Sat" date="21 Jun 2014" low="52" high="77" text="Sunny" code="32" />
<guid isPermaLink="false">FRXX0114_2014_06_21_7_00_CEST</guid>
</item>
</channel>
</rss>
```

FIGURE III.2.3. – Le flux XML

Comme on le constate sur l'image précédente, le fichier XML est organisé grâce à des balises possédant des **attributs**. Par exemple, `temp` est un attribut de la balise `yweather:condition`.
Notre but est donc de récupérer ce flux XML depuis notre carte Arduino mais surtout cibler correctement l'information dont nous avons besoin. C'est l'étape de **parsing** (parser des données revient à extraire une ou plusieurs informations ciblées dans un ensemble de données).
Il existe plusieurs méthodes afin d'arriver au bon résultat. On peut parser des données manuellement ou bien, ce que nous allons faire ici, utiliser une librairie qui nous simplifiera la tâche. L'une des grandes forces du langage Arduino et son extrême polyvalence et son extensibilité à l'aide de librairies écrites par la communauté, qui consistent en un ensemble de fonctions exécutants des tâches spécifiques. En d'autres termes, une librairie est un pack de Lego supplémentaire qui trouvera sa place sans aucun mal dans votre coffre à jouets informatique.
TextFinder nous aidera à extraire simplement l'information dont nous avons besoin dans le flux XML.
Téléchargez la librairie et copiez le dossier dans le répertoire des librairies d'Arduino.

III.2.3.0.1.1. Préparation du montage

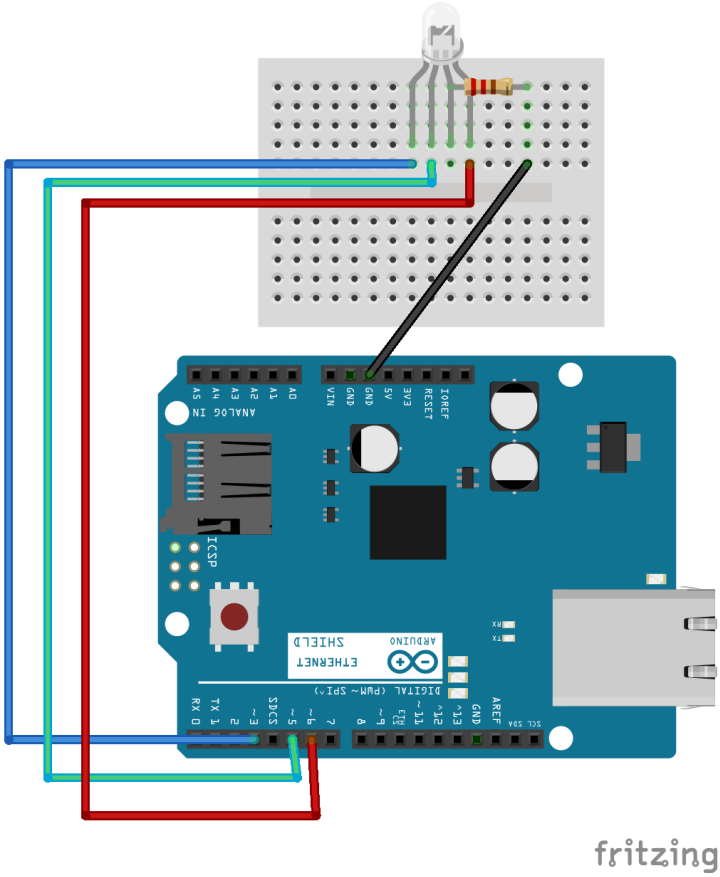


FIGURE III.2.4. – Branchement d'une led RGB

Le montage ci-dessus représente une LED à anode commune reliée aux pins digitaux 3, 5, 6 de la carte Arduino. La résistance de 220 Ohms sur la masse permet de ne pas suralimenter la LED. Enfin, on notera l'utilisation de pins digitaux PWM (symbolisés par le signe ~ à côté de leur numéro).

Ces pins sont particuliers puisqu'ils permettent la Power Width Modulation utilisée par la fonction `analogWrite` d'Arduino. Nous verrons leur utilité par la suite.

III.2.3.0.1.2. Test de la LED Un programme Arduino comporte deux blocs principaux. La fonction *setup* exécutée une seule fois lors de la mise sous tension de la carte et la fonction *loop* dont le contenu est exécuté tant que la carte est sous tension.
Le programme que nous allons écrire permet de tester notre montage. Créez un nouveau sketch Arduino. La première étape consiste à déclarer les pins utilisés par notre LED, nous utilisons des variables afin de pouvoir changer par la suite ces pins si besoin, sans avoir à rectifier manuellement dans tout le programme les endroits où ils sont mentionnés.

```
1 //Déclaration des pins utilisés par la LED
2 int ledR = 6;
3 int ledG = 5;
4 int ledB = 3;
```

Il n'y a pas besoin de déclarer ces pins comme étant des sorties (OUTPUT).

```
1 void setup(){
2   // les pins utilisant analogWrite n'ont pas besoin d'être déclar
        és en sorties
3 }
```

Nous pouvons désormais écrire des valeurs de courant sur notre LED. Nous utilisons ici la fonction **analogWrite**, qui permettra par la suite de mixer les différentes intensités de Rouge Vert ou Bleu de notre LED afin d'en changer sa couleur.

```
1 void loop(){
2   //Ecriture sur les pins de la LED
3   analogWrite(ledR, 64);
4 }
```

La LED est allumée en rouge à un quart de son intensité maximale. Les valeurs utilisées par la fonction analogWrite s'échelonnent de **0 à 255**. Un premier exercice pourrait consister à mélanger les couleurs entre elles, par exemple:

```
1  //Ecriture sur les pins de la LED
2   analogWrite(ledR, 180);
3   analogWrite(ledG, 200);
4   analogWrite(ledB, 0);
```

Ces valeurs me donnent du orange avec la LED que j'ai utilisée.

III.2.3.0.1.3. Déclaration d'une nouvelle fonction changement de couleur Notre objectif principal et de colorer la LED en fonction de la température extérieure. Plutôt que d'écrire à chaque fois les couleurs manuellement, nous allons simplifier le processus à l'aide d'une fonction

qui regroupera l'attribution des différentes intensités de rouge, vert, bleu en une seule ligne:

```
//Définition de la couleur grace à son code RGB
void setColor(int red, int green, int blue){
 analogWrite(ledR, red);
 analogWrite(ledG, green);
 analogWrite(ledB, blue);
}
```

III.2.3.0.1.4. Connexion à l'API Yahoo weather Récupérer les informations météo est très simple. En effet, il s'agit d'une simple **requête HTTP GET** permettant de récupérer le contenu d'une page. L'API de Yahoo ne nécessite pas de s'authentifier afin de pouvoir accéder aux données. En revanche, c'est le cas de certaines API comme celle de Twitter par exemple.
La suite du programme consiste à implémenter la récupération de la température en plusieurs étapes:

- Établir une connexion entre Arduino et Internet
- Récupérer une page
- Parcourir la page et récupérer l'information dont on a besoin
- Stocker cette information
- Tester cette donnée et choisir la couleur de LED appropriée
- Recommencer

Fort heureusement, le langage Arduino est très bien conçu. La plupart des méthodes liées à la connexion à un réseau sont implémentées au travers d'une classe cliente qui est native au langage Arduino. Il existera donc très peu de nuances dans le programme pour utiliser ou bien un shield ethernet ou WiFi. Un shield ethernet sera utilisé pour fournir les explications de la suite de cet article, en revanche, les sketchs pour le shield WiFi sont également téléchargeables.
Reprenons le programme que nous avons précédemment écrit. On ajoute au tout début de ce dernier les librairies nécessaires:

```
#include <SPI.h>
#include <Ethernet.h>
#include <TextFinder.h>
```

On ajoute ensuite les paramètres nécessaires à l'initialisation d'une connexion Ethernet:

```
//Adresse MAC de la carte
byte mac[] = {
  0xDE, 0xAF, 0xFF, 0xEF, 0xFE, 0xED};

EthernetClient client;

//Adresse du serveur
char server[] = "weather.yahooapis.com";

//Dernière connection au serveur en ms
unsigned long lastConnectionTime = 0;
```

```
12
13 //Etat de la connection
14 boolean lastConnected = false;
15
16 //délai entre chaque connection en ms (1mn)
17 const unsigned long postingInterval = 60*1000;
```

Le bloc d'instructions suivant est à placer dans le setup et permet l'initialisation de la connexion

```
1  void setup() {
2    //Initialisation du port série, utile au débuggage
3    Serial.begin(9600);
4
5    //Délai afin initialiser correctement le module ethernet
6    delay(1000);
7
8    //Initialisation de la connexion ethernet, attribution
       automatique d'une IP via DHCP
9    Ethernet.begin(mac);
10
11   //On affiche l'adresse IP du module
12  Serial.print("IP address: ");
13  Serial.println(Ethernet.localIP());
14 }
```

Passons maintenant au corps du programme:

```
1
2  void loop(){
3
4    //Les données sont envoyées sur le port série pour affichage
5    if (client.available()) {
6      char c = client.read();
7      Serial.print(c);
8    }
9
10     //Si il n'y a pas de connexion mais qu'il y en avait une lors
       de la dernière exécution du loop
11   //on arrete le client
12   if (!client.connected() && lastConnected) {
13     Serial.println();
14     Serial.println("disconnecting.");
15     client.stop();
16   }
17
18   //si il n'y a pas de connexion mais que 10 secondes se sont
       écoulées depuis la dernière connexion
19   //on essaye de nouveau
```

```
20    if(!client.connected() && (millis() - lastConnectionTime >
        postingInterval)) {
21      httpRequest();
22    }
23    //on stocke l'état de la connection
24    lastConnected = client.connected();
25
26  }
```

Déclarons maintenant la fonction permettant de faire une requête HTTP:

```
1   //fonction permettant d'effectuer une requête HTTP
2   void httpRequest() {
3     // si la connexion est réussie
4     if (client.connect(server, 80)) {
5       Serial.println("connecting...");
6       // on envoie le HTTP GET
7       client.println("GET /forecastrss?w=12724564&u=c HTTP/1.0");
8       client.println("Host: weather.yahooapis.com");
9       client.println("User-Agent: arduino-ethernet");
10      client.println("Connection: close");
11      client.println();
12
13      //on stock le moment de la dernière connexion
14      lastConnectionTime = millis();
15    }
16    else {
17      //si la connexion échoue
18      Serial.println("connection failed");
19      Serial.println("disconnecting.");
20      client.stop();
21    }
22  }
```

A ce stade, si vous téléversez le programme, vous devriez voir, grâce au moniteur série, le contenu du fichier XML récupéré par la carte. Il reste à parser les données afin de récupérer uniquement la température, c'est là qu'intervient TextFinder. Cette librairie permet de parcourir la page en fonction des balises que nous lui aurons demandé de cibler. Nous allons donc cibler l'attribut *temp* pour récupérer la température
Au début du programme sous EthernetClient, rajoutons l'instruction suivante qui a pour but d'initialiser l'objet client utilisé par la librairie:

```
1   TextFinder finder(client);
```

La suite est un jeu d'enfant puisqu'elle tient en quelques lignes, c'est là qu'opère la magie d'Arduino, nous allons dans un premier temps déclarer une nouvelle variable globale (à la suite des déclarations de la LED par exemple) permettant de stocker la température

```
int temp;
```

On intègre ensuite à la fonction httpRequest le code magique permettant de récupérer la température, à placer en dessous de *lastConnectionTime = millis() ;*

```
    // On cherche l'attribut temp
    if(finder.find("temp=") ){
      //On stocke la valeur voulue dans la variable temp
      temp = finder.getValue();
      //On affiche cette valeur à des fins de débuggage
      Serial.print("Temp C:  ");
      Serial.println(temp);
    }
    else {
      Serial.print("Pas de données");
    }
```

Nous allons commenter la portion de code suivant dans la boucle loop afin de ne pas être submergés d'informations dans le moniteur série:

```
/*
  //Les données sont envoyées sur le port série pour affichage
  if (client.available()) {
    char c = client.read();
    Serial.print(c);
  }
*/
```

```
//Si la température est inférieure ou égale à 15 degrés...
if(temp <= 15){
  setColor(0, 0, 255); //Bleu
}

//Si la température est supérieure ou égale à 25 degrés...
else if(temp >= 25){
  setColor(255, 0, 0); //Rouge
}

//Si la température est supérieure à 15 degrés ou inférieure à 25 degrés...
else if(temp > 15 || temp < 25){
  setColor(0, 255, 0); //Vert
}
```

Compilez et uploadez le code sur la carte pour voir le résultat dans le moniteur série. Une précision IMPORTANTE, il faut changer le délai d'actualisation à 5 secondes pour obtenir un

résultat immédiatement. En effet, les 60 secondes originales sont un délai plus qu'acceptable car la température ne change pas si rapidement. Cependant, lorsque la carte est mise sous tension, le programme attend 60 secondes avant d'effectuer la première requête!

III.2.3.0.1.5. Test de la température et changement de couleurs Souvenons nous de la fonction que nous avons écrite précédemment, setColor… c'est le moment de l'utiliser! Il va falloir tester la température pour voir si il fait chaud ou froid. Commençons avec nos trois couleurs, rouge pour chaud, vert pour tempéré et bleu pour froid. Placez le code suivant à la fin du loop:

```
1  //Si la température est inférieure ou égale à 15 degrés...
2  if(temp <= 15){
3    setColor(0, 0, 255); //Bleu
4  }
5
6  //Si la température est supérieure ou égale à 25 degrés...
7  else if(temp >= 25){
8    setColor(255, 0, 0); //Rouge
9  }
10
11 //Si la température est supérieure à 15 degrés ou inférieure à 25 ⌋
       degrés...
12 else if(temp > 15 || temp < 25){
13   setColor(0, 255, 0); //Vert
14 }
```

Vous venez de visualiser une donnée de manière tangible grâce à un actionneur. A vous de définir votre propre couleur en nuançant les couleurs par exemple, effet de fading sur la LED…
Merci à Thomas Meghe pour ce cours!

III.2.4. Exemples d'objets connectés

III.2.4.0.1. Les objets connectés du FabLab de Lannion

Élément externe (Video) —
Consultez cet élément à l'adresse https://geo.dailymotion.com/player.html?video=x2kgzxm&.

Video: Les objets connectés du FabLab de Lannion
Merci d'avoir regardé cette vidéo. Nous esperons que celle-ci vous donnera des idées pour vos futurs projets connectés. Aussi, nous vous invitons à jeter un coup d'oeil au projet laboîte de Baptiste: une petite horloge connectée à base d'Arduino qui va chercher de nombreuses informations sur Internet. En plus, comme c'est open-source (code Arduino et code serveur

), ce projet peut servir de base à vos projets.
Merci au FabLab Lannion pour cette vidéo

III.2.5. Electronique et Arduino

Avant de commencer, un avertissement que j'ai ajouté au sujet de l'utilisation d'un microcontrôleur 'nu' pour vos projets (cours Arduino sans l'Arduino de la semaine dernière).

> Le résultat de cette procédure de "flashage" c'est qu'un programme d'amorce (dit *Bootloader* en Anglais) se réveille chaque fois que le microcontrôleur est allumé ou réinitialisé (*RESET*). Entre autre, c'est ce programme qui nous permet le dialogue avec l'IDE Arduino et qui supervise le téléversement des Sketchs. Pour signaler sa présence, ce programme utilise le LED branché sur la sortie numérique 13 de l'Arduino: ce qui correspond à la broche 19 du microcontrôleur. Faites attention, donc, à ce que vous branchez sur cette broche 19 car, au démarrage, il sera configuré en tant que sortie (*OUTPUT*), et activé à l'état "haut" (*HIGH*) pendant quelques secondes. Si votre projet vous le permet, laissez un LED (avec sa résistance) branché sur la broche 19 pour éviter tout souci.

III.2.5.1. Arduinomètre connecté

Au fil des semaines nous avons amélioré notre montage de base, uniquement fait d'une simple diode. L'étalonnage, l'ajout d'autres types de sonde, jusqu'à l'intégration d'un afficheur LCD pour le rendre autonome. Nous avons dépensé beaucoup d'énergie récemment afin de libérer notre Arduinomètre de son cordon ombilical. Allez comprendre pourquoi, mais cette semaine je vous propose d'y ajouter à nouveau un cordon!
Pour commencer avec le monde de l'internet, et les objets connectés, nous allons voir s'il est possible de lire la température de notre Arduinomètre comme si l'Arduino était un serveur web.

III.2.5.1.1. Comment ?

Vous avez peut-être déjà lu mes remarques au sujet de notre sur-exposition aux micro-ondes, donc je ne m'attarderai pas plus sur ce point, mais mes montages 'connectés' sont toujours… ben, connectés, quoi. Pas de Wifi chez moi. Je suis déjà limite Electro Hyper-Sensible et donc je ne m'amuse pas à travailler avec la tête dans une soupe de micro-ondes! Ce montage se fait avec un Ethernet shield connecté avec le bon vieux câble RJ45.
Ce que j'ai en magasin chez moi c'est la version "budget serré" de cet interface, bâti sur le contrôleur ENC28J60. Carte qui se trouve même en kit , ou à construire de A à Z.
L'autre variante de carte Ethernet qui utilise un contrôleur W5100, est plus chère mais *beaucoup* plus performante et moins gourmande en ressources. Ici donc je parle de la version ENC28J60 mais, grâce à Baptiste qui m'a traduit le sketch pour une carte W5100 / Wiznet, vous pouvez aussi faire joujou si vous n'avez que cette dernière chez vous.
Voici à quoi ressemble la bête:

FIGURE III.2.5. – EtherCard

Le câblage des Ethernet shields n'est pas difficile mais, à cause de leur communication avec l'Arduino sur une liaison série un peu spéciale, nous n'avons pas le choix des broches à câbler, mis à part le signal "CS". Voici le câblage que je vous conseille:

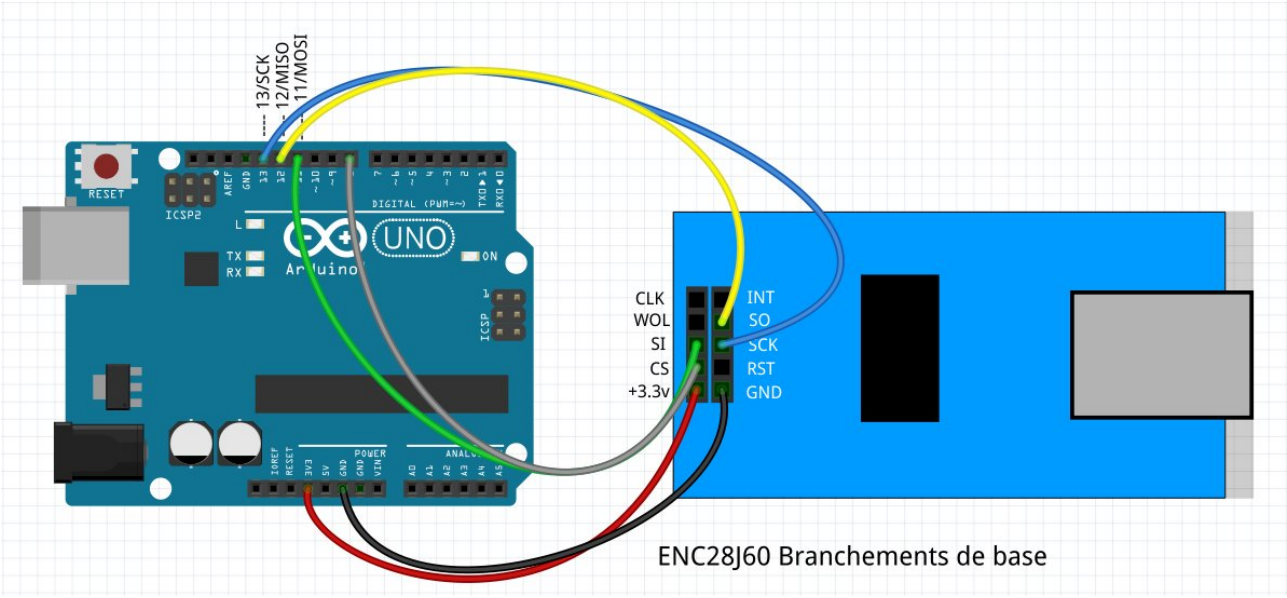


FIGURE III.2.6. – Branchement du module Ethernet

ATTENTION: l'alimentation de la carte est du 3,3v, et non 5v. Ne vous trompez pas de broche sur l'Arduino. Le choix de la broche 8 pour **CS** est essentiellement pour être compatible avec la plupart des montages utilisant cette Shield.

III.2.5.1.2. Le Sketch

Je suis obligé de partir de l'hypothèse que vous savez un peu ce qu'est le *html*, et que vous savez un peu comment cette page, que vous lisez en ce moment, vous est transmise par un serveur web. Nous allons faire abstraction de la plupart des *protocoles* utilisés (TCP/IP et al.), mais il faut soulever le capot un peu quand-même. L'essentiel à garder en mémoire est que chaque dialogue entre un objet connecté (Arduino, serveur, PC…) et un autre se passe par l'envoi et la réception de *trames* (sortes de paquets cadeaux), et que chaque trame comporte un en-tête qui explique d'où il vient, où il est sensé aller, et quel type de message il porte.
Nous allons essayer d'envoyer des trames de html pour construire une page web, et nous allons afficher la température sur cette page.

III.2.5.1.2.1. A vos marques… La bibliothèque se trouve ici: https://github.com/jcw/ethercard , et sa documentation ici: http://jeelabs.net/pub/docs/ethercard/ .
(pour la W5100 il suffit de cliquer sur "Sketch" et "Incorporer librarie", "Ethernet". Fastoche !)
Regardons ensemble les éléments majeurs du Sketch "Arduinomètre" modifié.

III.2.5.1.2.2. Déclarations

```
1 #include  <stdarg.h> // pour la fonction "format" #include
      <EtherCard.h>
2 #define Diode A0  // forward biased diode w/10K R to Vcc
3 #define Ether_CS 8 // Broche sur laquelle nous avons branché CS
```

La déclaration de `<stdarg.h>` nous permet d'accéder à quelques outils de l'environnement gcc - le compilateur qui mouline derrière l'interface Arduino. Vous trouverez plus d'informations sur le Wiki. Sinon, pas d'inquiétude particulière pour les "*includes*".

III.2.5.1.2.3. Variables J'ai mis en surbrillance quelques mots ou structures que nous n'avons pas vu jusque-là. A nouveau, je ne peux pas tout expliquer ici, mais vous trouverez plus d'informations sur le site Arduino ou sur mon-club-elec

```
// il faut une adresse unique pour votre réseau
static byte mymac[] = { 0x74,0x69,0x69,0x2D,0x30,0x31 };
// Mettre une addr. IP valide et unique sur votre réseau
static byte myip[] = {192,168,0,80};
// Espace de mémoire utilisé pour contenir les trames TCP/IP
    entrantes et sortantes
byte Ethernet::buffer[700];
// l'étoile définit un pointeur (adresse)
char* title = "Arduinomètre Website";
// pour contenir un message formaté ...
char tempString[32];
```

"Le mot-clé `static` est utilisé pour créer des variables qui sont visibles uniquement dans une fonction. A la différence des variables locales qui sont détruites et créées à chaque fois qu'une fonction est appelée, les variables *static* persistent au-delà de l'appel de la fonction, conservant leurs données entre deux appels de la fonction." (merci mon-club-elec)
`Ethernet::` fait référence à un espace de mémoire réservé, avec le nom "Ethernet". Cet espace est déclaré dans la bibliothèque EtherCard.
L'étoile après `char` nous fait soulever le couvercle de la boîte de Pandore: c'est pour indiquer que nous allons utiliser un *pointeur*. Le compilateur va nous réserver une zone de mémoire pour loger le texte "Arduinomètre Website" mais la variable `title` ne contient pas le texte, mais un ***pointeur*** vers cette zone de mémoire. C'est la bibliothèque EtherCard qui nous impose cette façon de faire - pour une fois, sur ce point, je vous demande de suivre sans essayer de comprendre.
La MAC adresse contient une valeur qui devrait bien fonctionner, sauf si vous êtes plusieurs à bricoler le même montage sur le même réseau local...
C'est un peu pareil pour l'adresse IP - mais déjà il faut vérifier que l'adresse soit compatible avec votre réseau. Pour un grand nombre de 'box', la plage 192.168.0.xxx est compatible. Regardez les paramètres réseau de votre PC pour avoir une idée.

III.2.5.1.2.4. Fonction 'format' Voilà une astuce qui va aussi être documentée sur le Wiki. Parfois nous avons besoin de *formater* (gérer la mise en page) l'information textuelle d'une façon plus évoluée que les simples instructions disponibles avec les commandes `Serial.print`. Une bibliothèque très puissante existe pour cela, mais nous n'avons pas beaucoup de place pour 'caser' ces fonctionnalités. Voici une façon d'accéder à un style de mise en forme assez sophistiqué mais pas trop gourmand.

