N°6 | MAI - JUIN 2015

HACKABLE MAGAZINE

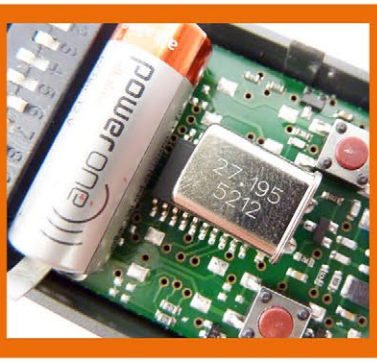
DÉMONTEZ | COMPRENEZ | ADAPTEZ | PARTAGEZ

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

ONDES & SDR

Testez la sécurité de votre télécommande de garage

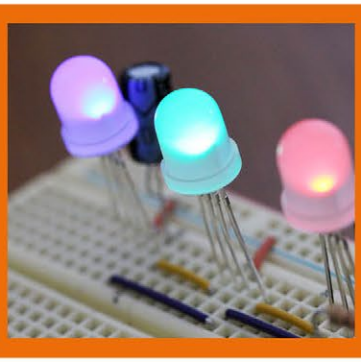
p. 76



WS2812B

Pilotez des dizaines de leds multicolores avec un fil

p. 08



ATTINY85

Créez des feux de circulation pour les Lego de vos enfants

p. 40

ARDUINO 1.6

Découvrez la nouvelle gestion de l'EEPROM interne

p. 32

Anonymat, surveillance, intimité...

PROTÉGEZ VOTRE VIE PRIVÉE

...avec Raspberry Pi et Tor

p. 54

1 - Configurez votre point d'accès Wifi

2 - Comprenez le fonctionnement de Tor

3 - Anonymisez vos communications

RADIOCOMMANDE

Connectez votre récepteur Futaba S.Bus à l'Arduino

p. 22

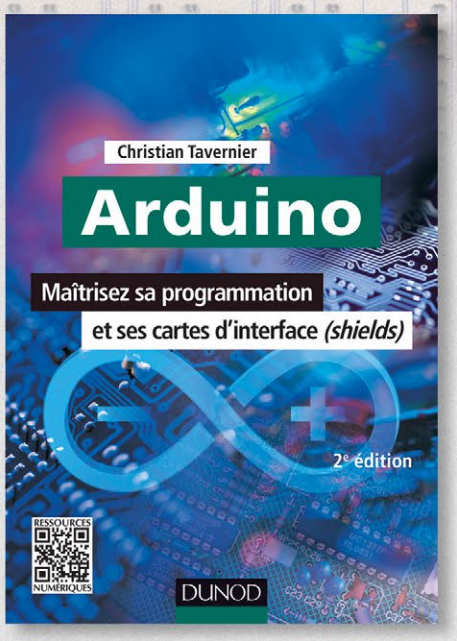
CONTRÔLE

Gérez les encodeurs rotatifs en toute simplicité

p. 20

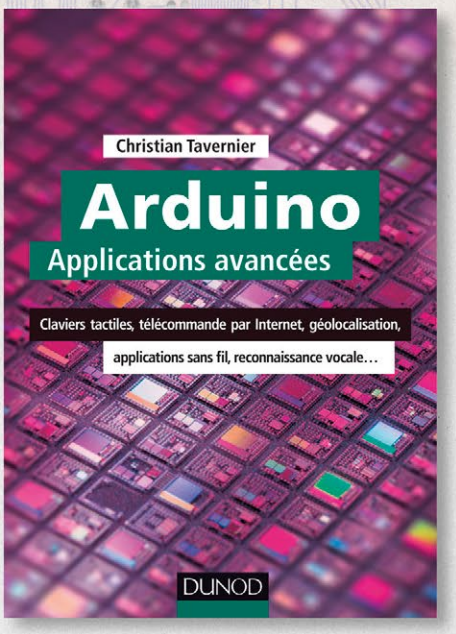
L 19338 - 6 - F: 7,90 € - RD

LES OUVRAGES PRATIQUES POUR LES MAKERS



9782100710409, 232 p., 24,90 €
Christian TAVERNIER

Tous les éléments nécessaires à la conception et à la mise en œuvre de nombreuses applications performantes avec Arduino.