```
// Fonction utilitaire 'détournée' du compilateur gcc // Permet de
    créer un message formaté pour impression / inclusion

void format(char *fmt, ... )
{
```

```
5     va_list args;   va_start (args, fmt );  // allocation d'espace
        en mémoire
6     vsnprintf(tempString, 32, fmt, args);  // formatage du message
        dans tempString
7     va_end (args);  // libération des ressources.
8   }
```

Vous allez bientôt voir pourquoi j'ai déclaré cette fonction.

III.2.5.1.2.5. lecture_temp

```
1  void  lecture_temp()
2  {
3    // Lire la température
4    DiodeValue = analogRead(Diode); // Lire la valeur
5    // Traduire la valeur en degrés C
6    Temperature = map(DiodeValue, 595, 409, 0, 100);
7  }
```

Tiens, tiens - cela me dit quelque chose… C'est l'algorithme pour convertir la tension lue par le convertisseur ADC en température. J'avais presque oublié le but de l'Arduinomètre!

III.2.5.1.2.6. Le setup();

```
1    // déclaration et initialisation de l'interface EtherCard avec
       "begin"
2    if (! ether.begin(sizeof(Ethernet::buffer), mymac, Ether_CS))
3       Serial.println("Erreur : carte contrôleur inaccessible");
4    else
5       Serial.println("Carte OK");
6
7    #if STATIC
8       if (! ether.staticSetup(myip))
9          Serial.println("Erreur d'attribution d'adresse IP");
10   #else
11      Serial.println("Negociation DHCP...");
12      if (! ether.dhcpSetup()) {
13         Serial.println( "Erreur DHCP, j'essaie l'adresse fixe...");
14         if (! ether.staticSetup(myip))
15            Serial.println("Erreur d'attribution d'adresse IP");
16      }
17   #endif
18      Serial.println();
```

Plusieurs nouveautés ici:
`ether.begin` est la déclaration proprement dite de la bibliothèque et de la carte. Les paramètres ne posent pas de problèmes particuliers. Notez l'utilisation de `sizeof();` pour déclarer la taille de l'espace `Ethernet::buffer` - cela évite des erreurs difficiles à diagnostiquer dans le

cas où on change la déclaration de cet espace.
`#define STATIC 0`, `#if STATIC`, `#else`, `#endif` : une astuce très pratique pour modifier le comportement du compilateur. Ici on peut choisir de ne pas tenter la négociation DHCP si on sait d'avance que cela ne marche pas chez nous. Pour cela, il suffit de changer `#define STATIC 0` en `#define STATIC 1`. Parfois il est utile d'apporter des changements sans en perdre la trace. Avec quelques mots clefs au début du Sketch, il est possible de modifier le programme compilé. Voir aussi `#define ARDUINO_UNO` et `#define TRACES` dans ce même Sketch pour d'autres utilisations des options de compilation.
C'était quoi, alors, tout ce charabia? En fait c'est ce que fait aussi votre ordinateur au démarrage: on demande une adresse IP pour qu'on puisse communiquer avec le protocole TCP/IP. C'est comme notre adresse postale sur le web. A partir de maintenant notre Arduino est connecté!

III.2.5.1.2.7. Page web Pour une fois cela tombe bien que `loop()` soit appelé sans cesse, car nous avons besoin d'une boucle d'attente - il faut attendre que quelqu'un essaie de communiquer avec nous. Quand nous voulons consulter une page web sur un serveur, nous mettons l'URL de la page dans le *browser* ou *navigateur*. L'URL est converti en adresse IP, et une trame spéciale est envoyée à cette adresse qui dit: "*Je me connecte chez vous en tant que client html, merci de me donner le contenu de cette page*". L'Arduino est désormais un webserveur, et nous attendons les clients. Regardons comment c'est fait:

```
1  void loop() {
2      word len = ether.packetReceive();  // Récupère le message
       venant du client (len = longueur du msg.)
3      word pos = ether.packetLoop(len);  // position d'éventuels
           paramètres
4  }
```

La fonction `ether.packetReceive();` regarde si nous avons reçu un message, et retourne la longueur du message reçu (ou "0" pour indiquer qu'il n'y a pas de message). Si un message a été reçu, `ether.packetLoop(len);` nous donne l'endroit dans `Ethernet::buffer` à partir duquel on peut chercher des informations utiles.

```
1  if(pos) {
2      // texte recherché dans les paramètres
3      if(strstr((char *)Ethernet::buffer + pos, "GET /?temp=") != 0)
           {
4          Serial.println("Recu commande.");
5          // Lire la température
6          lecture_temp();
7          // Construire le message
8          format("Valeur : %03d\t environ %02d°C", DiodeValue,
       Temperature);
9      }
10 }
```

Si nous avons reçu un message du client, on regarde si ce message contient la commande `?temp=`. Si c'est le cas, un appel à `lecture_temp();` permet la mise à jour des valeurs de

température. La fonction `format` (enfin!) nous aide à construire le texte à inclure dans le document html. Si vous voulez en savoir plus sur les commandes de formatage, je vous conseille de regarder `sprintf();` dans la documentation pour C ou C++.

```
1  // Pour ne pas écraser l'en-tete du protocole TCP/IP
2  BufferFiller bfill = ether.tcpOffset();
3
4  /*
5      Autre astuce pour ne pas consommer toute la memoire (RAM) de
       notre Arduino :
6      la fonction PSTR.
7      Ele force le stockage en memoire 'Flash' (espace program).
8      Sur le UNO il n'y a que 2Ko de RAM, mais 32Ko de Flash.
9  */
10
11 bfill.emit_p(PSTR("HTTP/1.0 200 OK\r\n"
12     "Content-Type: text/html\r\nPragma: no-cache\r\n\r\n"
13     "<html><head><meta http-equiv=\"content-type\" content=\"
           application/xhtml+xml; charset=utf-8\" />"
14     "<title>$S</title></head>"
15     "<body><h1>$S</h1><br />Derniere relevé de T° : $S<br />"
16     "</body></html>"), title, title, &tempString);
17
18 // Page web en html.  Les "$S" seront remplacés par les valeurs
       passées
19 // en paramètres 2, 3 et ainsi de suite
20 ether.httpServerReply(bfill.position());
```

`BufferFiller bfill = ether.tcpOffset();` initialise une trame vide, et nous positionne à l'endroit immédiatement après l'en-tête. `bfill.emit_p` est notre page web codée en html. Quelques détails intéressants: la fonction `PSTR` qui réserve la place pour ce texte en mémoire **Flash** (avec le code du programme) et non en mémoire RAM - car la RAM est très limitée sur l'Arduino UNO[1]. `bfill.emit_p` copie notre texte vers le "buffer", et en même temps elle remplace les `$S` avec les valeurs passées en paramètres. Vous souvenez-vous de l'étoile "pointeur"? C'est cette fonction `bfill.emit_p` qui exige qu'on lui passe des pointeurs au lieu des valeurs. Parce que `tempString` n'était pas déclaré en tant que *pointeur*, on force le passage de son *adresse* au lieu de sa valeur en le précédant d'un `&`.

III.2.5.1.3. Lançons-nous

Télécharger le Sketch entier ici: ArduinometreWeb.ino (ou la version W5100 ArduinometreWebW5100.ino), et l'ouvrir dans l'environnement Arduino. Vérifiez encore une fois les branchements, branchez le câble réseau. Téléversez le code, et ouvrir le **Serial Monitor**: noter que la vitesse de liaison série est de "**9600**" pour laisser la priorité à l'interface EtherCard. Dans la fenêtre du Serial Monitor vous devriez voir quelque chose comme suit:

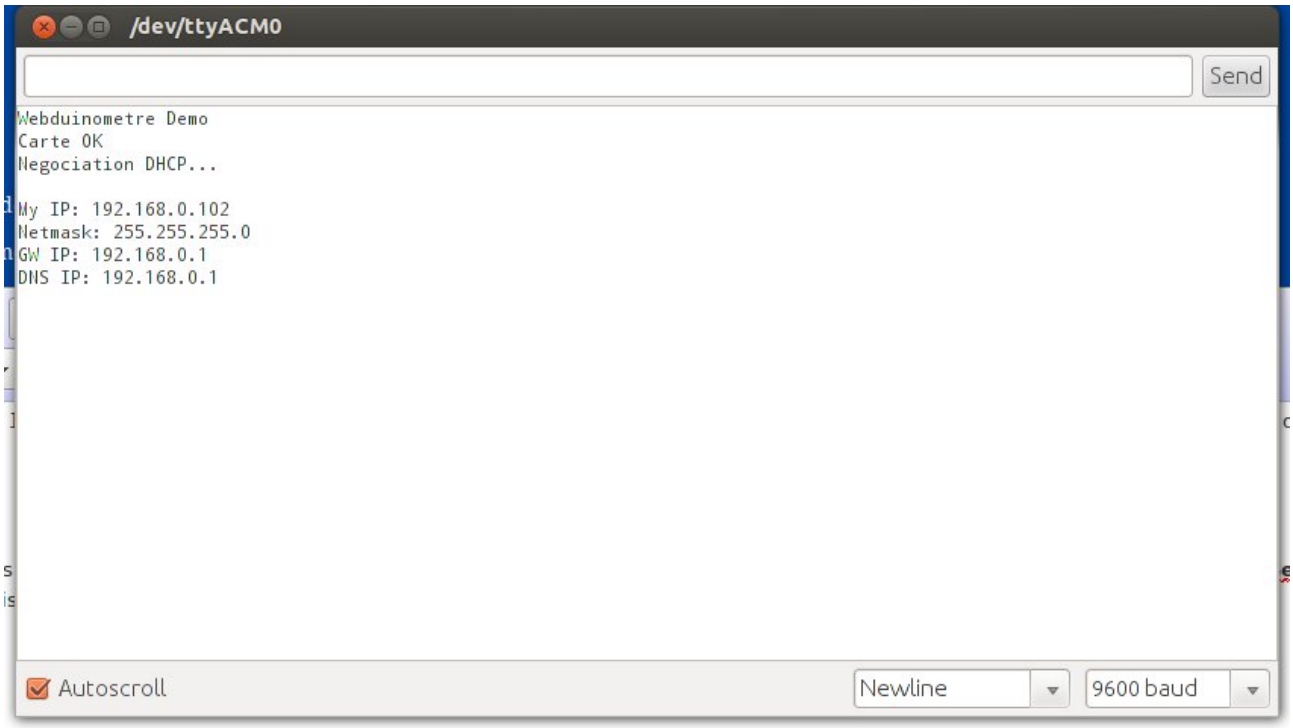


FIGURE III.2.7. – Liaison série

Il peut y avoir un délai avant l'affichage des paramètres IP, à cause des temps de négociation souvent très longues sur les routeurs et autres "box". Si maintenant on ouvre une fenêtre browser, et qu'on tape l'adresse de notre serveur ("My IP"), on devrait voir une page comme celle-ci:

FIGURE III.2.8. – Vue du serveur

Et pour voir la température, il faut ajouter `?temp=` à la fin de l'adresse:

FIGURE III.2.9. – Vue du serveur avec la température

Dites-moi… qui a oublié de brancher sa diode, hein?

Glenn Smith

III.2.6. Gravure CNC avec Inkscape

III.2.6.0.1. Inkscape GCODE Extension

Nous avons beaucoup joué avec Inkscape pour la création d'images en 2D. Je vais vous montrer que Inkscape peut aussi directement générer du GCODE pour une fraiseuse numérique ou un graveur laser.
L'astuce se trouve dans une extension gratuite, le GCODETOOLS Extension. Très simple à installer, il suffit de télécharger le fichier ici et de dé-zipper le contenu sous `Program Files\Inkscape\share\extensions\` (windows) ou `/usr/share/inkscape/extensions/` (Linux). Avec cette extension, il est possible de convertir n'importe quel dessin en GCODE pourvu que les traces soient converties en chemin (path).
J'ai eu l'idée de me servir de ma fraiseuse pour faire des enseignes et écriteaux pour mes voisins. Mais il reste un souci de taille pour graver du texte…
Les polices de caractères TrueType et al. écrivent des symboles avec leurs **contours**. Les imprimantes savent remplir les contours pour faire des symboles gras. Mais pour graver, il nous faut des symboles exprimés par **leur tracé**, ou chemin. Les logiciels spécifiques pour la gestion des machines CNC fournissent de telles polices, mais pas Inskcape. Mais, là aussi, une extension existe.
L'extension s'appel Hershey text. Voici le lien pour le télécharger. L'installation est la même que pour les GCODE TOOLS.

1. Vous voulez en savoir plus? Voici un tutoriel sur les mémoires d'Arduino

III.2.6.0.2. Faire un carré en GCODE

Pour voir comment les GCODE TOOLS fonctionnent, nous allons dessiner un carré. Ouvrez Inkscape et vérifiez que les extensions sont bien présentes: regardez dans le menu "Extensions" et vous devez voir "GCodeTools". Sous "Extensions/Render", vous devez voir "Hershey text…". Allez, nous allons dessiner ce carré:

- S'assurer que la feuille (page) est de taille A4 (dans "propriétés document") ;
- Cliquez sur la croix en bas à gauche afin que le paramètre "remplissage" soit renseigné à "aucun".
- Dessinez un carré ou rectangle au centre de la feuille

FIGURE III.2.10. – Carré Insckape

- Avec l'outil de sélection (petite flèche noire en haut à gauche), cliquez sur une des lignes du carré pour le sélectionner.
- Allez dans le menu "Chemins" (Paths) et cliquez sur "Objet -> Chemin". Vérifiez en bas de l'écran, vous devez voir le message "Chemin (4 … calque 1 …"
- Nous allons maintenant utiliser les GCode Tools pour générer le fichier pour notre CNC.

D'abord, il faut fixer une référence de point zéro:

- Dans le menu "Extensions/GCodeTools", sélectionnez l'option "Orientation points…". Une fenêtre s'ouvre. Remplissez les champs comme suit:

FIGURE III.2.11. – Menu

… et cliquez sur "Appliquer" puis "Fermer".

- Le paramètre "Z depth" correspond à la profondeur de coupe finale de notre dessin.
- Sur notre page, en bas à gauche, nous devrions avoir deux petites flèches de repère:

FIGURE III.2.12. – Flèches repère

- Ne désolidarisez pas les composants de ce groupe. Sélectionnez l'ensemble des flèches et les faire glisser jusqu'à faire coïncider le point de la flèche "0.0" avec le coin inférieur gauche du carré:

FIGURE III.2.13. – Ajustement

Maintenant, nous allons définir notre outil de gravure:

— Menu "Extensions/GCode Tools…", sélectionnez "Tools library". Une fenêtre s'ouvre. Laissez le choix "Defaut" et cliquez sur "Appliquer" puis, une fois cela fait, "Fermer".
— Vous devez maintenant avoir un pavé vert au milieu de la page, avec pleins de paramètres. On peut le déplacer pour le positionner dans notre carré (cela fait moins désordre…):

FIGURE III.2.14. – Paramètres

— Si les paramètres dans votre 'pavé' correspondent à ceux-ci, tout va bien. Il est possible de les modifier avec l'outil de texte (A|). Ces paramètres seront éventuellement à modifier spécifiquement pour votre machine CNC.
— Menu "Extensions/GCode Tools…", sélectionnez "Path to GCode", puis l'onglet "Préferences". Renseignez un nom de fichier et un dossier, "Carré.gcode", par exemple. Renseignez une valeur de quelques mm pour le paramètre "Z safe height…": il s'agit de la hauteur de sécurité à laisser pour les mouvements de la fraise entre chaque coupe. Maintenant sélectionnez l'onglet "Path to GCODE" et vérifiez que le paramètre *Depth function* soit renseigné comme "d". Cliquez sur "Appliquer". Vous aurez probablement une fenêtre d'avertissement:

FIGURE III.2.15. – Warning GCODE

Pas de panique, tout va bien. Cliquer sur "OK". Puis "Fermer" de l'autre fenêtre.

— Vous devez avoir un fichier GCODE dans le dossier spécifié dans l'étape précédente. Ouvrez le fichier avec un éditeur de texte. Vous devez avoir quelque chose comme ceci:

```
1 // Pour ne pas écraser l'en-tete du protocole TCP/IP
2 BufferFiller bfill = ether.tcpOffset();
3
4 /*
```

```
5       Autre astuce pour ne pas consommer toute la memoire (RAM) de
        notre Arduino :
6       la fonction PSTR.
7       Ele force le stockage en memoire 'Flash' (espace program).
8       Sur le UNO il n'y a que 2Ko de RAM, mais 32Ko de Flash.
9   */
10
11  bfill.emit_p(PSTR("HTTP/1.0 200 OK\r\n"
12      "Content-Type: text/html\r\nPragma: no-cache\r\n\r\n"
13      "<html><head><meta http-equiv=\"content-type\" content=\"
            application/xhtml+xml; charset=utf-8\" />"
14      "<title>$S</title></head>"
15      "<body><h1>$S</h1><br />Derniere relevé de T° : $S<br />"
16      "</body></html>"), title, title, &tempString);
17
18  // Page web en html.  Les "$S" seront remplacés par les valeurs
        passées
19  // en paramètres 2, 3 et ainsi de suite
20  ether.httpServerReply(bfill.position());
```

Ce sont les commandes en GCode pour dessiner le carré. Notez que le même dessin est répété 4 fois mais avec une profondeur (valeur de Z) qui augmente à chaque passe. Notre fraise et le moteur se portent mieux ainsi!

III.2.6.0.3. Hershey text

Pour faire l'écriteau le principe est le même. Recommencez avec une feuille A4 vierge et allez dans menu "Extensions/Render/Hershey text..." Une fenêtre (etc.)

FIGURE III.2.16. – Choix Hershey

- A coté de "Action", sélectionnez "Write glyph table" puis choisissez une police dans le menu déroulant "Font face". Cliquez sur "Appliquer".
- Dans notre dessin, nous voyons apparaître une table de correspondance ASCII et nos symboles. Supprimez l'ensemble des symboles chaque fois avant de choisir une autre police.
- Si nous saisissons du texte, et si nous choisissons l'option "Typeset that text", notre message sera converti en chemins dans notre dessin. Exemple: "Fun MOOC", "Serif medium" cela donne:

FIGURE III.2.17. – Démonstration

— Vous notez que le texte est rassemblé (group). Dés que vous avez rédimensionné et positionné le texte à votre convenance, il faut désolidariser cet ensemble.
— Menu "Extensions/GCode Tools…", "Orientation points…". Faire comme avant, déplacez l'origine vers le coin inférieur gauche de notre texte.
— Menu "Extensions/GCode Tools…", "Tools library". Déplacez le pavé pour que nous puissions toujours voir notre texte.
— Sélectionnez l'ensemble du texte, mais EXCLURE les flèches d'origine (MAJ + clic).
— Menu "Extensions/GCode Tools…", "Path to GCode", puis l'onglet "Préferences". Renseignez le nom du fichier. Revenez sur l'onglet "Path to GCODE" et "Appliquer"
— Donnez le fichier GCODE à votre CNC préféré!

ÉLÉMENT EXTERNE (VIDEO) —
Consultez cet élément à l'adresse https://www.youtube.com/embed/2nCBkYQWWlA?feature=oembed.

Video: La bêta en œuvre

Glenn Smith

Contenu masqué

Contenu masqué n°8

```
/*
  Pilotage de LED par Internet

 TP de la semaine 12 du MOOC "La Fabrication Numérique"

 Le montage :
 * Un shield Ethernet branché sur les broches 10, 11, 12, 13
 * Une LED branché sur la broche 2

 créé le 20 Juin 2014
 par Théophile Paimparay

 Ce code est en CC0 1.0 Universal

 https://www.france-universite-numerique-mooc.fr/courses/MinesTele
    com/04002/Trimestre_1_2014/about

 */

```

```
#include <SPI.h>
#include <Ethernet.h>
#include <String.h>

// Enter a MAC address and IP address for your controller below.
// The IP address will be dependent on your local network:
byte mac[] = {
  0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
IPAddress ip(192,168,1,177);

// Initialize the Ethernet server library
// with the IP address and port you want to use
// (port 80 is default for HTTP):
EthernetServer server(80);

// Initialize the LED pin
const int led = 2;

// Initialize the message sent by the client
String cmdString = "";

void setup() {
  // Open serial communications and wait for port to open:
  Serial.begin(9600);

  // Start the Ethernet connection and the server:
  Ethernet.begin(mac, ip);
  server.begin();

  Serial.print("Server is at ");
  Serial.println(Ethernet.localIP());

  // Setup as output the pin where the LED is connected
  pinMode(led, OUTPUT);
}


void loop() {
  // listen for incoming clients
  EthernetClient client = server.available();
  if (client) {
    Serial.println("new client");

    // an http request ends with a blank line
    boolean currentLineIsBlank = true;
    while (client.connected()) {
      if (client.available()) {
        char c = client.read();
        //store characters to string
        cmdString.concat(c);
```

```
69
70
71
72          // if you've gotten to the end of the line (received a
    newline
73          // character) and the line is blank, the http request has
    ended,
74          // so you can send a reply
75          if (c == '\n' && currentLineIsBlank) {
76            // send a standard http response header
77            client.println("HTTP/1.1 200 OK");
78            client.println("Content-Type: text/html");
79            client.println("Connection: close");  // the connection
    will be closed after completion of the response
80            client.println("Refresh: 5");  // refresh the page
    automatically every 5 sec
81            client.println();
82            client.println("<!DOCTYPE HTML>");
83            client.println("<html>");
84
85            //Parse the cmdString to obtain the command
86            if(cmdString.substring(6,13)== "led=on "){
87              // display LED ON in the web browser
88              client.print("LED ON");
89              // OPEN the LED
90              digitalWrite(led, HIGH);
91            }
92            if(cmdString.substring(6,14)=="led=off "){
93              // display LED OFF in the web browser
94              client.print("LED OFF");
95              // SHUTDOWN the LED
96              digitalWrite(led, LOW);
97            }
98
99            client.println("<br />");
100
101           client.println("</html>");
102           break;
103         }
104         if (c == '\n') {
105           // you're starting a new line
106           currentLineIsBlank = true;
107         }
108         else if (c != '\r') {
109           // you've gotten a character on the current line
110           currentLineIsBlank = false;
111         }
112       }
113     }
114     // give the web browser time to receive the data
```