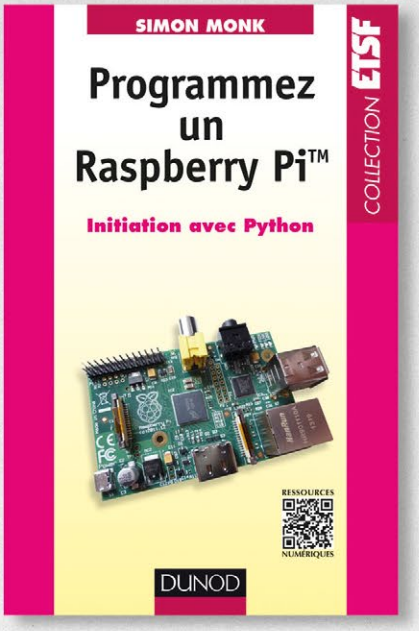


9782100582051, 224 p., 27 €
Christian TAVERNIER

Pour ceux qui souhaitent développer des applications évoluées à base d'Arduino.

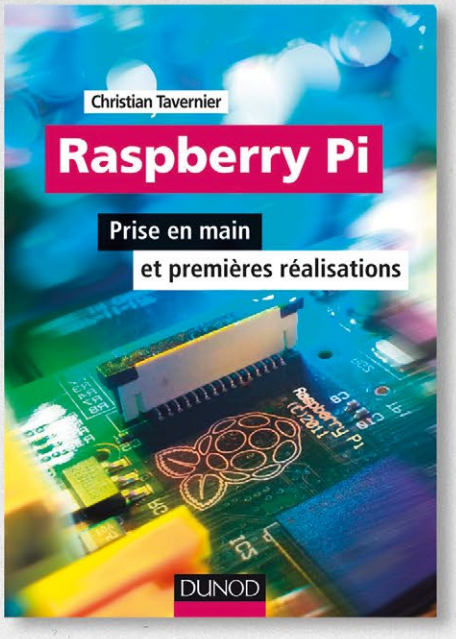
9782100717934, 304 p., 29,90 €
Tero KARVINEN et al.

Bien utiliser les capteurs pour concevoir des montages avec Arduino ou Raspberry Pi qui interagissent avec leur environnement.



9782100706594, 192 p., 16,90 €
Simon MONK

Cet ouvrage s'adresse à ceux qui découvrent le Raspberry Pi et leur explique en termes simples comment écrire des programmes à l'aide du langage Python.



9782100598915, 224 p., 19,90 €
Christian TAVERNIER

Configurez, paramétrez et découvrez les nombreuses possibilités du « micro-ordinateur » Raspberry Pi.

TOUT LE CATALOGUE SUR **DUNOD.COM**

DUNOD
ÉDITEUR DE SAVOIRS

ÉDITO

Je suis concerné... et sans doute vous aussi !

À l'heure où j'écris ces lignes, je suis en parallèle les débats à l'Assemblée nationale concernant le projet de loi sur le renseignement. Il est plus que probable qu'au moment où vous lirez cet édito, le texte, légèrement amendé, aura été adopté et en bonne voie d'être appliqué. À moins d'un passage puis d'un rejet par le Conseil constitutionnel, chargé de s'assurer que les lois sont conformes à la Constitution. Passage cependant conditionné par une saisine nécessitant 60 députés ou 60 sénateurs.

Pourquoi en parler ici, dans un magazine traitant de technique, d'exploration technologique et d'électronique ? Une raison pourrait être le fait que les activités que légalise cette loi reposent massivement sur l'utilisation de technologies intéressantes. L'IMSI catcher par exemple est souvent présenté comme un équipement complexe, coûteux et difficile à se procurer. En réalité, un IMSI catcher, comme en a fait la démonstration Chris Paget à Defcon 18 en 2010, peut se résumer à un équipement SDR (un USRP d'Ettus Research) utilisé avec les bons logiciels open source (OpenBTS+Asterisk sur GNU/Linux). Son travail est simplement de se faire passer pour une BTS (*Base Transceiver Station*, communément appelée une « antenne-relais GSM ») afin que les mobiles dans son champ d'action s'y associent pour communiquer.

De la même manière, la loi prévoit la mise en place de « boîtes noires », potentiellement chez les opérateurs, les fournisseurs d'accès et les hébergeurs. Il s'agit de dispositifs destinés à faire de la prévention et de la détection de menaces terroristes en repérant une « *succession suspecte de données de connexion de manière anonymisée* ».

Là encore, il n'y a rien qui ne soit déjà à la portée de n'importe quel bidouilleur, programmeur, administrateur système ou utilisateur avancé et curieux. N'importe quelle carte ou ordinateur, disposant de deux ports Ethernet peut, dans une certaine mesure,

suite page 4...