```
115     delay(1);
116     // close the connection:
117     client.stop();
118     //   Serial.println("client disonnected");
119     //clear the command Line
120     cmdString="";
121   }
122 }
```

Listing 19 – Correction du TP 10

Retourner au texte.

Quatrième partie

Annexe et compléments

Introduction

Retrouver dans cette partie quelques éléments ”Bonus” pour la compréhension du cours.

IV.1. Bases électronique

IV.1.1. Résistances (suite)

#Valeurs de résistances Les résistances sont fabriquées et triées dans plusieurs gammes de valeurs normalisées. On ne trouve pas, par exemple, une résistance de 40Ω dans le commerce. La valeur la plus proche dans la gamme normalisée est 39Ω.
En plus, la valeur d'une résistance n'est que très rarement *exactement* la valeur indiquée: il y a toujours une tolérance (normalement 5% ou 2%). Une tolérance de 5% sur 39Ω donne de 37,05Ω à 40,95Ω!
Si vous cherchez sur internet vous trouverez les infos sur les gammes de valeurs, et comment lire les valeurs (code de couleurs) des résistances.
Voici la gamme de valeurs 'simple', la gamme E12:

FIGURE IV.1.1. – Gamme de résistances E12

Cette séquence de valeurs se répète pour chaque multiple de 10. Quelques exemples:

- 10Ω, 12Ω, 15Ω …
- 100Ω, 120Ω, 150Ω …
- 1kΩ, 1.2kΩ, 1.5kΩ … et ainsi de suite

Regardons comment utiliser des résistances pour faire quelque chose d'utile: un diviseur de tension.
#Diviseur de tension Les microcontrôleurs comme l'Arduino ont des entrées analogiques qui nous permettent de 'lire' des valeurs de tension. Si vous regardez la spécification de ces entrées, vous verrez qu'elles sont limitées à 5v maximum (en général). Comment faire si on veut surveiller des tensions supérieures à 5v? Avec un diviseur de tension:

FIGURE IV.1.2. – Un diviseur de tension

La tension d'entrée est appliquée sur l'ensemble R1 et R2. Leur résistance combinée est de $ 10 k \Omega + 1.2 k \Omega = 11.2 k \Omega$ (11200 Ohms).
Supposons que la tension d'entrée peut monter jusqu'à 15v (un circuit électrique sur une voiture, par exemple) - si on applique notre formule $I = \frac{U}{R}$ cela nous donne l'intensité à travers les 2 résistances:

$I = \frac{15V}{11200=0.00134A}$ (1,34mA)
La sortie n'est branchée que sur R2. Calculons la tension sur R2 **(U = IR)**:
$U = 0.00134 \times 1200 = 1.6V$: pratiquement 1/10 de notre tension d'entrée.
En fait on peut calculer cette tension d'une autre manière, sans calculer l'intensité:
$V_{OUT} = \frac{R_2}{R_1+R_2} \times V_{IN} = \frac{1.2}{11.2} \times 15 = 1.6V$
On peut désormais surveiller les tensions supérieures à 5v avec ce montage. Avec ces deux valeurs de résistances nous avons un diviseur de presque 1/10. Nous allons voir plus loin comment faire pour arriver à exactement 1/10.

#Résistances en série et en parallèle Regardons le schéma suivant avec trois résistances:

Figure IV.1.3. – Résistances en parallèle

Notez le courant induit par les piles (I_{tot}).
Pour analyser ce circuit, on va dire:
- I_{R_1} = le courant qui traverse R1
- I_{R_2} = le courant qui traverse R2
- I_{R_3} = le courant qui traverse R3

Déjà il est évident (j'espère) que $I_{R_2} = I_{tot}$…
Mais aussi, si on réfléchit bien, on peut aussi dire que $I_{R_1} + I_{R_3} = I_{tot}$ car le courant qui traverse les points A et B est aussi I_{tot}. Nous ne pouvons calculer I_{tot} car *on ne connaît pas la résistance totale entre les points A et B.*
Alors, un petit complément des lois des résistances:
- **Pour N résistances connectées en parallèle:**
- $R_{tot} = \frac{1}{\frac{1}{R_1}+\frac{1}{R_2}+\cdots+\frac{1}{R_N}}$

Et donc $R_{AB} = \frac{1}{\frac{1}{R_1}+\frac{1}{R_3}} = \frac{1}{\frac{1}{220}+\frac{1}{150}} = 89.2\Omega$
(Notez bien que la résistance totale est ***TOUJOURS inférieure à la plus faible des résistances***)
On peut maintenant calculer Itot, car rappelons nous que $I = \frac{U}{R}$.
Donc $I_{tot} = \frac{3V}{R_{AB}+R_2} = \frac{3}{89.2+33} = 0.0245A$ (ou 24,5 mA)
On peut aussi vérifier notre hypothèse de tout à l'heure $I_{R_1} + I_{R_3} = I_{tot}$:
D'abord il nous faut la chute de tension VAB. On peut le calculer (avec $U = I \times R$):
$V_{AB} = I_{tot} \times R_{AB} = 0.0245 \times 89,2 = 2,19V$
Puis l'intensité à travers chaque résistance:
$I_{R_1} = \frac{2.19}{220} = 0.0099A$; $I_{R_3} = \frac{2.19}{150} = 0.0146A$.
$0.0146 + 0.0099 = 0.0245$ Ouf !
Allez, c'est presque fini pour les maths pour aujourd'hui. N'empêche que ça vous donne des outils pour analyser vos circuits!

#Diviseur de tension (suite) Revenons sur notre diviseur de tension de tout à l'heure…

FIGURE IV.1.4. – Diviseur de tension réglable

Ce symbole bizarre c'est un potentiomètre: une résistance variable. Voila comment régler notre diviseur de tension afin d'avoir 1/10 de la tension en sortie. Cela dit, même avec les potentiomètres de bonne qualité, il sera difficile de régler ce montage pour donner *exactement* 1/10 de division de tension.
Je propose de faire plutôt comme ici:

FIGURE IV.1.5. – Diviseur de tension avancé

Beaucoup de résistances! Aie!
Ce montage est destiné à donner à des entrées analogiques de l'Arduino la possibilité de 'lire' des tensions supérieures au maximum autorisé de 5v.
L'interrupteur est prévu afin de sélectionner un des deux diviseurs de tension ou l'entrée directe.
Prenons le diviseur R2-R7-R8: 2,2kΩ en série avec, mince, avec quoi? Supposons que le potentiomètre est au milieu de sa course – sa résistance sera alors environ 5kΩ. Nous avons alors deux résistances en parallèle de 5kΩ et 2,2kΩ – ce qui nous donne $\frac{1}{\frac{1}{5000}+\frac{1}{2200}}$ ou 1527.7Ω.
Le ratio de la résistance, et donc la division de tension sera alors $\frac{1527.7}{1527.7+10000} = 0.13$
Pas tout à fait 1/10, mais c'est déjà pas loin. A 25% de course (2,5kΩ) cela nous donne: $\frac{1}{\frac{1}{2500}+\frac{1}{2200}} = 1170\Omega$: et un ratio de $\frac{1170}{1170+10000} = 0.104$: pas mal!
L'intérêt de ce type de montage est de donner un réglage plus fin et donc plus précis que le montage précédent avec le seul potentiomètre.
Attention[1]: n'essayez JAMAIS de mesurer la tension du secteur avec l'Arduino - même avec un diviseur de tension.

FIGURE IV.1.6. – Schéma de principe, connexion d'une source AC

(source)

Arnaud

1. Il est possible de le faire en utilisant des convertiseurs AC/AC. Généralement ce sont des transformateurs qui vont abaisser la tension. Montage plutôt réservé à des connaisseurs.

IV.1.2. Diodes et condensateurs (suite)

##Condensateurs polarisés Il faut noter que, à partir de la valeur d'environ 1uF, bon nombre de condensateurs sont ***POLARISES*** - c'est à dire qu'ils ont un "+" et un "-". On les appelle souvent condensateurs *chimiques*. Il faut les brancher dans le bon sens afin d'éviter les surprises désagréables: inversés ils ont tendance à se gonfler jusqu'à parfois se rompre carrément. Les produits chimiques dedans sont très corrosifs...
D'habitude c'est le fil "-" qui est repéré avec une bande noire sur le coté. Photo: merci Wikipedia.

FIGURE IV.1.7. – Condensateurs chimiques

##Diodes (suite) Nous avons vu qu'une diode ne laisse passer le courant que dans un sens: uniquement quand l'anode se trouve plus positive que la cathode.
Le fait de bloquer le courant dans un sens est plus utile qu'on pense. Déjà l'alimentation domestique est en courant alternatif (CA): le sens du courant s'inverse 100 fois par seconde. Vu sur un oscilloscope, le courant alternatif ressemble à ceci:

FIGURE IV.1.8. – La tension du secteur a 50Hz

Même si on diminue la tension jusqu'à des dizaines de volts, cette alternance de courant fait qu'on ne peut rien faire d'utile en électronique car nous avons besoin d'un courant dans un seul sens – dit courant continu (souvent écrit simplement "cc").
Voila l'utilité de la diode: si on passe ce courant alternatif à travers la diode, le courant sera 'redressé' - il ne passera que dans un sens.
Le circuit:

FIGURE IV.1.9. – Une diode sur la phase

Et le résultat sur l'oscilloscope:

FIGURE IV.1.10. – La tension du secteur redressée demi-cycle

On voit bien que les crêtes négatives ont étés bloquées par la diode. Mais cela nous prive de la moitié de la puissance… En plus, on voit bien que la tension est à zéro volt pendant chaque demi-cycle (10ms) – pour un microcontrôleur type Arduino cela est catastrophique: 10ms correspondent à 160 000 instructions perdues (avec une horloge de 16MHz)… Comment récupérer les demi-cycles négatifs?
Avec un montage qu'on appelle un pont de diodes:

FIGURE IV.1.11. – Un pont de diodes

Et le tracé sur l'oscilloscope:

Ça va beaucoup mieux comme ça. Mais c'est encore loin d'une source de courant stable. Il va falloir ajouter quelque chose pour lisser les crêtes…
##Condensateurs à l'aide
Nous avons déjà vu qu'un condensateur a besoin d'un certain temps pour se charger et pour se décharger. Ce temps est un fonction de la capacité (en Farads). Si on mettait un condensateur de forte valeur à la sortie de notre montage "pont de diodes", ça va peut-être lisser les variations de tension. Un condensateur de forte valeur va se charger pendant que la tension augmente, et va se décharger pendant que la tension diminue. Tentons le coup:

FIGURE IV.1.12. – Un pont de diodes + un condensateur

Et voici le résultat:

Nous avons une tension *BEAUCOUP* plus stable avec à peu près 10% de variation. On voit aussi, à la fin du tracé, le temps pour se décharger complètement.
En augmentant la valeur du condensateur, 4700uF par exemple, on peut diminuer encore les variations, mais il est préférable de faire appel à un circuit dédié, appelé régulateur de tension, si on a besoin d'une source de tension ***vraiment*** stable.

##Filtration (suite) On vient de voir comment un gros condensateur peut filtrer les variations relativement lentes. Nous vivons entourés d'une 'soupe' de champs électromagnétiques à des fréquences jusqu'à plusieurs GHz (gigahertz) – les champs parfois très puissants (WiFi, téléphonie 4G…) Nos montages autour des microcontrôleurs, qui 'tournent' à 16MHz, génèrent eux aussi des signaux parasites aux mêmes fréquences. En électronique numérique ces parasites ont la fâcheuse tendance à nous embrouiller, et en analogique c'est encore pire.
Les condensateurs à la rescousse, comme Zorro?
Eh oui. Cette fois-ci c'est le tour des 'petits', car plus la fréquence des parasites à filtrer augmente, plus il faut diminuer la capacitance. Mais les vigilants d'entre vous vont me dire: "si on diminue la capacitance, l'effet de filtre 50Hz diminue aussi". Et ils ont raison. La solution est simple: on garde le 'gros' condensateur en filtre à l'arrivée de l'alimentation, et on ajoute des petits (souvent des 10nF à 100nF) tout près des circuits à protéger des hautes fréquences.
En exemple le schéma électronique de l'Arduino Uno

FIGURE IV.1.13. – Le régulateur de tension de l'Arduino

on voit à la sortie du régulateur de tension (en haut à droite) un premier condensateur de 47uF suivi d'un autre à 100nF. D'ailleurs sur le même schéma on voit C3, C6 et C7. Si on regarde sur la platine, ces petits condensateurs sont collés tout près des circuits qu'ils protègent.
Ne les oubliez pas sur vos montages!

IV.1.3. Diodes et condensateurs (suite et fin)

#Les diodes Zener
Certaines diodes sont fabriquées de manière qu'elles bloquent le courant inverse, mais à partir d'une certain tension elles 'craquent' et laissent quand-même passer le courant.
Elles s'appellent des diodes Zener. Leur symbole est légèrement modifié:

FIGURE IV.1.14. – Symbole de la diode zener

Voici une courbe caractéristique des diodes Zener:

FIGURE IV.1.15. – Courbe caractéristique de la diode zener

Le coté droit de la courbe, "Courant direct" montre le comportement d'une diode classique: à partir du seuil d'environ 0,7v la diode se met en conduction avec peu de résistance. Le coté courant inverse commence bien: l'intensité, I, reste à 0: pas de courant inverse. On augmente la tension (négative) jusqu'au point UZ et, d'un coup, nous avons à nouveau un comportement de diode, mais en sens inverse.
A quoi bon tout cela?
Alors, sachant que le point UZ peut être choisi par les fabricants, et remarquons au passage que, dès que la diode est en conduction inverse, il y a très peu de variation de tension U pour pas mal de variation d'intensité I: nous avons peut-être un moyen de faire un régulateur de tension…
Allez, un autre schéma, garçon:

FIGURE IV.1.16. – Branchement de la diode zener

Notons d'abord l'orientation de la diode. La cathode est branchée sur le coté positif du montage. La diode choisie est marquée 5v1 (cela veut dire 5,1v – c'est une autre façon de notez les valeurs). Cette valeur de tension correspond à la tension de 'claquage' UZ. La résistance sans valeur, à droite, représente une charge quelconque. Imaginons qu'on branche une source d'alimentation variable sur V_{IN}, et qu'on augmente progressivement la tension d'entrée à partir de 0v et jusqu'à 12v. Que sera la tension aux bornes de notre résistance 'charge'?
On dit qu'une image vaut 1000 mots, donc en voici une:

FIGURE IV.1.17. – Conséquence de la diode zener

La ligne verte c'est notre source d'alimentation, la ligne bleue, la tension de sortie. On voit que la tension de sortie reste bien fixe, à environ 5v, malgré la tension d'entrée qui pique à 12v.
Et si on ajoutait une diode Zener sur notre montage "pont de diodes et condensateur de filtrage"?

La ligne verte c'est le courant alternatif redressé et filtré avec le condensateur, la trace bleue c'est la tension aux bornes de la diode Zener.
Q: quel est la tension de claquage choisie dans ce montage?[1]
Et la résistance R dans le schéma?
Cette résistance joue un rôle primordial. Si on ne met pas de résistance du tout, dès que la diode se met en conduction elle se retrouvera en court-circuit sur V_{IN}. L'intensité va très vite

dépasser les limites pour ce pauvre composant: c'est une diode Zener prévue pour dissiper 0,5w (500mW). A 5v, 500mW c'est une intensité de 0,1A (100mA)… Alors, pour limiter l'intensité on ajoute, on ajoute? … une résistance! (j'ai parfois l'impression de parler tout seul)
On peut calculer la valeur de **R** pour ce montage pour le cas où il n'y a pas de charge en sortie: il faut 'laisser' 5,1v aux bornes de la diode, et on sait que le VMAX dans cette exemple est de 12v. Il faut aussi que la diode reste en conduction, il y a une intensité minimum à respecter (valeur à chercher dans les données constructeur). Pour cette diode, je sais que IMIN c'est 5mA.
Avec notre loi $R = \frac{U}{I}$ cela nous donne $R = \frac{12–5.1}{0.005} = 1380\Omega$
Mais cela ne nous aide pas des tonnes: c'est ***sans*** charge! Il faut savoir combien 'pompe' le montage électronique que nous voulons alimenter…
Prenons l'exemple d'un microcontrôleur du type ATmega328 (la base de notre cher Arduino) qu'on utilise pour faire allumer 3 LEDs et qui 'lit' trois boutons poussoirs. La documentation pour le microcontrôleur (ATmega328_datasheet) nous dit que, sans compter les périphériques, la consommation maximale du microcontrôleur est de l'ordre de 10mA. A cela on ajoute des LEDs, disons $3 \times 20mA$; et les boutons (souvenez vous: boutons 'tirés' à 5v avec une résistance de 10kΩ): $3 \times 0.5mA$.
(ndlr: Notez bien que ces valeurs ne correspondent pas du tout à celles de l'Arduino – qui, lui, est déjà équipé d'un régulateur de tension…)
La consommation totale de notre montage est alors $10mA + 60mA + 1.5mA = 71.5mA$
Laissons une petite marge de sécurité, disons 5%, cela nous donne 75mA
Il faut que, quand notre montage 'pompe' 75mA, qu'on trouve toujours (12v - 5,1v) 6v9 à travers R. Idéalement il faut que la diode aussi reste en conduction – il faut ajouter ses 5mA de courant de maintien. Notre formule de tout à l'heure nous donne maintenant la valeur de R en pleine charge: $\frac{6.9}{0.08} = 86\Omega$ Aie!
Sans charge la valeur était 1380Ω. Comment faire, alors?
La réponse, évidemment, se trouve quelque part entre les deux… La valeur maximale de R sert pour limiter l'intensité à travers la diode quand la charge est au minimum. Dans notre exemple la charge minimum va se trouver aux alentours de 5mA (cf. datasheet) *(NDLR encore moins si on active le mode sommeil (sleep))* Pour 5mA de charge plus 5mA à travers la diode nous avons besoin d'une résistance de 690Ω. Mais à 80mA de consommation, on trouve une chute de tension de ($U = I \times R$) $0.08 \times 690 = 55V$ – donc cela ne marche pas!
La solution 'Zener toute seule' fonctionne bien *(si, si il faut me croire)* – mais ce n'est valable que dans les situations ou le courant de charge reste relativement stable. Nous n'avons pas encore trouvé notre Graal…
Hep, je me demande si, par hasard, les transistors ne peuvent pas nous aider?
#Correction de signaux
#Diodes et condensateurs ensemble…
Pour donner un aperçu d'autres possibilités avec les diodes et condensateurs, quelques petits montages pour démontrer que quelques composants peuvent souvant nous simplifier l'interface avec le monde extérieur – éliminant des casse-têtes de programmation.
##Détection flanc-montant
Ce montage avec un condensateur, une résistance et une diode permet de 'détecter' le flanc montant d'un signal :

1. Réponse: **12 V**

FIGURE IV.1.18. – Détecteur de front montant

Il est important de noter que le branchement des signaux d'entrée et de sortie ont toujours une référence vers 0v ("GND" pour 'ground' (terre) en anglais).
Au repos la résistance 'tire' la sortie vers 0v. Quand la tension d'entrée monte, le condensateur se charge et se décharge tout de suite à travers la résistance ($T = R \times C$). Quand la tension à l'entrée redescend, la sortie devrait descendre d'autant mais la cathode de la diode se trouve à un potentiel négatif par rapport à 0v et va se mettre en conduction, court-circuitant la résistance.
Dans l'oscillographe suivant, la trace verte c'est l'entrée, et la bleue c'est la sortie :

FIGURE IV.1.19. – Action du détecteur de front montant

En augmentant la valeur du condensateur on peut augmenter la durée du signal en sortie. A l'instant 2,0s on voit une petite descente de signal d'environ 0,7v. Voici notre paramètre fêtish pour les diodes : leur chute de tension en conduction.
##Détection flanc-descendant
Les mêmes composants un peu réorganisés, avec une connexion supplémentaire (le +5v) :

FIGURE IV.1.20. – Détecteur de front descendant

Et cela donne :

FIGURE IV.1.21. – Action du détecteur de front descendant

Oui, c'est l'inverse exact de l'autre oscillographe. Maintenant la résistance 'tire' la sortie au repos à 5v. On voit la petite 'flèche' provoquée par la mise en conduction de la diode.

IV.1.4. Branchements LED

Bonjour, Un petit message pour vous aider avec les LEDs (ou DELs si vous préférez). Le cours électronique va bientôt aborder les ”pourquoi” et ”comment”, ici je voulais juste vous montrer comment repérer les branchements et comment les câbler.
D'abord, les branchements:

Figure IV.1.22. – Les pôles d'une LED

Sur les LEDs classiques, le moulage présente un plat sur le bord du boitier. Il est du coté de la Cathode, le 'moins' de la LED (pour se souvenir de l'électrode négative pensez à MOKA pour «MOins CAthode»).
Lorsque la LED est neuve, on constate également qu'une patte est plus longue que l'autre. La courte est la Cathode, le moins. On peut aussi dire que c'est la 'moins' longue pour s'en rapeller.
Mais sur une LED récuperée ou déjà bien tordue, ce n'est pas aussi facile à réperer…
Alors, comment la brancher?
Si vous regardez de près, par transparence vous verrez qu'une partie à l'intérieur est en forme creux, un sorte de bol. Ce coté c'est le Cathode - la connexion NÉGATIVE.

Figure IV.1.23. – Exemple de branchement d'une LED

Vous avez tous compris (j'espère), qu'il faut mettre une résistance en série avec la LED afin de limiter l'intensité qui traverse la LED. Bientôt le module du MOOC pour clarifier le calcul de cette résistance, mais pour vos premiers pas, avec un arduino, vous pouvez utiliser une résistance de 150 à 220 ohms.
La résistance peut être câblée avant ou après la LED - il n'y a pas de différence car tout le courant qui traverse la LED traverse aussi la résistance. (pensez au branchements d'eau).
Ceci n'est qu'un exemple, ce n'est pas la seule et unique façon de câbler une LED.
Contributeurs:

— Glenn
— .AleX.

IV.1.5. Pont de diodes

Dans un autre article Wiki j'ai parlé d'un montage avec 4 diodes – le **pont de diodes** – utilisé pour redresser le courant alternatif (CA) en courant continu (CC).

Il n'est pas évident pour les premiers venus de comprendre comment ça marche, pourquoi ça marche et comment on le branche. Je reprends le sujet ici afin d'essayer de dissiper le brouillard…
Comme déjà évoqué dans le cours, l'alimentation domestique est en courant alternatif (CA): le sens du courant s'inverse 100 fois par seconde.
Vu sur un oscilloscope, le courant alternatif ressemble à ceci:

FIGURE IV.1.24. – Oscillogramme du secteur, 50Hz

Sur cette trace la tension maxi est de 25v – c'est pour la démonstration.

N'oublie JAMAIS que la tension secteur est plutôt de 230v à 250v et que ces tensions là sont LÉTALES.

Fin d'avertissement.
Regardons cette trace d'oscilloscope de plus près… ou, plutôt, regardons le schéma électronique complet:

FIGURE IV.1.25. – Un transformateur

L'oscilloscope est branché sur les points A et B. La résistance de 100Ω est notre 'charge'. La bidule un peu étrange au milieu est un transformateur de tension. Les 250V secteur sont 'transformées' en 25v (mais toujours en courant alternatif). C'est un moyen fiable de nous séparer des dangers du secteur…
Il faut regarder ce qui se passe pendant chaque demi-cycle alternatif. On ajoute la diode. Pendant le premier demi-cycle, la borne '4' du transformateur devient progressivement positive par rapport à la borne '3' [1] :

FIGURE IV.1.26. – Transformateur + une diode

Vu l'orientation de la diode, le courant peut passer. La tension redescend vers zéro volt ; c'est la fin du demi-cycle. Pendant le demi-cycle suivant toute est inversé : c'est la borne '3' qui devient positive par rapport à la borne '4'.

FIGURE IV.1.27. – Tension secteur redressée demi-cycle

On voit bien que les crêtes négatives ont été bloquées par la diode. Comment récupérer les demi-cycles négatifs ? Pour faire passer le courant, il faut carrément inverser la diode – mais cela va inverser la polarité de notre montage…
Vient ce montage bizarre avec 4 diodes – le pont de diodes…

FIGURE IV.1.28. – Le pont de diodes

On peut câbler 4 diodes comme dans le schéma, ou on peut utiliser un pont ”préfabriqué”. Ils existent en diverses formes et tailles:

FIGURE IV.1.29. – Photo de ponts de diodes

Voici un autre façon de le dessiner :

FIGURE IV.1.30. – Schéma alternatif du pont de diodes

Et voici ce qui se passe pendant le premier demi-cycle : les diodes «passantes» sont en rouge :

FIGURE IV.1.31. – Alternance positive, le premier cycle

Et le deuxième :

FIGURE IV.1.32. – Alternance négative, le second cycle

Si vous regardez bien, vous voyez que chaque demi-cycle est correctement «aiguillé» par les diodes, et on récupère une succession de cycles avec la bonne polarité.
Comme sur la trace d'oscilloscope :

FIGURE IV.1.33. – Tension secteur redressée

Et voila! Vous savez (presque) tout sur le pont de diodes!

IV.1.6. Bouton poussoir

Plusieurs personnes sur le forum demandent le pourquoi de la résistance pour le montage du bouton poussoir.
D'abord, les branchements. Voici comment il faut orienter les boutons:

FIGURE IV.1.34. – Orientation du bouton poussoir

Notez bien que pour trouver la patte 1 il faut orienter le bouton pour que les pattes sortent du haut et du bas. Les pattes 1 et 2 sont reliées entre-elles à l'intérieur du bouton. Pareil pour les pattes 3 et 4. Il n'y a donc pas de souci si le bouton est inversé (c'est à dire tourné de 180°).
Montage platine, entrée tirée vers 5V:

FIGURE IV.1.35. – Montage d'un bouton

1. C'est une hypothèse : il fallait commencer dans un sens ou l'autre…

Et le schéma:

FIGURE IV.1.36. – Schéma électronique du branchement du bouton

Et maintenant quelques explications
Quand on définit une broche de l'Arduino comme entrée (avec la commande `pinMode(nn,INPUT)`) l'électronique à l'intérieur du microcontrôleur tente de valider en permanence une entrée au niveau BAS ou HAUT.
Il est difficile d'avoir un signal en entrée qui est toujours à *exactement* 0V ou *exactement* 5V.
Le microcontrôleur travaille donc par paliers ou seuils. Une tension d'entrée sous le seuil est considérée comme étant le niveau «bas»; supérieur au seuil c'est le niveau «haut». En plus (et ceux d'entre vous qui ont déjà eu affaire à des amplificateurs audio savent ce qu'est le ronflement) une entrée laissée «flottante» est parasitée par toutes sortes de signaux ambiants.
Pour que nos programmes fonctionnent bien, il est primordial que les entrées soient TOUJOURS dans un état connu. On «tire» alors l'entrée vers 0V/GND ou +5V pour que, au repos, l'entrée reste tranquillement soit bas, soit haut.
Partant du principe que l'entrée est branchée sur 0V (état bas). Pour changer d'état il faut d'abord débrancher l'entrée du fil 0V puis le brancher sur le fil 5V: sinon en branchant le +5V sur le 0V on provoque un court-circuit! C'est LA qu'on trouve l'intérêt de la résistance.
Si on «tire» notre entrée vers un état haut ou bas avec une résistance d'une valeur relativement élevée, le problème de court-circuit ne se pose plus.
Un exemple: on tire l'entrée vers 0V avec une résistance de 10kohms (10,000 ohms).

FIGURE IV.1.37. – Une résistance en pull-down

Si maintenant on branche le 5V pour amener l'entrée à l'état haut, cela nous donne:

FIGURE IV.1.38. – Une résistance en pull-down avec le 5V

Prenons notre cours sur le loi d'ohm (semaine 2): $I = \frac{U}{R}$ pour trouver le courant qui traverse la résistance: $I = \frac{5V}{10000=0.0005=0.5mA}$: pas grand chose.
Et, si vous regardez de près, l'entrée de l'Arduino se trouve bien branché sur 5V.

Et le composante bleue sur la platine? Ben je ne veux pas trop en parler pour l'instant… C'est un condensateur et il est là pour "lisser" les fluctuations dues aux rebonds des contacts des interrupteurs. On parle des condensateurs plus tard dans le cours.
Dernière question, pourquoi «tirer» vers 5V en non le 0V comme dans l'exemple ici?
En fait c'est plutôt une convention historique… Autrefois les broches d'entrée sur les composants électroniques «consommaient» moins de courant si elles étaient tirées vers 5V que si elles étaient tirées vers 0V… De nos jours, avec les microcontrôleurs modernes, la différence est négligeable. C'est aussi par compatibilité avec certaines fonctionnalités de l'Arduino[1].
Vous pouvez donc câbler vos boutons poussoirs comme ici:

FIGURE IV.1.39. – Câblage du bouton et de sa résistance de pull-down

Dans cette configuration, avec `digitalRead()` vous auriez un état '0' au repos, et l'état '1' quand le bouton est appuyé.
Glenn

IV.1.7. Photorésistance (et simulateur)

La photorésistance a des caractéristiques comme tous les autres composants, le plus souvent ce composant est connu par le nom "LDR" (light-dependent resistor).
Les deux caractéristiques les plus importantes sont:

- La résistance dans l'obscurité (R Dark) => elle varie entre 100k et 2M (pour les modèles les plus courants) suivant la référence du composant
- La variation de cette résistance placée en pleine lumière => la résistance (R Dark) est divisée par 20 à 50 (dans certains cas plus de mille) suivant la référence du composant.

###Résistance en série Le montage préconisé pour l'Arduino est le suivant :

FIGURE IV.1.40. – Branchement de la photorésistance

Les valeurs obtenues lors de la lecture de la valeur analogique de la LDR connectée a Arduino dépendent de la lumière, des caractéristiques de la LDR et de la valeur de la résistance placée en série avec la LDR.
Pour balayer au mieux les 1024 valeurs possibles, il est utile de choisir judicieusement la valeur de la résistance. Choisir comme valeur de la résistance: (R-Dark / 5) donne de bon résultats pour des valeurs de R-Dark entre 100k et 2M.

1. Pour les personnes qui ont 'découvert' les options de résistances `pullup` intégrées dans l'Arduino: "pull-**up**" veut dire "tirer vers le **haut**": au repos l'entrée est donc tirée vers le 5V, pas vers la masse.

Alors pourquoi cette résistance en série?
La fonction `analogRead()` est censée lire une **TENSION** de 0 à 5v. Pour lire une tension il faut avoir un **courant** qui traverse une **résistance** (loi d'ohm, cours semaine 2).
Donc avec notre montage l'Arduino est en train de lire la tension aux bornes de cette résistance. Comme nous avons vu dans le Wiki "*Résistances (suite)*", ce montage est en fait un *diviseur de tension* ***variable***.
Et le choix de la valeur de résistance? pour limiter l'intensité quand la photorésistance se trouve en plein soleil (car sa résistance peut chuter à quelques centaines d'ohms seulement) on limite l'intensité à quelques milli-ampères. De plus, la valeur de la résistance conditionne l'amplitude des valeurs mesurées entre la situation obscurité et la situation pleine lumière.
Comme toujours, afin de calculer des valeurs optimales il faut chercher la documentation technique de *votre* photorésistance…
###Mesure de R-Dark
En utilisant le schéma proposé dans le TP, placer la LDR dans l'obscurité, faire varier la valeur de la résistance en série jusqu'à obtenir: «sensor = 511» ou une valeur proche. La valeur de la résistance série donne la valeur R-Dark.
###La LDR dans le simulateur
Une LDR est disponible dans les composants additionnels. Cette LDR a pour valeur R-Dark=180 kOhm et à la lumière, sa résistance tombe à 3,6 kOhm. La résistance à placer en série devrait être de 36k mais 1k donne des résultats avec moins d'amplitude mais acceptables.
Au lancement du programme, la LDR est supposée être dans l'obscurité, un click dessus et elle passe en pleine lumière. Il ne semble donc pas possible d'obtenir une variation continue de type analogique. L'utilisation d'un potentiomètre à la place de la LDR peut permettre des valeurs analogiques par paliers.

IV.1.8. Caractéristiques des composants

En électronique, une chose indispensable au bon fonctionnement est la documentation. En premier, la documentation produite par la personne qui conçoit le circuit. Mais cette personne devra également disposer de la documentation de l'ensemble des composants utilisés. Il s'agit de la fameuse *Datasheet*, le document qui contient toutes les informations possibles sur le composant qu'il traite.
La lecture de ce document parfois un peu long (plusieurs centaines de pages pour les microcontrôleurs, même les plus simples) et intégralement en anglais technique rebuterait n'importe qui. Mais l'électronicien n'est pas n'importe qui et l'économie de cette lecture n'est pas vraiment une option.
Par exemple, voici les réponses aux questions fréquemment abordées dans les forums:

- Quelle est la chute de tension de ma LED? Se rapporter au document suivant: LED (j'ai pris ce fabricant car il produit les LEDs que j'utilise, même si ce n'est pas ce modèle exactement.).
- Quel est le courant maximum débité par une sortie de l'Arduino? Se rapporter au document suivant: arduino section "Electrical Characteristics" rubrique "Absolute Maximum Ratings" p 313, ligne "DC Current per I/O Pin".
- Quelles sont les caractéristiques de mes résistances? Se rapporter au document suivant: résistances (6 pages pour une résistances, tout y est).

Pour ceux qui sont pressés, il y a une section assez courte toujours présente dans ce type de document: *Absolute maximum ratings* qui liste les valeurs maximales (tension, intensité, dissipation thermique…) acceptables avant dégradation progressive ou définitive du composant.

S'il y a une chose obligatoire à lire avant d'utiliser un nouveau composant, c'est bien ça.
Donc, la consultation de la documentation peut:

— faire gagner du temps
— faire gagner de l'argent (on peut éviter de faire fumer des choses et donc de devoir les racheter)
— faire progresser en anglais le lecteur.

Pourquoi s'en priver?
TPE

IV.1.9. Mesures de tension et d'intensité

#Le multimètre
Il est presque indispensable d'avoir un multimètre à disposition pour contrôler vos montages électroniques – surtout pour éviter d'endommager ('cramer') certains composants qui n'aiment pas être inversés ou suralimentés… La plupart des multimètres en vente actuellement sont des multimètres à affichage numérique, mais il en existe aussi à affichage analogique (à aiguille):

FIGURE IV.1.41. – Un multimètre numérique

FIGURE IV.1.42. – Un multimètre à aiguilles

Un multimètre simple, d'un coût d'une dizaine d'Euros, suffit pour commencer en électronique.

Vérifiez bien que les branchements des deux sondes -ET- le réglage de l'appareil correspondent bien aux conditions à mesurer.

Un multimètre réglé sur l'échelle de 20mA ne va pas aimer voir passer un courant de 20A. Et n'essayez pas de mesurer la *tension* avec le multimètre réglé sur une échelle d'*intensité* - c'est 'vu' comme un court-circuit! Parfait pour cramer des composants!
##Mesures de tension
Réglé sur une échelle de tension, le multimètre va nous renseigner sur la chute de tension réelle aux bornes de la résistance. Ici on voit le montage en version schématique et en 'réel':

FIGURE IV.1.43. – Montage de la connexion du voltmètre

FIGURE IV.1.44. – Schéma électronique de branchement du voltmètre

Notez comment le voltmètre est branché: branchement +ve (rouge) sur le plus positif des deux fils de la résistance. Sur un montage 'inconnu' il faut parfois tâter en regardant l'affichage du voltmètre afin de savoir quel côté d'un composant est le +ve.
##Mesures d'intensité
Vous avez vu que les mesures de tension peuvent se faire sans modifier le câblage d'un circuit. Pour mesurer l'intensité ce n'est pas aussi facile. Pour voir, par exemple, l'intensité du courant à travers la DEL, il faut couper le circuit pour insérer le multimètre, pour que tout le courant qui traverse la DEL passe aussi à travers le multimètre:

FIGURE IV.1.45. – Schéma électronique de branchement de l'ampèremètre

Notez à nouveau qu'il faut brancher l'ampèremètre dans le bon sens: côté "+" branché sur le point le plus +ve des deux. Dans cet exemple on peut insérer l'ampèremètre n'importe où dans le circuit car nous n'avons que deux composants (la résistance et la DEL) en série.
Glenn

IV.1.10. Zéro, un, euh... (calcul et notation binaire)

> Il y a 10 sortes de gens au monde: ceux qui connaîssent le binaire et les autres.
>
> *Anonyme*

Cette plaisanterie, opaque pour un non informaticien, nous rappelle que tout est affaire de notations quand il s'agit de nombres et qu'à la question «Quel est le successeur de 1» on répond naturellement «2». Seulement le numérique n'aime pas trop compter au delà de 1 mais cela n'empêche pourtant nullement de faire des calculs.
En numérique l'élément le plus fin s'appelle un bit (contraction de *BInary uniT*), isolé il ne permet de représenter que deux états, on a le choix pour distinguer ces deux états:

- vrai / faux,
- haut / bas,
- rigolo / triste,
- rouge / vert,
- ouvert / fermé,

— allumé / éteint,
— …
— 0 / 1.

On utilisera la dernière forme car c'est plus simple en électronique: 0 pas de courant (au potentiel de la masse: 0 V) et 1 du courant qui passe (au potentiel de Vdd, le plus souvent 3,3 V ou 5 V).
Que peut-on faire avec un bit? Déjà on peut allumer ou éteindre une LED, donc si on a en entrée A un bit et une sortie S reliée à une LED (avec une résistance !), nous pouvons écrire ce que l'on appelle une table de vérité comme ceci:

A	**S**
0	**0**
1	**1**

Table de vérité de l'identité

L'état de la sortie est égal à l'état de l'entrée, cet opérateur s'appelle l'**identité** (*buffer*) et n'a guère d'utilité (sauf pour amplifier un signal un peu faiblard), on peut utiliser aussi la notation: $S = A$
On peut imaginer l'opérateur inverse:

A	**S**
0	**1**
1	**0**

Table de vérité du NON logique (inverseur)

On inverse l'état de la sortie par rapport à l'état en entrée, c'est l'opérateur **non** (*not*). On notera:
`S = ¬ A` ou simplement `S = non A` ou encore en plaçant une barre horizontale au dessus du A (chose que je ne peux/sais pas faire avec le Wiki).
Bien sûr il est possible de d'avoir deux autres opérateurs triviaux:
`S = 0` et `S = 1` il n'y a pas vraiment de notation mais ceux qui suivent ce MOOC savent que c'est une histoire de résistance qu'on appelle *pulldown* ou *pullup*, ça mérite pas un opérateur pour ça!
Mais on ne va pas aller très loin avec juste ça en poche…
Et si on s'amusait plutôt avec deux entrées A et B? Du coup il y a beaucoup plus de combinaisons possibles, voyons ce que l'on peut faire. Si maintenant je souhaite allumer ma LED que si A **et** B sont à 1, ça donnera la table de vérité suivante:

A	**B**	**S**
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**1**

Table de vérité du ET logique

C'est l'opérateur **et** (*and*) mais on peut l'assimiler aussi à la multiplication car:

— $0 \times 0 = 0$
— $0 \times 1 = 0$
— $1 \times 0 = 0$
— $1 \times 1 = 1$

On le note donc sous la forme d'une multiplication (en utilisant le point comme symbole): `S = A . B` mais `S = A et B` est tout à fait compréhensible aussi.
Déjà c'est plus intéressant. Un exemple d'utilisation est pour une machine outil dangeureuse: on veut s'assurer qu'au moment où la découpeuse à plasma va fonctionner l'opérateur aura les deux mains en dehors de la zone dangeureuse, posées sur des gros boutons excentrés A **et** B. Tant que l'on a pas A **et** B, S restera à 0 et n'alimentera pas le générateur de plasma.
Autre opérateur intéressant: le **ou inclusif** (*or*), c'est une addition sans retenue, voici sa table de vérité:

A	**B**	**S**
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**1**

Table de vérité du OU logique

ou `S = A + B`
On parle de ou inclusif car dans le cas où les deux entrées sont à 1 on choisit de mettre la sortie également à 1. Mais on peut imaginer un **ou exclusif** (*xor*) qui lui, dans ce cas, mettrait la sortie à zéro:

A	**B**	**S**
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**0**

Table de vérité du XOR, "OU exclusif"

ou `S = A  B` (un signe `+` placé dans un cercle).
Il y a une notation que l'on utilise pour les schémas électroniques, voici la symbolique des opérateurs (plutôt que d'opérateurs on parle de portes logiques en électronique) que nous avons vu:

FIGURE IV.1.46. – Symbole des portes logiques

Attention: c'est le petit cercle en sortie de la porte non qui indique que l'opérateur non s'applique.
Vous disposez maintenant de **l'ensemble** des ingrédients pour faire la partie logique d'un microprocesseur!
Allez, on continue un petit peu: comment on va arriver à compter jusqu'à 2 vu que `1 . 1 = 1`, `1 + 1 = 1` et que `1  1 = 0`?
Il va falloir deux bits pour pouvoir représenter cette valeur donc on va avoir notre sortie S et puis une autre sortie que l'on va appeler C. C contiendra le débordement de l'addition, la retenue ou *carry* en anglais. Avant de donner la solution, écrivons la table de vérité:

A	B	C	S
0	0	**0**	**0**
0	1	**0**	**1**
1	0	**0**	**1**
1	1	**1**	**0**

Table de vérité de l'opération

Quand cela déborde `C` passe à l'état `1`, donc `1 + 1 = 10` en binaire!
Regardons les deux sorties une à une. Pour S on voit que c'est la table de vérité du ou exclusif et pour C la table de vérité du et. Dès lors on peut réécrire notre système comme ceci: S = A B; C = A . B
En version schéma électronique cela donne ça:

FIGURE IV.1.47. – Un additionneur 2 bits

C'est un additionneur 1 bit, on peut étendre ce principe pour travailler sur 2, 3, 4,..., x bits mais cette forme n'est pas la plus efficace car il y un délai lié à la propagation de la retenue en cascade, mais cela fonctionnera.
On peut en reliant les deux sorties S et C à un ou inclusif suivi d'un non (une porte non ou ou *nor* en anglais) détecter quand le résultat est zéro, etc. Avec beaucoup (vraiment beaucoup) de petites briques comme cela on construit *effectivement* un microprocesseur.
Bon alors *maintenant* vous vous situez dans la catégorie `00` ou `01`? ;-)

IV.2. Éléments de programmation

IV.2.1. Les commentaires

Les commentaires, même s'ils n'agissent pas *directement* sur le comportement du programme, sont des éléments essentiels dans le travail de programmation. Souvent ils sont perçus comme une contrainte et on en met parce qu'il faut en mettre, sinon c'est mal vu, alors que c'est une richesse et un complément au code qui s'avèrent utiles en particulier pour élaborer et faire évoluer un programme.

IV.2.1.0.1. La forme

Deux façons de mettre des commentaires sont possibles:

```
1 /*
2   Ceci est un commentaire
3   sur plusieurs lignes
4 */
```

Cette façon de commenter permet d'utiliser plusieurs lignes ce qui est très pratique quand on veut décrire, par exemple, un algorithme.

```
1 // Ceci est un commentaire court
```

Avec la deuxième façon, les commentaires sont limités à la ligne courante. On les place généralement à la droite du code pour préciser l'effet que cette ligne de code va produire. Le texte est libre dans les commentaires sauf pour la première forme où `*/` est interdit puisque cela signifie fin de commentaire. Il est également interdit d'imbriquer les commentaires longs, c'est-à-dire:

```
1 /* /* ceci est une construction invalide */ */
```

IV.2.1.0.2. Quand mettre des commentaires ?

Il y a trois écoles:

— jamais;
— quand on estime que c'est utile;
— à chaque ligne de code son commentaire.

Si on élimine les extrêmes, il ne reste plus que «quand on estime que c'est utile». Or, ce n'est pas forcément toujours facile de déterminer si c'est utile ou pas.

IV.2.1.0.2.1. Utile pour qui ? Déjà pour soi! Les débutants pensent souvent que ce n'est pas la peine, juste une perte de temps, parce que c'est *mon* code, c'est *moi* qui l'ai écrit, *je* sais comment ça marche, etc. Quand on revient, ne serait-ce que trois mois plus tard, sur *son* code et que l'on veut le modifier c'est à ce moment que l'on s'aperçoit que des commentaires *auraient* été utiles. Bien sûr on peut se remettre dans le bain avec un programme de 20 lignes, mais quand il y a 2000 lignes, 20000 lignes ou 200000 lignes c'est une toute autre histoire…
Ensuite le code est de plus en plus partagé et donc lu par des personnes différentes. Qu'on le veuille ou non, chaque programmeur a son style et «entrer» dans le code d'une autre personne n'est pas toujours immédiat, il y a besoin de guides comme une documentation et… des commentaires.
Donc «l'audience» va aussi déterminer la quantité de commentaires nécessaires et utiles. Si le programme s'adresse à des débutants qui apprennent le langage en plus de la programmation, mettre un commentaire par ligne, éventuellement redondant avec le code lui-même, est utile. En effet, cela permet de conforter le lecteur dans son apprentissage. En revanche pour un programme destiné à une audience de développeurs professionnels, c'est une pollution (essayez de proposer un correctif pour le noyau Linux avec un commentaire par ligne pour voir la réaction que cela suscite !).
Néanmoins, dès lors que l'on commence à être à l'aise et que l'audience du code s'adresse à des personnes au moins du même niveau, il convient de prendre l'habitude de mettre des commentaires **et** d'écouter le retour des lecteurs de façon à vérifier la pertinence des commentaires.

IV.2.1.0.3. Les commentaires subliminaux

On n'y pense pas toujours mais les noms que l'on donne aux choses sont essentiels pour la compréhension. Un nom de variable doit indiquer la nature de l'objet ou de la grandeur qui est représentée, sa fonction, sa raison d'être… Pareil pour un nom de fonction. Toutefois, afin de raccourcir les noms il est fréquent d'utiliser des acronymes ou d'enlever la plupart des voyelles, d'utiliser un préfixe commun afin d'identifier une famille de fonctions, cela demande un petit peu d'entraînement.
Note: En programmation, dans le code, il est d'usage d'utiliser des mots anglais, je ne dérogerai pas à cette règle dans les exemples, en revanche les commentaires seront en français.
Par exemple voici quelques noms de fonctions utilisées en langage C pour manipuler des chaînes (cela n'a aucun intérêt pour l'environnement Arduino, c'est juste une illustration) :

```
strcmp   // STRing CoMPare - comparaison de chaînes
strcpy   // STRing CoPY - copie d'une chaîne
strdup   // STRing DUPlicate - duplication d'une chaîne
strlen   // STRing LENgth - longueur d'une chaîne
...
```

Le nom doit suffisamment être évocateur pour que même une personne n'ayant pas connaissance de cette fonction puisse deviner à quoi elle sert et continuer la lecture du code.
En contre exemple voici ce qu'il faut éviter à tout prix:

```
int var = 0; // j'initialise var à 0
```

Aucune sémantique, ni côté code ni côté commentaire.
En revanche:

```
int fill_70_flag = 0; // indique seuil de remplissage 70% atteint
```

Possède de la sémantique tant côté code que côté commentaire.

IV.2.1.0.4. Où mettre un commentaire ?

IV.2.1.0.4.1. Au début du programme Outre un descriptif sommaire, le nom de l'auteur et la licence applicable. C'est l'endroit idéal pour expliquer ce que vous allez faire. Ce type de commentaire s'écrit **avant** de coder et permet de vérifier que vous avez une idée précise de ce que vous allez faire tout en étant capable de l'expliquer à quelqu'un d'autre. C'est souvent le moment où l'on découvre des problèmes auxquels on n'avait pas pensé! Par ailleurs, c'est aussi l'occasion de documenter un algorithme un peu compliqué ou de mettre des références vers des ressources (liens vers des sites web, références à des livres…) qui vous ont servi pour le travail préparatoire. Il faudra probablement revenir compléter ce commentaire une fois le programme réalisé car nul doute qu'il y aura des précisions ou des choses de découvertes entre temps qui méritent d'y figurer.
Voici un exemple (fictif et tronqué):

```
/*
  Pilotage d'une fraiseuse pour réaliser des circuits imprimés
  Copyright 2014 © fb251
  Code sous licence WTF 2.0 : http://wtflicence.com

  Nous nous appuyons sur le format RS-274X (Gerber) révision H
  les spécifications sont téléchargeables sur :
    http://www.ucamco.com/

  Note : ce format est assez vieux (~ 1970) et est dérivé de la
  norme RS-274. Il faut faire très attention aux arrondis selon
  que le fichier source utilise comme unité les pouces ou le
  système métrique. Par ailleurs, certains logiciels comme Eaggle
  ne respectent pas strictement le format ce qui oblige à des
  contorsions (signalés dans le code avec le marqueur $HACK$).

  La difficulté principale consiste à créer les pistes d'isolement
  car il faut inverser la polarité RS-274 mais ce n'est pas
    suffisant
  aussi nous utilisons l'algorithme...
*/
```

Notez qu'une bonne pratique consiste à indiquer que l'on utilise des marqueurs dans les commentaires du code (ici $HACK$) pour signaler que le code a un comportement particulier et non naturel dicté par des conditions spécifiques. Des marqueurs standards existent, citons:

— TODO - il reste du code à faire,
— FIXME - il y a un bug à fixer.

Personnellement j'encadre le marqueur avec des $ comme ça c'est très rapide à trouver avec la fonction de recherche de l'éditeur de texte.
Voici un exemple d'utilisation de marqueur:

```
if (0) // $FIXME$ : on n'exécute jamais la branche if !
```

IV.2.1.0.4.2. Avant une fonction Lorsque l'on utilise du code modulaire découpé en fonctions, dès lors que celles-ci ne sont pas triviales, il convient de placer en en-tête de fonction un commentaire sous forme multi lignes qui va expliquer le rôle de la fonction.
Voici un exemple tiré cette fois de code assembleur (le code ne figure pas, on ne s'intéresse qu'aux commentaires):

```
/*! int fss_strcmpa (const char * s1, const char * s2)

  @brief Compare rapidement deux chaînes de caractères selon les
  même conventions que strcmp()

  @param s1 est la première chaîne
  @param s2 est la deuxième chaîne
  @return > 0 si s1 > s2, < 0 si s1 < s2 et 0 si s1 == s2

  @remark ATTENTION ! Cette fonction ne peut fonctionner qu'avec
    des
  chaînes alignées sur des frontières modulo 16 octets, la fin de
    la
  chaîne doit être complétée avec des zéros jusqu'à la frontière de
  16 octets qui suit. Les registres xmm0, xmm1, xmm2, rcx et rdx
    sont
  écrasés. Le processeur doit obligatoirement disposer de SSE > 2.

  @remark Le code a été testé et optimisé sur AMD Phenom II X6, X8
  et Intel Core i3, Xeon, Pentium et Atom.
*/
```

Évidemment, c'est long et ça paraît fastidieux à taper, mais le code qui suit derrière (58 lignes) a demandé 3 jours de travail donc tout est relatif! Par ailleurs, et cela peut s'appliquer à du code Arduino, il est possible de créer **automatiquement** la documentation technique avec des outils comme DOxygen qui va interpréter les mots clés qui commencent par `@` à l'intérieur des commentaires.
Donc s'astreindre à mettre des commentaires normalisés en en-tête de fonction peut faire gagner énormément de temps en plus de la clarté que cela procure, car chaque fonction peut-être vue comme un contrat: *voilà ce que je fais, dans quelles conditions et ce dont j'ai besoin.*
C'est, bien sûr, démesuré dans le cadre des projets de ce MOOC, mais si un jour vous vous lancez dans la réalisation d'une fraiseuse numérique…

IV.2.1.0.5. Que mettre dans un commentaire (pour du code) ?

Nous avons vu que cela dépendait de l'audience concernée par la lecture du code aussi nous allons essayer à chaque fois de donner des exemples dans différents contextes.

```
i ++; // on incrémente i
```

Le commentaire paraphrase le code, si l'audience connaît l'opérateur `++` l'intérêt est nul. Si l'opérateur `++` n'est pas connu ça apporte une information mais dans ce cas on préferera probablement:

```
i ++; // i ++ est équivalent à i = i + 1
```

Qui contient plus de sémantique et qui oblige le lecteur à se concentrer pour assimiler cette nouvelle information.
Si maintenant `i` n'est pas simplement une «vulgaire variable de boucle» mais sert à trouver le premier emplacement non vide dans un tableau il est peut-être préférable: de changer son nom et d'expliquer ce que l'on fait.

```
idx ++; // on va tester si l'emplacement suivant est non vide
```

`idx` se comprend comme la contraction du mot index et le commentaire indique à quel endroit dans l'exécution de l'algorithme nous en sommes (algorithme qui a bien sûr été décrit préalablement en en-tête de fonction ou de programme, n'est-ce pas ?).
Autre cas de figure: une construction de code parfaitement valide mais peu usuelle qui peut être difficile à déchiffrer, même pour un développeur qui n'est pas débutant (que Glenn me pardonne mais dans ce cas précis j'ai à utiliser du code un peu *tricky*):

```
flag ^= 1;
```

Ça ira peut-être mieux avec un commentaire?

```
flag ^= 1; // inversion : 0 -> 1 ou 1 -> 0 via un ou exclusif
```

Pour terminer cette partie voici un exemple complet avec du code simple utilement commenté:

```
/*
  int fahr2celsius (int fahrenheit)

  Convertit des degrés Fahrenheit en degrés Celsius.

  Paramètre : fahrenheit la température en degrés Fahrenheit
  Sortie : la valeur convertie en degrés Celsius

  Notes :
   - le résultat est un entier et nous utilisons des entiers
   pour le calcul donc la valeur est arrondie et approximative
   - si le résultat conduit à une température plus basse
   que le zéro absolu nous retournons le zéro absolu soit
```

```
14    -273°C
15
16   La formule permettant la conversion °F -> °C est la suivante :
17
18   °C = (°F - 32) / 1,8
19
20   Toutefois, comme nous travaillons avec des entiers et souhaitons
21   minimiser les approximations nous allons réaliser les opérations
22   suivantes dans cet ordre :
23
24   °C = ((°F - 32) * 5) / 9
25
26   Résultats attendus :
27
28   0 °F    -> -17 °C
29   32 °F   -> 0 °C
30   50 °F   -> 10 °C
31   212 °F  -> 100 °C
32   -458 °F -> -272 °C
33   -475 °F -> -273 °C
34
35   Documentation utilisée : https://fr.wikipedia.org/wiki/Fahrenheit
36 */
37
38 int fahr2celsius (int fahrenheit)
39 {
40   int temp;                // résultat de la conversion
41
42   temp = fahrenheit - 32; // on retranche le décalage
43   temp = (temp * 5) / 9;  // on s'assure que la multiplication
44                           // est faite avant la division pour
45                           // limiter la perte de précision
46
47   if (temp < -273)        // en dessous du zéro absolu ?
48       temp = -273;        // il faisait un peu (trop) froid !
49
50   return temp;
51 }
```

IV.2.1.0.6. En résumé

Commenter *bien* est un exercice difficile et demande de la pratique et des lecteurs. Cela peut sembler curieux mais quand on a l'habitude l'essentiel du travail intelligent de programmation se fait pendant que l'on écrit les commentaires, le reste, dans le jargon, ça s'appelle «*pisser de la ligne*».

IV.2.2. Programmation structurée (programmation avancée)

IV.2.2.1. Programmation structurée

IV.2.2.1.1. De l'utilité de la programmation structurée

Dans les premiers cours de ce MOOC nous avons appris à donner des instructions séquentielles à la carte Arduino. Toutefois, il est facile d'imaginer que lorsque le problème est d'une nature complexe et nécessite beaucoup d'instructions pour sa résolution, l'approche séquentielle (ou linéaire) n'est plus satisfaisante.
Heureusement, les langages informatiques ainsi que les processeurs permettent depuis quasiment le début de faire de la programmation structurée qui facilite, en particulier, la lisibilité et la compréhension, la maintenance du code et le travail à plusieurs sur un projet.

IV.2.2.1.2. C'est quoi ?

Il y a deux grands axes:
- *au niveau macroscopique*: la modularité, c'est-à-dire découper un gros morceau en plus petit morceaux,
- *au niveau microscopique*: le contrôle du flux d'instructions.

IV.2.2.1.3. Modularité

Pour l'essentiel, et dans le cadre de ce MOOC, il y a deux leviers:

IV.2.2.1.3.1. La fonction Une fonction permet d'encapsuler du code avec une petite latitude de paramétrage. L'intérêt est que cela évite de dupliquer des instructions et qu'en découpant le code en série de fonctions, celui-ci devient plus clair et plus compréhensible.
Une fonction peut accepter des paramètres en entrée et renvoyer **une** valeur en sortie. Aussi bien les paramètres que la valeur retournée sont facultatifs. La déclaration *générique* d'une fonction se fait comme ceci:

```
1 sortie nom (paramètre_1 arg1, paramètre_2 arg2, ..., paramètre_n
      argn)
2 {
3   corps de la fonction
4 }
```

où `sortie` et `paramètre_*` sont des types. Nous allons prendre comme exemple une fonction qui allume une LED pendant une durée exprimée en millisecondes. Cette fonction ne retournera pas de valeur mais prendra deux paramètres: la broche où la LED est raccordée et la durée pendant laquelle la LED devra être allumée. Cela se traduit de la façon suivante:

```
1 void lightLED (int pin, int duration)
2 {
3   digitalWrite (pin, HIGH);
4   delay (duration);
5   digitalWrite (pin, LOW);
```

```
}
```

Si on reprend le TP n°2 mais en utilisant cette nouvelle fonction, `loop()` se réduit à:

```
void loop ()
{
  lightLED (led_verte, 3000);
  lightLED (led_orange, 1000);
  lightLED (led_rouge, 3000);
}
```

Même avec cet exemple très simple l'intérêt est assez évident et immédiat.
Pour retourner une valeur l'instruction return est utilisée, par exemple pour une fonction qui réalise une addition:

```
int add (int a, int b)
{
  return a + b;
}
```

IV.2.2.1.3.2. La bibliothèque Une bibliothèque (mais c'est souvent - à tort - le mot librairie qui est utilisé), peut être vue comme un module contenant une collection de fonctions spécifiquement dédiées à une tâche (par exemple: piloter un écran à cristaux liquides, commander un moteur pas-à-pas, ...),
Nous avons déjà utilisé une bibliothèque (celle qui pilote le port USB pour afficher des valeurs sur l'écran de l'ordinateur et qui commence par le préfixe `Serial`).
Nous en resterons là pour les bibliothèques.

IV.2.2.1.4. Contrôle du flux d'instructions

Le flux d'instruction est séquentiel mais des constructions permettent de casser cette séquentialité: les conditionnelles, les cas et les boucles.

IV.2.2.1.4.1. Les conditionnelles Nous avons déjà vu une forme de conditionnelle: l'instruction `if`:

```
if ( *condition est à vrai* )
{
  faire quelque chose
}
else
{
  faire autre chose
}
```

Cette forme de conditionnelle permet de choisir la branche d'instructions à exécuter:

— Si la condition est évaluée à vrai le code contenu dans la branche `if` est exécuté.
— Si elle est évaluée à fausse ce premier bloc de code n'est pas exécuté et si une branche `else` est présente c'est le code de cette branche qui sera exécuté.

Remarque: une condition est vraie dès lors que son évaluation donne une valeur différente de zéro, une valeur égale à zéro correspond à une condition fausse.
Ainsi `if (1)` sera évaluée à vraie et `if (0)` à fausse. On peut utiliser des variables ou des expressions arithmétiques complexes.
Par exemple:

```
1 if ((variable % 2) == 0) ... // si la variable est paire
```

L'opérateur modulo `%` teste le reste de la division. Si la variable est divisible par deux le résultat de l'expression donnera zéro qui comparée à zéro donnera vrai, c'est-à-dire un.
Ou sous une forme plus compacte:

```
1 if (! (variable % 2)) ... // si la variable N'est PAS IMpaire
```

Sous cette forme, plutôt que de faire une comparaison avec zéro nous inversons le résultat: ***si la variable n'est pas impaire*** *(donc expression vraie si la variable est paire). Cela demande un peu d'habitude pour être à l'aise avec ce type de formulation aussi il est préférable d'utiliser les formes explicites pour commencer.*
L'opérateur `!` inverse le résultat de la condition, attention aussi à l'opérateur de comparaison `==` qu'il ne faut pas confondre avec l'opérateur d'affectation `=`.
Remarque: c'est un choix personnel mais je préfère toujours utiliser des parenthèses pour indiquer l'ordre dans lequel les opérations doivent être effectuées.

```
1 int i = 2;
2
3 if (i)
4 {
5   fonction () ;
6   i = i - 1;
7 }
```

Le code ci-dessus permet d'appeler deux fois une fonction, c'est ce que l'on appelle une boucle et comme c'est très utilisé en programmation il existe des instructions spéciales, ce qui permet d'éviter la construction ci-dessus qui n'est pas recommandable. Nous aborderons les boucles un peu plus loin.

IV.2.2.1.4.2. Les cas Les cas permettent une écriture plus légère d'une forme de conditionnelles. C'est beaucoup utilisé avec les automates. Voici la syntaxe:

```
1 switch ( expression )
2 {
3   case ( valeur_1 ) :
```

```
    traitement
    break;

  case ( valeur_2 ) :
    traitement
    break;

  ...

  case ( valeur_n ) :
    traitement
    break;

  default :
    traitement
    break;
}
```

L'expression est évaluée puis en fonction de la valeur obtenue on va se brancher sur le cas qui correspond à cette valeur. Si aucun cas ne correspond et si le cas `default` est présent (c'est optionnel) alors c'est le traitement qui correspond à `default` qui sera exécuté.
Remarque: l'instruction break permet de bien cloisonner les cas, car une fois un cas traité on sort du `switch`. Sans les instructions `break`, si nous avions à traiter le cas `valeur_1` nous exécuterions en fait l'ensemble du code du switch! Cela peut avoir à certaines occasions un intérêt mais avant d'en arriver là n'oubliez pas de mettre `break` après chaque cas!

IV.2.2.1.4.3. Les boucles La forme, à mon avis, la plus simple de boucle est la boucle `while`. While signifie en anglais «tant que». Elle s'utilise de la façon suivante:

```
while ( condition )
{
  faire quelque chose
}
```

Deux constructions «extrêmes» sont possibles: `while(1)` la boucle est exécutée sans fin, `while(0)` la boucle n'est jamais exécutée. Comme avec les conditionnelles *condition* peut être une expression complexe.
Une autre forme de boucle, très proche de la boucle `while` est la boucle `do...while`, ce qui signifie «faire tant que». La façon dont cette boucle est construite *implique* que le code de la boucle sera exécuté au moins une fois; en effet la condition n'est testée qu'à la fin de la boucle. Voici comment elle s'utilise:

```
do
{
  faire quelque chose
} while ( condition );
```

Notez que la syntaxe du langage exige un `;` dans ce cas après le `while`.

Enfin la dernière forme de boucle présentée dans ce document : la boucle `for` ce qui signifie «pour». Bien maniée elle permet des constructions impressionnantes mais nous nous contenterons dans ce document d'un usage simple et sage! Voici sa syntaxe:

```
for (initialisation ; condition ; action)
{
  faire quelque chose
}
```

L'utilisation la plus courante de cette boucle est d'itérer x fois, voici un exemple:

```
for (int i = 0 ; i < 10 ; i ++) ...  // on exécute 10 fois la
    boucle
```

Décomposons ce qui se passe. L'initialisation de la boucle consiste en la déclaration d'une variable `i` initialisée à 0, l'action à chaque fois que l'on passe dans la boucle est d'incrémenter i (la notation `i ++` est équivalente à `i = i + 1`) et enfin la condition pour que la boucle s'exécute est que `i` soit strictement inférieure à `10`.
Mais souvent on comprend mieux visuellement en réécrivant une boucle `for` sous la forme d'une réécriture avec une boucle `while`, voici ce que cela donne:

```
int i = 0 ;              // initialisation

while (i < 10)           // condition
{
  faire quelque chose
  i ++;                  // action
}
```

Enfin et c'est valable pour toutes les boucles décrites dans ce document il existe des instructions spéciales qui permettent de mettre fin à une boucle ou d'éviter de continuer l'exécution du code de la boucle pour passer à l'itération suivante.
L'instruction `break`, que nous avons déjà vue en conjonction avec `switch`, «rompt» la boucle, elle s'utilise comme ceci:

```
while ( condition_1 )
{
  faire quelque chose

  if ( condition_2 )
    break; // on casse la boucle et on sort

  suite de faire quelque chose
}
```

C'est une instruction très utile car cela facilite le traitement d'un cas exceptionnel. La deuxième instruction est `continue` qui donne ordre à la boucle de passer immédiatement à l'itération

suivante:

```
while ( condition_1 )
{
  if ( condition_2 )
      continue;
  faire quelque chose
}
```

Si condition_2 est vraie pour l'itération courante `faire quelque chose` ne sera pas exécuté, `continue` forcera une nouvelle itération. Cette instruction peut-être très utile pour gérer des cas particuliers.
J'espère qu'avec ces quelques «munitions» vous serez armés pour faire du beau code bien structuré.

IV.2.3. Arduino Thérémine et Processing (programmation avancée)

Le TP sur le thérémine semblait une bonne occasion de tester la liaison Arduino Processing afin de faire varier un son géré par Processing à partir du niveau de lumière détecté par Arduino. Sans aucune expérience sur Processing cela semblait difficile mais à partir de deux exemples fournis dans l'environnement Processing, l'objectif semblait atteignable.

IV.2.3.0.0.1. Elément positifs : Processing s'installe facilement (mais le temps d'installation est plus grand), l'éditeur est très semblable à celui d'Arduino, la programmation utilise un langage identique et la forme générale des programmes est également très proche d'Arduino.
Cahier des charges:
À partir de deux LDR mesurant le niveau de lumière (main droite et main gauche pour faire varier la luminosité sur les LDR) nous désirons faire varier la fréquence et l'amplitude d'un son produit par la carte son du PC.

IV.2.3.0.0.2. Plan de travail :

- Faire générer alternativement deux valeurs numériques par Arduino, l'une représentative du niveau de lumière sur la LDR de droite l'autre représentative de la LDR de gauche.
- Communiquer ces deux valeurs à Processing en utilisant la liaison série.
- Implanter dans Processing une lecture de ces deux valeurs et les utiliser pour générer le son, cette fonctionnalité étant prévue dans processing (voir l'exemple fourni dans «File/Exemples/Librairies/minim/SynthesizeSound»).

?

Première question: comment échanger des données entre Arduino et Processing?
Deuxième question: comment modifier les caractéristiques du son produit par Processing à partir de ces deux valeurs?
Troisième question liée à la seconde: quelles sont les contraintes de la programmation dans Processing?

Reponse à la question 3

Tous les exemples commencent par inclure une ou plusieurs bibliothèques spécifiques aux éléments à mettre en œuvre (liaison série, son, images, vidéo …).
Pour les déclarations et les initialisations (`setup()`), identique à Arduino.
Pour `loop()`, il semble que cette procédure soit systématiquement remplacée par `draw()` qui a la même fonction mais doit certainement inclure l'appel à d'autres procédures pouvant être déclarées dans les programmes mais sans jamais être appelées. Il semble possible de remplacer `draw()` par `loop()` dans plusieurs programmes testés mais il est nécessaire de rajouter dans cette boucle les appels à toutes les procédures déclarées dans le programme.

Réponse à la question 1

La lecture par processing de valeurs numériques fournies par Arduino est relativement simple, l'exemple ci-dessous permet de vérifier cette fonctionnalité:

```
/*Programme pour Arduino
  Le programme place sur la sortie série des valeurs de 0 à 255 et
  de 255 à 0 avec une pause de 10ms entre chaque nombre.

  Ce programme n'est pas commenté car il est simple
  avec des instructions connues
*/

int val=0;

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

void loop()
{
    for (int i=0;i<255;i++)
    {
    Serial.write(i);
    delay(10);
    }
    for (int i=255;i>0;i--)
    {
    Serial.write(i);
    delay(10);
    }
}
```

```
/*Programme pour Processing
  Le programme fait changer la couleur d'un rectangle affiché dans
    une fenêtre.
  Les couleurs du noir au blanc sont définies par la valeur
  reçue sur la liaison série
```

```
5   Ce programme est largement inspiré de l'exemple
      "serial/simpleread"
6 */
7
8 import processing.serial.*; // importation de la bibliothéque
      serial
9 Serial myPort; // Création d'un l'objet Serial class
10 int val; // Donnée en provenance de la liaison série
11
12 void setup()
13 {
14   // taille de la fenêtre d'affichage
15   size(200, 200);
16   // déclaration du port de la liaison série (ne pas modifier même
       pour CM3)
17   String portName = Serial.list()[0];
18   // déclaration des caractéristiques de la liaison série
19   myPort = new Serial(this, portName, 9600);
20 }
21
22 void draw()
23 {
24   if ( myPort.available() > 0) // si une donnée est présente
25   {
26     // lecture de la donnée
27     val = myPort.read();
28   }
29   // mise à blanc du fond
30   background(255);
31   // choix de la couleur du rectangle en fonction de la valeur lue
32   fill(val);
33   // tracé du rectangle
34   rect(50, 50, 100, 100);
35 }
```

IV.2.3.0.0.3. Utilisation des deux programmes 1°) Charger le programme Arduino et l'exécuter.
2°) Charger le programme Processing et l'exécuter
Si tout est OK, la fenêtre s'ouvre et le rectangle change de couleur toutes les 10ms (2mn pour passer progressivement du noir au blanc).

Gérard04

IV.2.4. L'indentation

Beaucoup de participants au MOOC l'ont constaté: les codes sont parfois difficilement lisibles. Il existe une manière de présenter son code de façon à le rendre compréhensible par quelqu'un d'autre: l'indentation.
Il ne s'agit que de forme: cela n'a aucune incidence sur le fonctionnement du programme.

Par exemple, ce code est valide mais illisible par un humain:

```
const unsigned char a = 0; unsigned char b = a; unsigned int c; if
    ((a + b) == c) a = a + 1; else a = 2;
```

Le même, correctement indenté:

```
  const unsigned char a = 0;
  unsigned char b = a;
  unsigned int c;
  if ((a + b) == c)
    a = a + 1;
  else
    a = 2;
```

Cette manière de décaler le texte, de faire ainsi apparaître sa structure, c'est l'indentation. Les principes sont simples:

1. Quand on rentre dans une section de code, on décale tout le contenu de cette section d'une tabulation.
2. Quand on sort d'une section de code, on retire une tabulation pour la suite.

Ainsi, voici un code correctement indenté:

```
int a = 0;
int b = 0;
for (a = 0; a < 10; a++)
  if (a < 5)
    b = b + 1;
  else
    b = b + 2;
```

Pour identifier facilement les sections, il est recommandé d'utiliser les accolades `{}`. Le code devient alors :

```
int a = 0;
int b = 0;
for (a = 0; a < 10; a++) {
  if (a < 5) {
    b = b + 1;
  }
  else {
    b = b + 2;
  }
}
```

Le problème, c'est que l'on se retrouve avec une série d'accolades fermantes sans savoir ce qu'elles ferment. Il est alors très utile de commenter son code… CQFD 🍊

```
int a = 0; // Variable pour le FOR
int b = 0; // Variable résultat
for (a = 0; a < 10; a++) { // Boucle de traitement
  if (a < 5) { // Test sur la valeur de a
    b = b + 1;
  } // Fin si a < 5
  else { // a >= 5
    b = b + 2;
  } // Fin si a >= 5
} // Fin boucle de traitement
```

C'est tout de suite plus clair. Et d'expérience, c'est indispensable quand on reprend son propre code après quelques jours.
Encore une fois, toutes ces astuces ne constituent que de la forme et n'affectent pas le bon fonctionnement du programme. Mais cela permet de corriger très vite du code en identifiant rapidement les sections à problème.
Il y a cependant quelques exceptions à cette règle esthétique: les langages Python et Cobol par exemple, qui font de l'indentation du code un élément indispensable, tellement indispensable qu'on peut même se passer d'accolades! (mais pas des commentaires…)
En Python, un code mal indenté dysfonctionnera à coup sûr.

TPE

IV.2.5. Les tableaux

Un tableau est une série de données stockées consécutivement. Dit comme cela c'est peu vendeur et pourtant le tableau est un outil très utile en programmation, en particulier lorsque l'on fait des automates (des feux de circulation au hasard).
Plutôt que de multiplier les variables, comme malheureusement on le trouve fréquemment dans du code, typiquement:

```
int etat1 = 0 ;
int etat2 = 0 ;
int etat3 = 0 ;
```

Il existe une autre méthode qui consiste à regrouper des variables dans une même zone. Cette zone sera accessible comme une variable par son nom, mais vient s'ajouter un index précisant l'élément du tableau sur lequel on désire travailler. Un tableau se déclare de cette façon dans un programme:

```
int etats [3] = { 0, 0, 0 } ;
```

L'«équivalence conceptuelle» est assez simple entre les deux formes, mais celle-ci n'apporte rien de concret ou de directement perceptible:

```
etat1 = etats [0] ;
etat2 = etats [1] ;
etat3 = etats [2] ;
```

Dans la plupart des langages de programmation l'index du premier élément est à zéro et le dernier vaut *n-1* pour *n* éléments. Cela demande un peu de gymnastique mais nous allons voir que c'est très pratique car c'est de l'adressage indexé. Au lieu d'écrire par exemple:

```
if (etat1 == 1)
  ...
else if (etat2 == 1)
  ...
else if (etat3 == 1)
  ...
else
  ... // Ooppps : c'était pas prévu
```

On va utiliser cette forme:

```
if (etats [index] == 1)
  ...
```

En maintenant une variable `index` qui permet de savoir où l'on se trouve et sur quoi on travaille, et donc de réduire la combinatoire qui existe lorsque l'on travaille avec des variables indépendantes et les risques d'erreurs associés.
Utilisés conjointement avec une boucle, les tableaux permettent de réaliser des initialisations quelque peu fastidieuses de façon économique, par exemple:

```
# define PINS 3

int outputs [PINS] = { 1, 2, 3 } ;  // broches en sortie
int inputs [PINS] = { 4, 5, 6 } ; // broches en entrée


// Initialise les entrées / sorties du microcontrôleur

void setup ()
{
  int i ;

  for (i = 0 ; i < PINS ; i ++)
  {
    pinMode (outputs [i], OUTPUT) ;
    pinMode (inputs [i], INPUT) ;
  }
}
```

On peut mettre tout ce que l'on veut dans un tableau dès lors qu'il y a assez de mémoire et que les éléments sont du même type. Par exemple des caractères:

```
1 char message [5] = { 'M', 'O', 'O', 'C', 0 } ;
```

Cette suite de caractères est terminée par un zéro qui n'est pas le caractère représentant le zéro (sinon il aurait été placé entre apostrophes), mais la valeur zéro qui va nous servir de marqueur de fin de message.
Utilisé avec une boucle, voici une façon d'utiliser le tableau (on suppose qu'une fonction magique `putc ()` permet d'afficher un caractère sur un dispositif quelconque):

```
1  void loop ()
2  {
3    int idx = 0 ;
4
5    while (message [idx])
6    {
7      putc (message [idx]) ;
8      idx = idx + 1 ;
9    }
10
11   for (;;)    // forever : boucle infinie
12     ;
13 }
```

La décomposition sous une forme «plus littéraire» de la fonction est comme suit:

- tant que l'on ne tombe pas sur un élément du tableau ayant comme *valeur* zéro:
  - on lit l'élément du tableau pointé par l'index courant (`idx`),
  - on affiche ce caractère,
  - on passe à l'index suivant,
- on arrête d'afficher (en partant volontairement en boucle infinie).

Bien évidemment si l'on change le contenu du tableau `message` (en prenant soin tout de même de laisser le zéro qui sert de marqueur de fin) il n'y a absolument rien à modifier dans le code de `loop ()`. C'est intéressant car c'est *presque* changer du code mais en ne touchant qu'à des données…
Maintenant que nous avons un nouvel outil à notre disposition voyons comment l'utiliser. On reprend le TP n°2 (le feu tricolore: 3 secondes au vert, une seconde à l'orange et trois secondes au rouge). C'est un cycle de trois états, et chaque état est caractérisé par un couple couleur et durée. Ça ressemble à des données stockées consécutivement sous la forme d'un tableau contenant trois couples couleur et durée, la couleur ayant un index pair (0, 2 et 4) et la durée un index impair (1, 3 et 5) dans ce tableau:

```
1 // Six éléments numérotés de 0 à 5
2
3 int feu [6] =
4 {
```

```
  // état 0 (vert, 3 s)
  1,                      // broche led verte, index 0
  3000,                   // durée en ms, index 1

  // état 1 (orange, 1 s)
  2,                      // broche led orange, index 2
  1000,                   // durée, index 3

  // état 2 (rouge, 3 s)
  3,                      // broche led rouge, index 4
  3000                    // durée, index 5
} ;
```

Le programme est quasiment écrit il ne reste plus «qu'à connecter» les données avec un peu de code:

```
// On paramètre les broches dont on a besoin en output

void setup ()
{
  int idx = 0 ;

  // Les numéros de broches sont aux index 0, 2 et 4
  for (idx = 0 ; idx < 3 ; idx = idx + 1)
    pinMode (etat_feu [idx * 2], OUTPUT) ;
}


int state = 0 ; // indique l'état dans lequel nous sommes
                // en déplacant un curseur (index) sur le
                // tableau

void loop ()
{
  digitalWrite (feu [state], HIGH) ;  // la broche
  delay (feu [state + 1]) ;           // le délai
  digitalWrite (feu [state], LOW) ;

  // On se prépare pour l'état suivant
  state = state + 2 ;                 // couple suivant

  // Si l'on est arrivé en fin de tableau, on recommence depuis le
      début
  if (state > 4)
    state = 0 ;
}
```

Nous avons réalisé un automate qui se contente de boucler à partir des données du tableau et qui en fin de tableau recommence au début, ce n'est pas très spectaculaire, en revanche voyons comment on peut utiliser cette technique sur un problème nettement plus complexe. Vous vous

souvenez du feu piéton avec bouton?
En temps normal le feu piéton reste au rouge et le feu voiture enchaîne le cycle vert, orange, rouge habituel. Si un piéton, *alors que le feu voiture est au vert*, appuie sur le bouton alors le feu voiture enchaîne un cycle vert, orange, rouge mais en ajoutant deux secondes au feu rouge et en faisant passer le feu piéton au vert quand le feu voiture sera au rouge.
Donc un état se résume à la couleur du feu voiture, la couleur du feu piéton et le délai *mais* il y a deux cycles: le cycle normal et le cycle bouton appuyé.

```
// les états de notre système

int feux [21] =
{
  // Pivot
  // -----

  // état 0, vert, rouge, 3 s
  1,              // led verte feu voiture, index 0
  4,              // led rouge feu piéton, index 1
  3000,           // durée en ms, index 2

  // Cycle normal
  // ------------

  // état 1, orange, rouge, 1 s
  2,              // led orange feu voiture, index 3
  4,              // led rouge feu piéton, index 4
  3000,           // durée en ms, index 5

  // état 2, rouge, rouge, 3 s
  3,              // led rouge feu voiture, index 6
  4,              // led rouge feu piéton, index 7
  3000,           // durée en ms, index 8

  // état 3 : marqueur de fin
  0, 0, 0,        // index 9, 10 et 11

  // Cycle bouton appuyé
  // -------------------

  // état 4, orange, rouge, 1 s
  2,              // led orange feu voiture, index 12
  4,              // led rouge feu piéton, index 13
  3000,           // durée en ms, index 14

  // état 5, rouge, vert, 5 s
  3,              // led rouge feu voiture, index 15
  5,              // led verte feu piéton, index 16
  5000            // durée en ms, index 17

```

```
42    // état 6, marqueur de fin
43    0, 0, 0          // index 18, 19 et 20
44  } ;
45
46  # define    START_NORMAL_CYCLE      0   // état de départ cycle
        normal
47  # define    START_CROSSING_CYCLE    12  // état de départ cycle
        bouton appuyé
```

On passe sous silence la partie `setup()` qui n'est guère passionnante exceptée la mise en place d'une interruption sur le bouton pour se concentrer sur `loop()`

```
1   int state = 0 ;                 // état courant
2   volatile int crossing = 0 ;     // état du bouton (changé par
        interruption)
3
4
5   // Effectue l'allumage et l'extinction en fonction de l'état
6   void action (int index)
7   {
8       digitalWrite (feux [index], HIGH) ;     // voiture
9       digitalWrite (feux [index+1], HIGH) ;   // piéton
10      delay (feux [index+2]) ;                // délai
11      digitalWrite (feux [index], LOW) ;      // voiture
12      digitalWrite (feux [index+1], LOW) ;    // piéton
13  }
14
15  void loop ()
16  {
17
18    action (state) ;                  // on traite l'état courant
19
20    if (state == START_NORMAL_CYCLE)  // feu voiture au vert ?
21    {
22      if (crossing)                   // tester l'état du bouton
23        state = START_CROSSING_CYCLE ;  // passer au cycle bouton
              appuyé
24    }
25    else
26      state = state + 3 ;             // passer au triplet suivant
27                                      // (= état suivant dans le
                                            cycle)
28
29    crossing = 0 ;                    // désarmer le bouton
30
31    if (feux [state] == 0)            // marqueur de fin atteint ?
32      state = START_NORMAL_CYCLE ;    // on revient au début du
            cycle
33  }
```

Le code est concis car la logique est très simple:
— traiter l'état courant,
— déterminer l'état suivant,
— reboucler.
En fonction de l'état du bouton quand le feu voiture est vert, nous choisissons le cycle à exécuter, nous l'exécutons état par état jusqu'à tomber sur le marqueur de fin puis revenons à l'état de départ. L'état zéro (feu vert pour les voitures est un pivot car il est commun aux deux cycles: c'est à l'issue de l'état zéro que l'on va décider, en fonction du bouton, si on passe à l'état 1 (cycle normal) ou à l'état 4 (cycle bouton appuyé).
Toute la difficulté consiste à concevoir le tableau. Faire un schéma sous la forme d'un graphe est souvent une bonne idée de façon à valider le modèle et à s'assurer qu'aucun cas n'a été oublié.
Il existe beaucoup de variantes de cette technique mais précisons que cela ne sert pas qu'à faire des automates pour des feux; on peut citer la réalisation de protocoles de communication, des compilateurs, etc.
L'inconvénient est que si le code est simple, les données le sont moins. Or, sur un automate un peu compliqué avec un tableau volumineux, une modification peut être délicate à faire et exige de la rigueur dans sa réalisation. Néanmoins, dans beaucoup de situations un tableau peut s'avérer très pratique.

IV.2.6. Les pointeurs (programmation avancée)

Glenn a introduit en catimini lors des cours de la semaine 10 le pointeur. C'est généralement le sujet qui rebute le plus dans l'apprentissage de la programmation. Certains langages de programmation empêchent son utilisation (Java, C#…), d'autres la limitent (Pascal). La seule évocation du mot «pointeur» fait frémir un responsable d'assurance qualité. Seul le `goto` *peut prétendre avoir une réputation aussi sulfureuse, mais on ne va pas s'arrêter à ces rumeurs, découvrons la bête de l'intérieur comme tout bon bidouilleur le ferait.*

> Par ma foi! Il y a plus de quarante ans que je dis de la prose sans que j'en susse rien, et je vous suis le plus obligé du monde de m'avoir appris cela.
>
> *Molière, le bourgeois gentilhomme*

Eh oui! Que vous le vouliez ou non, indirectement, vous faites usage de pointeurs en écrivant ou en exécutant un programme. C'est tout simplement essentiel et indispensable en informatique.
Un pointeur est une variable qui a pour valeur une adresse vers la mémoire.
C'est une indirection en ce sens que l'on ne travaille pas *directement* sur une valeur mais sur l'emplacement où est stockée cette valeur. Dans le cas classique d'une variable comme:

```
1 int a = 1 ;
```

On manipule implicitement un pointeur puisque ce code signifie «range à *l'adresse* désignée par `a` la valeur `1`», simplement tout cela est masqué: on voit `a` comme une valeur et non pas comme étant l'emplacement où cette valeur est stockée. C'est beaucoup plus facile et naturel à manipuler sous cette forme aussi pour *vraiment* travailler avec un pointeur il faut le demander explicitement au compilateur.
Pour désigner une adresse, et non pas une valeur, il faut utiliser l'opérateur `&`, ainsi:

```
& a
```

ne désignera pas la valeur `1` mais l'adresse en mémoire où est stockée cette valeur de type `int` aussi notre pointeur `p` sera déclaré comme un pointeur sur un type `int` pour notre variable `a` comme ceci:

```
int * p = & a ;
```

L'astérisque indique qu'il y a une indirection. `p`, dans cet exemple, contient l'adresse de `a` mais à travers `p` (noté `* p`) on peut récupérer la valeur contenue dans `a`. Pour résumer:

```
// Déclarations
int a = 1 ;          // la variable
int * p = & a ;      // un pointeur sur cette variable

// Utilisations
if (p == & a)        // vrai : p contient l'adresse de a
...
if (* p == 1)        // vrai : * p revient à lire le contenu de a,
    c'est-à-dire 1
```

Cela semble un peu compliqué pour pas grand chose mais nous avons créé de l'abstraction et celle-ci, bien utilisée, est un fantastique levier en programmation comme nous allons le voir.

IV.2.6.0.1. Quelques exemples

Nous avons vu qu'une fonction ne pouvait retourner qu'au plus une valeur, c'est parfois limitant. Par exemple, s'il est logique de savoir si une fonction a bien réussi à s'exécuter, on aime bien savoir *a contrario* pourquoi tel n'a pas été le cas en disposant de plus d'informations. Le seul problème c'est qu'on ne peut retourner qu'une valeur. Vraiment?

```
int dumb_function (int * error_code)
{
  if (error_code != NULL)
    * error_code = -1 ;

  return 0 ;
}
```

Décomposons le fonctionnement de cette fonction qui n'a aucun intérêt pratique:

```
if (error_code != NULL)
```

Nous vérifions que le pointeur `error_code` contient une adresse mémoire *a priori* valide; la constante `NULL` étant une adresse signifiant «pas d'adresse valide» (généralement cette adresse est égale à zéro).

```
* error_code = -1 ;
```

Nous affectons **à travers** le pointeur `error_code` dans la zone de mémoire de type `int` qu'il désigne la valeur `-1`

```
    return 0 ;
```

La valeur zéro est retournée par la fonction.
En situation:

```
int ok ;
int error ;

ok = dumb_function (& error) ;

if (! ok)
{
  if (error == -1)
  ...
}
...
```

On décompose. Nous invoquons notre fonction en indiquant que le code supplémentaire doit être renvoyé *à travers* la variable error (notation `& error`) et récupérons le résultat de l'exécution de la fonction dans `ok`. Si la fonction retourne zéro nous nous attendons à une information supplémentaire qui sera placée dans notre variable `error` par la fonction, car celle-ci a eu connaissance de *l'adresse* où stocker cette valeur par l'intermédiaire du paramètre sous forme de pointeur `error_code`. Techniquement, notre fonction a donc retourné deux valeurs!
Si vous avez lu l'article Wiki sur les tableaux, vous savez qu'un tableau est une suite consécutive de valeurs, donc une adresse et une longueur pour stocker x objets de type y, par exemple une suite de caractères comme ceci:

```
char msg [5] = { 'M', 'O', 'O', 'C', 0 } ;
```

Rappel: le zéro (valeur) à la fin est un marqueur signalant que la chaîne de caractères est terminée.
On peut s'amuser avec un pointeur:

```
char * m = & msg [0] ;
```

Ça devient compliqué, aussi la règle est de **toujours décomposer**. On va travailler sur une suite de caractères donc le type est `char`, c'est un pointeur donc `*`, et l'adresse de début est le premier élément du tableau, donc `msg [0]` mais préfixé par `&` pour indiquer que c'est l'adresse qui nous intéresse.
À travers ce pointeur `m` nous pouvons accéder au tableau, caractère par caractère: `* m` retournera

ainsi le premier caractère de "MOOC" donc 'M'. Mais comme ce pointeur est une variable nous pouvons faire aussi des opérations arithmétiques dessus comme l'incrémentation `++`. Voyons ce que cela donne sur un exemple complet:

```
int len = 0 ;

while (* m ++)
  len ++ ;
```

C'est cryptique mais rien ne résiste à la décomposition. Il y a toutefois une subtilité qui est que l'opérateur ++ s'applique au pointeur **et non à la valeur pointée** et est *postfixé* (exécuté en dernier) autrement dit la séquence `* m ++` revient aux instructions suivantes:

```
x = * m ;
m = m + 1 ;
```

Là aussi il faut prendre le temps de bien examiner la situation: la première étape est de lire le caractère qui est pointé à l'adresse courante du pointeur, puis ce pointeur est déplacé sur le caractère suivant, c'est-à-dire l'adresse qui est située à un caractère de distance après l'adresse de base de notre tableau - c'est le sens de l'opérateur `+` qui signifie «adresse courante plus un objet de la taille d'un `char`».
Le tout étant dans une boucle `while`, en une ligne de code nous arrivons mine de rien à exprimer «tant que la chaîne de caractères n'est pas terminée…», c'est puissant l'abstraction! Quand le marqueur de fin est rencontré la boucle s'arrête et la variable `len` contient alors la longueur de notre chaîne c'est-à-dire quatre (le marqueur de fin n'est pas comptabilisé).
Si l'on souhaite maintenant copier une chaîne de caractères `src` (*source*) vers un autre emplacement `dst` (*destination*) le code correspondant est d'une concision extrême:

```
while (* dst ++ = * src ++)
  ;
```

Il faut **dé-com-po-ser**! Au ralenti voici la version un peu plus longue du code:

```
while (* src)
{
  * dst = * src ;
  src = src + 1 ;
  dst = dst + 1 ;
}

* dst = 0 ;
```

On récupère le caractère courant de la chaîne source; si celui-ci à la valeur zéro la boucle s'arrête. Le caractère courant pointé par `src` est copié dans `dst` puis on passe au caractère suivant tant pour la source que pour la destination. Enfin, et contrairement à la version courte, il faut penser à ajouter le marqueur de fin à la chaîne de destination lorsque la boucle est terminée.

Beaucoup d'autres utilisations sont possibles, à commencer par les listes chaînées.
Il est bien sûr possible d'utiliser des pointeurs sur des pointeurs, des pointeurs de fonctions, etc. Inutile d'ajouter qu'il faut de l'entraînement car il est facile de s'y perdre dans ces indirections et qu'une erreur conduit généralement à un matraquage de la mémoire en règle qui peut donner des effets curieux et variés très difficilement traçables.
Le pointeur est un outil très puissant mais sa puissance est à la hauteur des dégâts qu'il peut occasionner mal utilisé. Avant d'être *vraiment* à l'aise avec ces objets, il faut des années de pratique et avoir commis beaucoup de bugs, donc patience!

IV.3. Arduino (documentation et exemples)

IV.3.1. Trucs et astuces

#Pense-bête
Voici un petit schéma reprenant les entrées et sorties de l'Arduino UNO. Ça peut être bien pratique:

FIGURE IV.3.1. – Arduino Uno

#Arrêtez: je veux descendre!
La structure d'un Sketch ou programme pour l'Arduino est une procédure `setup()` appelée une fois après l'initialisation (reset) de l'Arduino, puis une `loop()` , c'est-à-dire une boucle, qui est appelée à l'infini...
Parfois il est utile de pouvoir arrêter le programme: à cause d'une erreur, par exemple. Comment faire?
L'Arduino est infatigable: il ne s'arrête jamais. Mais on peut faire en sorte que notre Sketch s'arrête, en le bloquant. Une sorte de clef à mollette coincée dans l'engrenage...
Pour ce faire, on fait appel à une boucle qui ne se termine jamais. Comme ceci:

```
1 do { } while (true);  // arrêter le programme
```

Il n'y a rien entre les accolades `{}`: donc rien ne se passe, et la valeur `true` est ***toujours*** vraie (synonyme de la valeur 1).
L'Arduino va donc se mettre à ne **rien faire de nouveau** (toutes les sorties restent en l'état), jusqu'à ce qu'on appuie sur la touche "Reset" qui relance `setup()` puis `loop()`.
Voici un exemple d'utilisation:

```
1 void loop() {
2   val = analogRead(ma_broche_analogique);
3   if ( val != 0 ) {  // Si valeur = 0 un problème s'est produit
4     faire_quelquechose();
5   }
6   else { // valeur = 0
7     Serial.println("Erreur. J'en ai marre.");
8     do { } while (true);  // grève générale
9   }
```

```
10 }
```

IV.3.1.1. Commandes compilateur

Si vous regardez le Sketch "ArduinomtreWeb", que j'ai publié semaine 10, vous verrez plusieurs lignes qui commencent par `#` (comme pour `#include`) mais qui ne sont pas des `#define` que j'utilise habituellement. Ce sont des *compiler directives*, ou des commandes pour diriger le compilateur.
Il y à deux formes dans ce sketch:

```
1 #define ARDUINO_UNO
2 #define TRACES 1
```

La première forme est utilisée pour définir un nom. Pour le compilateur, le nom existe, ou il n'existe pas. On peut faire, donc:

```
1 #if defined ARDUINO_UNO
2   #define Ether_CS 8 // Broche sur laquelle nous avons branché CS
        (UNO)
3 #else
4   #define Ether_CS 48 // Broche sur laquelle nous avons branché CS
        (MEGA)
5 #endif
```

et aussi

```
1 #if defined ARDUINO_UNO
2   analogReference(INTERNAL);  // UNO
3 #else
4   analogReference(INTERNAL1V1);  // Mega
5 #endif
```

Notez bien que ce sont des directives pour le compilateur: ce ne sont pas des instructions en C proprement dit. Par la déclaration de `#define ARDUINO_UNO` ou `#define ARDUINO_MEGA` au début de mon sketch je peux faire compiler le même sketch correctement pour un Arduino Uno ou un Arduino Mega.
L'autre forme `#define TRACES 1`, elle, attribue une valeur binaire au nom déclaré. C'est très pratique pour supprimer des morceaux de code (les `Serial.print`, par exemple) sans pour autant les perdre pour toujours. Parfois une fonction est difficile à mettre au point: on passe du temps à écrire des commandes pour nous afficher des résultats sur le *serial monitor*, on pense que c'est corrigé, mais pas complètement sûr … Avec les commandes suivantes on peux recompiler le Sketch sans les traces, mais si on a à nouveau besoin on ne change qu'une ligne en haut du sketch! Comme ici:

```
void loop() {
  word len = ether.packetReceive();
  word pos = ether.packetLoop(len);

#if TRACES
  if (len && pos) {
    Serial.print("|");
    for (count = pos; count<len; count++) {
      Serial.print(char(Ethernet::buffer[count]));
    }
  Serial.println("|");
  Serial.print("Longueur message : ");
  Serial.println(len);
  }

  if (pos) {
    Serial.print("Pos :");
    Serial.println(pos);
  }
#endif
}
```

Si `#define TRACES 0` est déclaré dans le Sketch, le compilateur va compiler ceci:

```
void loop() {
  word len = ether.packetReceive();
  word pos = ether.packetLoop(len);
}
```

Tout ce qui se trouve entre le `#if` et le `#endif` est traité comme si nous l'avions mis en commentaire et donc n'est pas compilé.

IV.3.1.2. Mise en forme du texte

Les fonctions standards `lcd.print` et `Serial.print` n'ont pas beaucoup de moyens pour gérer la mise en forme d'un texte. Il existe une bibliothèque `stdio.h` qui nous offre des fonctions très puissantes, comme `sprintf()` - mais cette bibliothèque est **très** gourmande en ressources - et souvent nous mange presque toute la mémoire RAM de notre Arduino.
Vous avez vu avec le montage "Arduinoweb" qu'il est utile de pouvoir formater du texte sans pour autant faire appel à la grosse cavalerie de `stdio.h`.
C'est pour cela que j'ai déniché une autre bibliothèque, plus petite et un peu moins performant - mais aussi moins gourmande: `stdarg.h`. Avec cette bibliothèque nous avons accès à une fonction `vsnprintf()` qui sert à la mise en forme d'un message avant son impression, affichage ou, comme dans l'exemple "Ardruinoweb" inclusion dans une page *html* ou *php*.
Voici comment s'en servir:

IV.3.1.2.1. Déclaration

Toute au début du sketch il faut déclarer la bibliothèque avec

```
#include <stdarg.h>
```

IV.3.1.2.2. Variables

Pour fonctionner, `vsnprintf()` à besoin d'une zone tampon en mémoire. Il faut dimensionner cette zone (on l'appel "buffer" en anglais) avec soin: trop grande et nous n'aurions plus de place pour nos variables ; trop petite et la fonction va se mettre à écraser des données... Essayez de calculer le nombre maximum de symboles que nous avons besoin de formater - avec un minimum de texte *statique* (qui ne change pas). N'oubliez pas la règle: il faut toujours ajouter un octet de plus que le nombre de symboles, pour stocker la *fin de chaîne* `'\0'`.
Calculs faits, nous pouvons créer notre zone tampon:

```
char formatString[33];
```

Le nom n'est pas important mais comme toujours il faut mieux lui donner un nom qui nous explique ce qu'il fait ! D'habitude on déclare cette variable en tant que ***globale*** (c'est à dire avant `setup()`) pour qu'elle soit accessible par tout le monde.

IV.3.1.2.3. La fonction `vsnprintf()`

Je trouve que la meilleure façon de déclarer cette fonction est de "l'encapsuler" dans une fonction complète. Voici comment je fais:

```
void format(char *fmt, ... ){
  va_list args;
  va_start (args, fmt );  // allocation d'espace en mémoire
  vsnprintf(formatString, 32, fmt, args);  // formatage du message
  va_end (args);  // libération des ressources.
}
```

Pour comprendre comment ça marche, regardons toute de suite comment s'en servir, avec un appel à notre fonction:

```
format("Valeur : %04d, environ %02d°C", v1, v2);
```

L'idée ici c'est de 'formater' les deux valeurs v1 et v2 et de les inclure dans le message. Je vais expliquer les hiéroglyphes après les `%` un peu plus loin.
La première chose bizarre est que, dans la déclaration de notre fonction `format()` nous n'avons pas un nombre fixe d'arguments. C'est un peu normal car on ne sait pas d'avance combien de valeurs on va vouloir formater. Le premier argument `char *fmt` nous fait passer un *pointeur* vers notre texte de départ, au lieu de passer le texte entier. Les points de suspension fait réserver de la place pour 1 ou plus d'arguments additionnels.

La ligne suivante, `va_list args` déclare une variable `args` de type `va_list` (`va_list` est déclaré dans la bibliothèque): c'est cette variable qui va contenir les arguments que nous passerons à la fonction.
`va_start (args, fmt );` initialise l'environnement pour `vsnprintf()`, récupère nos arguments pour les stocker dans `args` et prépare le boulot pour la suite.
`vsnprintf(formatString, 32, fmt, args);` Pour terminer le boulot. Le "32" c'est le nombre de symboles maximum à écrire dans notre zone tampon. Si le message résultant est plus long, il est 'tronqué': les symboles de trop sont jetés. Il ne nous reste qu'à faire le ménage (libérer les ressources attribués avec `va_start`): `va_end (args);` Voilà.
Si la valeur de *v1* était 1023 et celle de *v2* était 26, le message résultant serait

```
Valeur : 1023, environ 26°C
```

IV.3.1.2.4. Les hiéroglyphes

C'est la partie la plus intéressante!
Chaque fois que `vsnprintf()` trouve un `%` dans le message, il cherche à décoder des commandes de formatage, lui indiquant la forme que nous voulions. D'abord on lui indique le nombre d'emplacements à réserver, et optionnellement le nombre de places décimales pour les nombres de type `float`.
Exemples:

```
2       2 caractères
02      2 caractères, "0"s de remplissage
3.02    3 digits dont deux digits après la virgule
```

Et après vient la type de résultat. Il faut que les *types* du format demandé et l'argument correspondent (on ne peut pas formater une variable de type `int` en `%2.03f`, par exemple). Voici quelques exemples:

```
%i ou %d    int
%x ou %X    int exprimé en hexadécimal (minuscule/majuscule)
%c          char
%f          float
%e ou %E    float en format SCI (1.42e34)
%s          string
```

Et quelques exemples plus concrets :

```
{
  int a,b;
  float c,d;
  a = 15;
  b = a / 2;
  format("%d",b);
```

```
  Serial.println(formatString);
  format("%3d",b);
  Serial.println(formatString);
  format("%03d",b);
  Serial.println(formatString);
  c = 15.3;
  d = c / 3;
  format("%3.2f",d);
  Serial.println(formatString);
}
```

Donnera:

```
7
  7
007
5.10
```

Pour plus d'exemples et pour plus d'informations il faut se référer à un livre, un cours, un tutoriel de programmation en C.

IV.3.2. Le bouton rotatif

##Le Bouton Dompté
Voici le bouton rotatif, Parfois appelé Encodeur rotatif. Quelques images:

FIGURE IV.3.2. – L'encodeur vue de profil

FIGURE IV.3.3. – L'encodeur vue de dos

et une forme de symbole

FIGURE IV.3.4. – Un symbole électronique de l'encodeur

Objet d'apparence simple, pas très cher (moins de 3€ pour un modèle simple avec contacteur), sa simplicité est rapidement oubliée quand il s'agit de faire quelque chose avec. Voici le problème:

FIGURE IV.3.5. – Chronogramme de l'encodeur

(image: merci Arduino Playground. Article avec beaucoup d'infos ici)
Les deux broches A et B font contact de temps en temps avec la connexion commune. Leur séquence est décalée d'un quart de cycle. Le "challenge" est de décoder le sens de rotation et le nombre de cycles parcourus chaque fois qu'on tourne le bouton. C'est un défi car les impulsions peuvent êtres rapides si on tourne le bouton avec un peu de d'énergie!
Le deuxième problème avec certains boutons c'est qu'ils sont très bruyants. Non, ils ne chantent pas - ils génèrent beaucoup de parasites dus à leur conception mécanique (effet de rebonds). Nous avons déjà vu que les condensateurs nous aident à diminuer les effets des rebonds de contacts, regardons ensemble comment résoudre le problème informatique.
Une solution, parmi des dizaines, relativement simple est abordée dans l'article d'Arduino Playground, solution que je vais tenter d'expliquer ici.

IV.3.2.0.1. Sens de rotation

Si on regarde l'image, et qu'on ne se soucie que de la broche A, et que les flancs montants (endroit où le signal va de 0V vers 5V): en tournant le bouton vers la droite, on peut remarquer qu'à chaque flanc montant de A, le signal B est stable et à 5V. Si on tourne dans l'autre sens (de droite à gauche), à chaque flanc montant de A, le signal B est stable et à 0V. Pour compter le nombre de cycles, il faut compter le nombre de flancs montants.
Cette solution est simple et peut déjà faire l'affaire pour nous. Reste le problème de comment détecter les mouvements pendant le déroulement de notre Sketch Arduino.

IV.3.2.0.2. Désolé de vous interrompre

Si vous avez suivi le module sur les interruptions (semaine 9) vous avez sûrement deviné comment faire. Si on raccorde la broches A à une entrée INT sur l'Arduino, on peut détecter les flancs montants avec la déclaration:

```
1 attachInterrupt(0, doSomething, RISING);
```

Le `0` veut dire qu'on veux utiliser INT0 (c'est la broche 2 de l'Arduino); `doSomething` est le nom de notre fonction de traitement d'interruption; `RISING` veut dire interruption sur flancs montants.
La broche B est connectée à n'importe quelle entrée numérique: pas besoin d'interruption car on va juste vérifier son état au moment où détecte le flanc montant A.
Voici à quoi peut ressembler notre fonction ISR (Interrupt Service Routine):

```
void doSomething() {
  if (digitalRead(boutonPinB) == 1)
    bouton++;
  else
    bouton--;
}
```

La déclaration et la fonction font en sorte que, peu importe ce que fait notre programme à nous, les flancs montants de la broche A du bouton sont détectés automatiquement (invisiblement) - le sens de rotation est calculé (on lit l'état de la broche B) - et la valeur d'une variable "bouton" est augmentée ou diminuée pour chaque impulsion d'A.
Il nous reste à vérifier si la valeur de bouton a changé et de faire le nécessaire. L'ajout d'une deuxième variable, de type `boolean` (valeurs "vrai" ou "faux"), peut nous aider.

IV.3.2.0.3. Montage

Voici le montage sur platine avec Fritzing. Le bouton poussoir sur la droite simule le bouton de contact (broches 4 et 5) qui se déclenche quand on appuie sur le bouton rotatif. L'encodeur dans Fritzing n'est pas équipé avec ce bouton.

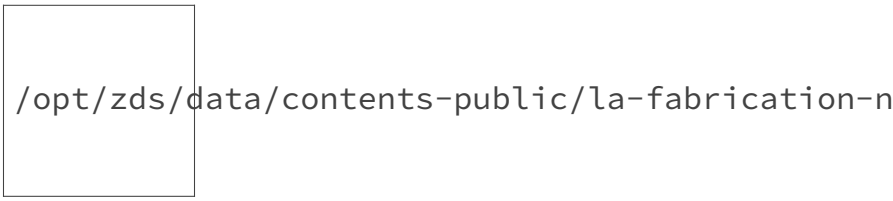


Figure IV.3.6. – Montage de l'encodeur

Et le schéma. Notez bien les condensateurs. Aussi vous notez que, pour le contact d'appui, j'ai ajouté une résistance de pull-up à la place de la résistance interne: c'est parce que le bouton que j'ai générait beaucoup de parasites et il fallait réduire la résistance de pull-up pour réduire les effets de rebond. Essayez chez vous sans la résistance d'abord (en changant la déclaration pour la broche d'entrée de `INPUT` à `INPUT_PULLUP`), mais si vous remarquez des réactions aléatoires, câblez les contacts du bouton comme dessiné ici.

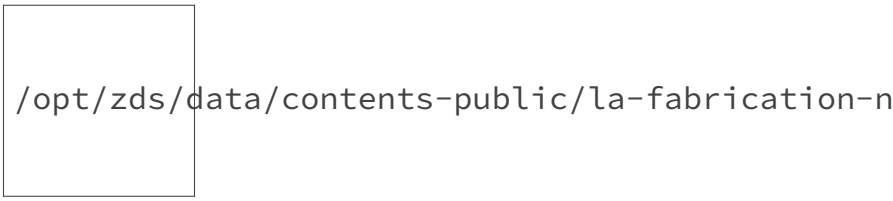


Figure IV.3.7. – Branchement de l'encodeur

IV.3.2.0.4. Sketch

Voici le sketch en entier:

```
/*
  Gestion d'un bouton/encodeur rotatif avec interruptions
  La broche commun, et un des deux broches du bouton de contact
  sont branchées sur GND.
  Des condensateurs de 100nF sont conseillés en // avec les trois
  broches de contact (A, B, et bouton)
  La broche "A" est branchée sur l'entrée 2 sur Arduino (INT0)
  Bouton est branché sur l'entrée 3 sur Arduino (INT1)
  La broche "B" est branchée sur n'importe entrée numérique

  Le sens de l'encodeur (broche A / B) n'est pas important :
  il est facile de changer le "sens de rotation" dans le code

  Les entrées sont déclarées avec les résistances de pullup
    activées

  Varier la luminosité d'un LED

  Glenn Smith  MOOC Fabrication Numérique  Mai 2014
*/

#define boutonPinA  2  // C'est l'entrée INT0
#define boutonPinB  4  // Pas d'interruption
#define boutonPinC  3  // (contact pression) C'est  l'entrée INT1

#define GreenLed 9  // ou n'importe sortie numérique

volatile byte valeur_encodeur = 0;  // volatile changé en cachette
    par l'ISR
volatile boolean encodeur_changed = false;
volatile boolean contact = false;

byte  brightness;

// Les deux fonctions ISR

void doContact() {
  // Nous arrivons ici sur flanc descendant sur broche 3
  contact = true;
}

void doEncoder() {
//  Nous arrivons ici sur flanc montant sur broche 2
if (digitalRead(boutonPinB) == 1) // Lire broche "B" de l'encodeur
  valeur_encodeur++;  // Direction +ve : augmenter la valeur
else
  valeur_encodeur--;  // Direction -ve : diminuer la valeur
}

```

```
48 void setup() {
49   pinMode(boutonPinA, INPUT_PULLUP);
50   pinMode(boutonPinB, INPUT_PULLUP);
51   pinMode(boutonPinC, INPUT_PULLUP);
52
53   pinMode(GreenLed, OUTPUT);
54
55   // encoder pin "A" on interrupt 0 - (pin 2)
56   attachInterrupt(0, doEncoder, RISING);
57   // encoder pin "C" on interrupt 1 - (pin 3)
58   attachInterrupt(1, doContact, FALLING);
59
60   Serial.begin (115200);
61   Serial.println("Hello");
62 }
63
64 void loop(){
65   if ( contact ) {  // est ce que le bouton à été appuyé ?
66     Serial.print
          ("Varier l'intensite, appuyer pour sauvegarder : ");
67     contact = false;  // sinon on ne s'arrête jamais !
68     do {
69       // utilise la valeur pour gérer une LED
70       brightness = constrain((valeur_encodeur * 4), 0, 255);
71       analogWrite(GreenLed, brightness);
72     // rester dans cette boucle tant que le bouton n'a pas été
          appuyé
73     }  while (not contact);
74   contact = false;   // sinon on s'arrête jamais !
75   Serial.print("New value= ");
76   Serial.println(valeur_encodeur, DEC);
77   delay(500);
78   }
79 }
```

IV.3.3. Simulation du bouton rotatif

Si vous n'avez pas l'encodeur rotatif dans vos pièces, ce montage avec 3 boutons classiques sert à démontrer comment utiliser deux interruptions pour gérer plusieurs boutons.

IV.3.3.0.1. Le montage

IV.3.3.0.1.1. Platine

FIGURE IV.3.8. – Montage avec 3 boutons

IV.3.3.0.1.2. Schéma

FIGURE IV.3.9. – Schéma électronique avec 3 boutons

Le boutons "**A**" et "**B**" font l'office des contactes A et B du bouton rotatif. Le troisième bouton est notre bouton d'appel / validation.
Vous pouvez utiliser le même Sketch que pour le *vrai* bouton rotatif, vérifier simplement que l'entrée "C" (pin 2) soit déclarée en `INPUT-PULLUP` pour activer la résistance pull-up.

IV.3.3.0.2. Utilisation

On peut considérer le bouton "C" comme un bouton d'appel: pour dire qu'on veut changer un paramètre, par exemple. Si on clic sur "A" cela fait incrémenter notre variable. Si on maintient "B" appuyé et on clic sur "A" la variable est décrémentée.
Il est possible de construire des menus avec juste ces trois entrées. Essayer avec l'afficheur LCD
N.B.: *Si vous voulez ajouter l'afficheur LCD il faut changer un peu le câblage par rapport à l'exemple Aduinomètre, mais les commentaires dans le sketch devraient suffire pour vous guider.*

IV.3.4. Quelques liens et autres documents utiles

IV.3.4.1. Zeste de Savoir

Il y a un tutoriel très complet sur Zeste de Savoir: https://zestedesavoir.com/tutoriels/537/arduino-premiers-pas-en-informatique-embarquee/
Ainsi qu'une catégorie regroupant différents tutoriels sur ce thème: https://zestedesavoir.com/tutoriels/ ?tag=arduino

IV.3.4.2. Le site mon-club-elec (à voir vraiment)

Le site www.mon-club-elec.fr est avant tout un site perso amateur sur lequel sont mis en ligne des développements en électronique numérique programmée, dans une optique ludique et expérimentale, "just for fun"! www.mon-club-elec.fr … voudrait aussi être un site fait pour tous ceux qui veulent s'y mettre et qui cherchent de l'info!
Lien vers l'accueil du site "mon-club-elec partie électronique numérique programmée"
Plus spécifiquement sur Arduino, cette partie est la traduction en français et commentée de la référence officielle (en anglais) du langage Arduino. Il constitue en quelque sorte un site miroir en français du site Arduino officiel.
Lien vers la partie "Référence Arduino français"

IV.3.4.3. Le site flossmanuals (les livres FabLabs et Arduino)

Flossmanuals est une plateforme de partage de livres électroniques sur les logiciels et matériels libres.
Lien vers le livre FabLabs
Lien vers le livre Arduino

IV.3.5. Logiciels autour d'Arduino

Une des particularités des cartes Arduino est leur capacité à communiquer facilement avec d'autres logiciels, également libres, pour piloter des environnements multimédia en fonction de grandeurs physiques détectées par des capteurs, ou à l'inverse de piloter des actionneurs reliés à la carte à partir d'éléments multimédia et/ou d'un PC).

IV.3.5.0.1. Processing

Processing est tout particulièrement adapté à la création plastique et graphique interactive, il s'adresse aux artistes en «arts numériques» et aux graphistes. (http://fr.wikipedia.org/wiki/Processing)
Documentation et exemples sur flossmanuals
MANUEL d'utilisation de Processing

IV.3.5.0.2. Pure Data

Pure Data (en abrégé pd) est un logiciel de programmation graphique pour la création musicale et multimédia en temps réel. Il permet également de gérer des signaux entrants dans l'ordinateur (signaux de capteurs ou événements réseau par exemple) et de gérer des signaux sortants (par des protocoles de réseau ou protocoles électroniques pour le pilotage de matériels divers). (http://fr.wikipedia.org/wiki/Pure_Data)
Documentation et exemples sur flossmanuals

Pour la réalisation d'un circuit imprimé à partir d'un montage sur plaque d'essais, il existe de nombreux logiciels existant (image du montage, schéma, typon pour gravure …)

IV.3.5.0.3. Fritzing

Fritzing est un projet de logiciel libre, destiné aux non-professionnels de l'électronique. Il a notamment pour vocation de favoriser l'échange de circuits électroniques libres et d'accompagner l'apprentissage de la conception de circuits. C'est un logiciel d'édition de circuit imprimé, disponible dans 16 langues dont le français. Il est adapté aux débutants ou confirmés en électronique pour faire rapidement des circuits simples, et est également un bon outil didactique pour apprendre à bidouiller en électronique par la pratique. (http://fr.wikipedia.org/wiki/Fritzing)
Téléchargement de Fritzing

IV.3.6. Lexique électronique et informatique anglais → français

IV.3.6.0.1. A

— ADC (*Analog to Digital Converter*): Convertisseur Analogique Numérique. Ce circuit électronique convertit une valeur analogique par échantillonnage et quantification en une valeur numérique.
— Aliasing, anti-aliasing: crénelage, anti-crénelage, c'est-à-dire lissage, en parlant d'un flux de données.
— API (*Application Programming Interface*): Interface par laquelle une application peut contrôler une couche plus basse (par exemple: des pilotes de périphériques, une bibliothèque de manipulation de chaînes de caractères, etc.).

IV.3.6.0.2. B

— binary: binaire. Seulement Vrai / Faux, ou 0 / 1.
— BJT (*Bipolar Junction Transistor*): transistor à jonction bipolaire.
— boolean: booléen, se dit d'un système qui ne peut avoir que deux états: vrai ou faux. Type de variable binaire.
— bounce: rebond.
— breadboard: plaque d'expérimentation.
— breakdown: claquage.
— bridge: pont.
— bug: dysfonctionnement logiciel (*bug* signifie insecte, cafard, bestiole, et au début de l'informatique on imputait *facilement* toute erreur aux insectes qui venaient se coller sur les lampes à vide très chaudes et perturbaient le fonctionnement). Le premier *bug* était dû à un véritable cafard qui avait fait "planter" le premier ordinateur américain, dans les années 40.

IV.3.6.0.3. C

— calibration, to calibrate: étalonnage, étalonner.
— capacitor: condensateur.
— chip: puce électronique.
— CMOS (*Complementary Metal–Oxide–Semiconductor*): technologie de fabrication de circuit intégré.
— CNC (*Computer Numerical Control*): machine à commande numérique.
— CPU (*Central Processing Unit*): micro-processeur. Fait les calculs dans un ordinateur.
— current: courant.

IV.3.6.0.4. D

— DAC (*Digital to Analog Converter*): convertit une grandeur numérique en une grandeur analogique.
— Darlington: montage en série de deux transistors ayant pour effet d'obtenir un gain global, en régime linéaire, égal au produit du gain de chaque transistor.
— deadlock: étreinte fatale, se produit quand l'arbitrage de l'attribution d'une même ressource entre deux ou plusieurs sous-systèmes s'avère impossible car ils s'attendent mutuellement.

— DEMUX (*DEMUltipleXer*): démultiplexeur. Sépare des signaux disjoints ayant été transmis par un même canal.
— DFA (*Deterministic Finite Automaton*): automate déterministe à états finis. Technique logicielle qui permet de résoudre un problème en le décomposant en une série d'états et de transitions.
— digit: chiffre.
— digital: numérique. 0,1,2,3, …
— DIY (*Do It Yourself*): faites le par vous-même, fabriqué maison, se dit d'une réalisation amateur sans connotation péjorative. Règle d'or aux Fablabs et à l'utilisation d'Arduino.
— download: action de télécharger un fichier depuis un serveur ou un périphérique.

IV.3.6.0.5. E

— earth: terre, point supposé être au potentiel commun de 0V (ne pas confondre avec la masse (*ground*)).
— edge: segment de droite joignant deux sommets.

IV.3.6.0.6. F

— fan: ventilateur.
— feedback: boucle de contre-réaction, rétro-action, retour d'information.
— feet: pied, unité de mesure impériale correspondant à **0,3048 mètre** dans le système international.
— FET (*Field-Effect Transistor*): transistor à effet de champ.
— fetch (to): aller chercher (des valeurs…)
— firmware: logiciel chargé dans un système autonome ou un microcontrôleur (on parle plus rarement de magiciel).
— flag: drapeau de signalement.
— float: variable ou nombre à virgule flottante.
— FPU (*Floating Point Unit*): coprocesseur chargé d'effectuer des opérations mathématiques sur des nombres à virgule.
— frequency: fréquence.

IV.3.6.0.7. G

— gain: facteur d'amplification.
— GPIO (*General-Purpose Input/Ouput*): bloc fonctionnel d'un micro-contrôleur gérant les entrées-sorties.
— ground: masse, point arbitraire considéré comme étant au potentiel de référence de 0V (ne pas confondre avec la terre (*earth*)).

IV.3.6.0.8. H

— H-bridge: circuit permettant de commander des moteurs pas-à-pas au moyen de transistors de puissance.

IV.3.6.0.9. I

— I/O (*Input / Output*): entrées / sorties.

— I²C (*Inter Integrated Circuit*): norme de communication série synchrone très utilisée en électronique pour faire communiquer des équipements entre eux (on parle aussi d'interface à deux fils). Inventée par Philipps.
— IC (*Integrated Circuit*): circuit intégré.
— inch: pouce, unité de mesure impériale correspondant à **25,4 mm** dans le système international.
— Inductor: inductance ou, paradoxalement, en français *self* (terme qui n'est pas employé en anglais).

IV.3.6.0.10. J

— Jumper: cavalier, sur un circuit imprimé, ou fil éléctrique permettant de relier une broche à un composant, un capteur, une plaque d'essais, etc.

IV.3.6.0.11. K

IV.3.6.0.12. L

— LDR (*Light-Dependant Resistor*): photorésistance. Capteur de lumière.
— LED (*Light-Emitting Diode*): Diode ÉlectroLuminescente (DEL).
— library: bibliothèque. Fonctions 'toutes prêtes' pour une utilisation dans son propre programme.
— lock: verrou.
— loop: boucle. Un des deux éléments nécéssaires aux programmes Arduino.
— LSB (*Least Significant Bit*): bit de poids faible dans un mot.

IV.3.6.0.13. M

— MCU (*Micro Controller Unit*): microcontrôleur disposant d'entrées sorties analogiques et numériques et de fonctions permettant de manipuler des capteurs et des effecteurs.
— mesh: maillage. (modélisation 3D)
— mil (ou *thou*): unité de mesure impériale correspondant à un millième de pouce (*inch*) soit dans le système international **25,40 µm** ou encore **0,0254 mm**.
— MOSFET (*Metal Oxyde Semiconductor Field-Effect Transistor*): transistor à effet de champ à grille métal-oxyde.
— MSB (*Most Significant Bit*): bit de poids fort dans un mot.
— MUX (*MUltipleXer*): multiplexeur. Joint des signaux distincts de façon à les transmettre par un même canal.

IV.3.6.0.14. N

IV.3.6.0.15. O

— offset: décalage.
— Op-Amp (*Operational Amplifier*): amplificateur opérationnel ou (par abus de langage) amplificateur différentiel.

IV.3.6.0.16. P

— parsing, to parse: analyse lexicale et sémantique.
— PCB (*Printed Circuit Board*): circuit imprimé.

— pin: broche. Entrée ou sortie.
— polling: scrutation. Technique logicielle qui consiste à vérifier très souvent si un état a changé en vue d'une action sans utiliser des mécanismes asynchrones.
— power: puissance.
— power rail: bus d'alimentation.
— power supply: alimentation.
— PWM (*Pulse Width Modulation*): modulation de largeur d'impulsion. Technique de modulation qui consiste à faire varier le rapport de cycle, pour une fréquence fixe donnée, entre un état haut et un état bas.

IV.3.6.0.17. Q

IV.3.6.0.18. R

— resistor: résistance.
— RTC (*real time clock*): composant chargé de garder l'heure en mémoire grâce à sa propre pile électrique.

IV.3.6.0.19. S

— short circuit: court-circuit.
— side effect: effet de bord.
— sink current: courant de fuite.
— SMD (*Surface Mounted Device*): composant monté en surface (CMS en français).
— spectrum analyzer: équipement de mesure indiquant l'intensité d'un signal dans le domaine des fréquences.
— SPI (*Serial Protocol Interface*): bus de données série synchrone utilisant quatre signaux pour communiquer avec des périphériques.
— strap: câble court utilisé pour réaliser une connexion électrique entre deux broches ou deux composants, pour les circuits à montage en surface on utilise des résistances de 0 Ω.
— sweep: balayage.

IV.3.6.0.20. T

— thermistor: thermistance.
— TTL (*Transistor-Transistor Logic*): circuits logiques à base de transistors bipolaires. Se dit aussi de la tension utilisée dans de tels circuits (5 volts, par exemple).

IV.3.6.0.21. U

— UJT (*UniJonction Transistor*): transistor unijonction.
— upload: action de télécharger un fichier en direction d'un serveur, on utilise aussi le mot téléverser. Utilisée dans le logiciel Arduino mais aussi dans les serveurs Web.
— USB (*Universal Serial Bus*): bus de donnée série très utilisé pour communiquer avec des périphériques informatiques. L'Arduino l'utilise pour se connecter à l'ordinateur afin d'être programmé. Voir ports COM.

IV.3.6.0.22. V

— valve: tube à vide.

— Vcc: tension positive d'alimentation d'un circuit.
— Vdd: autre dénomination pour Vcc.
— Vee: tension négative d'alimentation (ou souvent le 0V) d'un circuit.
— vertex (pluriel: *vertices*): sommet.
— voltage: tension.
— voltage divisor: diviseur potentiométrique (ou aussi diviseur de tension).
— Vss: autre dénomination pour Vee.

IV.3.6.0.23. W

— waveform: caractérise la nature d'un signal alternatif régulier; trois grandes familles existent: carré, triangulaire et sinusoïdal.

IV.3.6.0.24. X

IV.3.6.0.25. Y

IV.3.6.0.26. Z

IV.4. Arduinomètre

Introduction

FIGURE IV.4.1. – Arduinomètre

IV.4.1. Arduinomètre avec sonde LM35

IV.4.1.1. Arduinomètre avec sonde LM35

IV.4.1.1.1. Le LM35

C'est un circuit intégré, de la taille d'un petit transistor, conçu pour générer une tension proportionnelle à la température en son cœur.

FIGURE IV.4.2. – Détails de la sonde

D'après les données fabricant, on peut atteindre une précision de 0,5°C. Voici un extrait des données de chez National Semiconductor:

FIGURE IV.4.3. – Fiche technique (datasheet) du LM35 (extrait)

Nous allons nous contenter du branchement "simple" (schéma de gauche), prévu pour la plage de températures de 2°C à 150°C.
Notez que cette sonde est déjà calibrée, nous avons juste à convertir la tension de sortie en valeur de température en °C.
La conversion est assez simple: la tension de base est 0V et elle augmente de 10mV pour chaque

degré Centigrade. Donc à 20°C on aura une tension de 200mV. Ainsi à 150°C la tension sera de 1500mV, ou 1,5V.
L'Arduinomètre est programmé (pour des raisons de précision) à n'accepter que les tensions jusqu'à 1,1V: donc notre plage de température utile sera de 2°C à 110°C.
##Schéma de branchement
Très simple: la bête n'a que trois pattes: +5v (alimentation) ; V_{sortie}; 0v. Voici le montage sur une platine (j'ai omis l'afficheur LCD: ses branchements ne changent pas):

FIGURE IV.4.4. – Montage du LM35

Une fois n'est pas coutume: pas de schéma, c'est tellement simple.
##Sonde flexible
Comme vous voyez sur la première image, il est possible de fabriquer une sonde flexible à l'instar de notre sonde à diode. A cause des tensions très faibles à basse température, je vous conseille de trouver du câble blindé (genre câble pour microphone). Sinon, au moins faire une paire torsadée avec le fil de sortie et le fil 0v.
Isoler les trois "pattes" avec de la gaine thermorétractable puis protéger l'ensemble avec une deuxième gaine par-dessus (dans l'image j'ai retiré la gaine de protection: on voit les trois pattes).
Dans l'image suivante on voit les branchements avec un câble blindé: fil rouge = +5v ; fil blanc = V_{sortie} ; fil orange = blindage = 0v.

FIGURE IV.4.5. – Câblage de la sonde

Comme je l'ai dit en préambule, la température correspond à la température au cœur de la bête. L'encapsulation en plastique est relativement isolante, la chaleur est plutôt transmise par les pattes. Ce point n'a pas d'incidence dans la plupart des cas, mais vous remarquerez peut-être un temps de réponse un peu long par rapport à la sonde diode.

IV.4.1.1.2. Modification Sketch

Je vous laisse modifier le sketch vous-même! Non, non - ce n'est pas si difficile que cela. La seule vraie modification c'est pour les paramètres de la commande `map()`.
La gamme de température devient de **2°** à **110°** ; et la gamme de valeurs retournées par `analogRead()` devient (20/1.075) = 11023.
##Double sonde
Pour voir à quel point la diode est précise (ou pas) il est possible de modifier le sketch afin de lire la valeur des deux sondes. L'Arduino est équipé de 6 entrées analogiques, il n'y a pas de raison de s'en priver!

Attention, cependant: le convertisseur analogique > numérique (ADC) a besoin d'un certain temps pour effectuer ses opérations, et un temps de repos entre chaque lecture. Je vous conseille de faire en sorte de laisser un minimum de 1ms entre deux lectures.
Par exemple:

```
diode_value = analogRead(diode_pin);  // lire valeur de la sonde
    diode
delay(1); // attente
lm35_value = analogRead(lm35_pin);  // lire valeur de la sonde lm35
```

Bon courage!
Glenn

IV.4.2. Arduinomètre avec thermistance

##La thermistance
C'est une résistance variable, sa valeur (en Ohms) varie avec sa température. Deux grandes familles de thermistances existent: les thermistance à coefficient *négatif* (NTC) ; et les thermistances à coefficient *positif* (PTC).
Pour les NTC la résistance ***diminue*** avec une augmentation de T° ; et, vous avez deviné, ***augmente*** pour les PTC. La plupart des thermistances en vente sont des NTC - et c'est sûrement ce que vous avez dans votre kit.
##Réponse non-lineaire
Un des soucis avec les thermistances c'est que la variation de résistance n'est pas linéaire par rapport aux changements de température. C'est-à-dire que, supposant qu'entre 110° et 100° la résistance change de 1000ohms; entre 20° et 10° la différence de résistance ne sera peut-être que de 300ohms (chiffres inventés, mais cela donne l'idée).
Souvent dans la documentation fabricant on ne trouve que des tables de valeurs, pas des courbes. Les rares fois où des courbes sont publiées, elles sont de ce style (données réelles, graphique tableur):

Figure IV.4.6. – Courbe logarithmique

Où est le problème? C'est vrai que la courbe semble relativement droite. Mais l'axe de résistance est en échelle **LOGARITHMIQUE**. Si je vous montre les mêmes valeurs sur une échelle linéaire vous comprendrez vite notre souci:

FIGURE IV.4.7. – Courbe linéaire

Aïe!

Pour corser le problème (au moins pour moi), les thermistances ne sont aucunement normalisées: une thermistance avec une résistance de 47kohms de chez Dédé n'aura pas la même courbe de réponse qu'une thermistance de même valeur chez Tartampion.
C'est essentiellement pour ces raisons que j'ai choisi de faire l'Arduinomètre avec une diode... Mais, pour des raisons qui m'échappent, la diode a été "oubliée" de la liste des pièces, me semble-t-il.
Tout n'est pas perdu, cependant. Si on se contente d'une utilisation sur une plage de températures restreinte, autour de la température 'nominale' (25°), avec quelques astuces électroniques on peut avoir une réponse assez linéaire.
Voici une façon de faire (car il y a pléthore de solutions plus ou moins complexes). J'utilise cette façon pour un thermostat que j'ai conçu pour gérer un chauffage de serre maraîchère ou un châssis chauffant pour les plantes.

IV.4.2.0.1. Montage

L'ajout de deux résistances (pour de meilleurs résultats on choisit des résistances de précision, 1% ou mieux - mais ce n'est pas critique ici) fait en sorte d'aplatir la courbe sur la plage d'environ +/- 50°C autour du point nominal. Voici donc un montage *compromis* entre les valeurs de résistances que vous devrez avoir, et une correction au plus juste:

FIGURE IV.4.8. – Branchement de la thermistance

Si vous ne pouvez pas trouver une résistance de 47k, mettre plusieurs résistances de 10k en série. Plus vous êtes proche des 47k (ex. 5 x 10k) mieux c'est. Voici ce montage sur la platine (sans l'afficheur):

FIGURE IV.4.9. – Montage de la thermistance

Comme les résistances, les thermistances ne sont pas polarisées: elles n'ont pas de sens de branchement.
##Sonde
Pour calibrer le montage, il faut fabriquer une sonde étanche, comme pour la diode. Pas de précaution particulière, vous pouvez suivre la même procédure que pour la sonde diode. L'étanchéité est importante car la présence d'eau va fausser les valeurs.

##Étalonnage
Procéder comme pour le montage diode: mesure des valeurs proche de 0°C (verre d'eau glacé) ; et à 100°C (eau bouillante). Ensuite corriger les valeurs dans la commande `map()`
Si vous avez une thermistance ET une diode, vous pouvez comparer leur comportement en modifiant l'Arduinomètre: utiliser une deuxième entrée analogique. Regarder à la fin de l'article "Arduinomètre LM35" pour plus d'infos.
Good luck!
Glenn

Conclusion

Un tutoriel, ce sont bien entendu des auteurs, mais ce sont aussi plein de petites mains qui sont venues aider à son import depuis FUN vers ZdS:

— Andr0
— artragis
— Coyote
— elyppire
— firm1

Un gros merci à eux pour les efforts fournis!