Hackable Magazine
est édité par Les Éditions Diamond

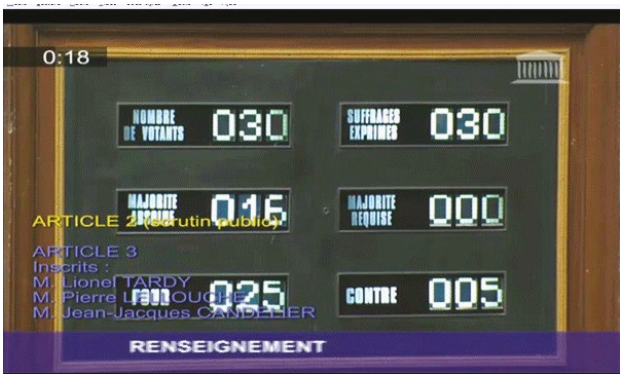
LES ÉDITIONS DIAMOND
STIMULANT DE MATIÈRE GRISE

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

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

Plate-forme de Saint-Quentin-Fallavier.
Tél. : 04 74 82 63 04
IMPRIMÉ en Allemagne - PRINTED in Germany
Dépôt légal : À parution,
N° ISSN : en cours
Commission paritaire : K92470
Périodicité : bimestriel
Prix de vente : 7,90 €

SOMMAIRE



Lors des débats et des amendements concernant le projet de loi sur le renseignement, l'Assemblée nationale était, comme souvent, relativement déserte. Ici, les résultats du scrutin concernant l'article 2 amendé sur ce qu'il est devenu courant de désigner par les termes « boites noires ».

faire exactement la même chose que ces « boîtes noires ». Les systèmes anti-SPAM comme Apache SpamAssassin reposent sur le mécanisme de filtrage bayésien : une corrélation pondérée entre la présence de certains éléments et la probabilité qu'il s'agisse de SPAM. Remplacez les mails par des données anonymes et choisissez judicieusement les éléments ainsi que leur « poids » dans le calcul et vous avez un système de détection de « successions suspectes de données de connexion ». Les anti-virus fonctionnent sur un principe proche, tout comme les détecteurs d'intrusion (IDS pour *Intrusion Detection System*) qui inspectent le trafic réseau à la recherche de comportements révélateurs d'une tentative d'attaque.

Bien que le fonctionnement et les algorithmes utilisés resteront secrets, ils seront soumis au contrôle de la CNCTR, la « Commission Nationale de Contrôle des Techniques de Renseignement ». Je vais l'exprimer autrement : c'est un ensemble de logiciels **propriétaires** dont le fonctionnement est validé par un groupe incluant « *une personnalité qualifiée pour ses connaissances en matière de communications électroniques* ».

Le simple fait que le code source des logiciels utilisés ne soit pas disponible et vérifiable par tous est suffisant pour douter de sa qualité, son efficacité et sa précision. La composition de la CNCTR, n'arrange rien à cela : 3 députés, 3 sénateurs, 3 membres magistrats du Conseil d'État, 3 magistrats de la Cour de cassation et une seule personne qui potentiellement s'y connaît en technologie, sous la forme d'un représentant désigné par l'ARCEP ! Je vous laisse faire le calcul de la part ridicule qui est faite par cette commission aux seuls acteurs qui légitimement pourraient comprendre ce qui est réellement mis en œuvre...

Ces deux exemples ne sont qu'une partie de ce qui va être (ou est maintenant) légalisé, et il n'y a absolument rien de technologiquement novateur dans tout cela. Et je sais, par expérience, que ce n'est pas là quelque chose de très complexe. Le problème n'est donc pas technique, loin de là, en dehors de la mise à l'échelle de ces systèmes. Il y a une différence entre analyser les données d'une connexion et celles de milliers, de centaines de milliers ou de millions de connexions. Mais c'est une simple question de moyens, de puissance et de financement.

Non, en vérité le problème n'est pas technique, mais analytique, car le texte, dans sa version disponible à l'heure d'aujourd'hui sur le site de l'Assemblée nationale, parle de « mise en œuvre sur les informations et documents traités par leurs réseaux [opérateurs entre autres] d'un dispositif destiné à révéler, sur la seule base de traitements automatisés d'éléments anonymes, une menace terroriste ». Le texte ajoute ensuite « *Si une telle menace est ainsi révélée, le Premier ministre ou l'une des personnes déléguées par lui peut décider de la levée de l'anonymat sur les données[...]* » (précisons ici un non-sens important : s'il est possible d'identifier des personnes dans les données, c'est qu'elles ne sont pas anonymes).

Nous voici au cœur du problème. Laissez-moi vous exposer cela de manière plus intelligible qu'un texte de loi : un système automatisé sélectionne des « cibles » anonymes probables et une entité humaine, sur cette base ou en partie, décide ou non de lever l'anonymat et de révéler à qui sont ces données. La question est donc : qu'est-ce qui caractérise, dans des données anonymes, une menace terroriste ?

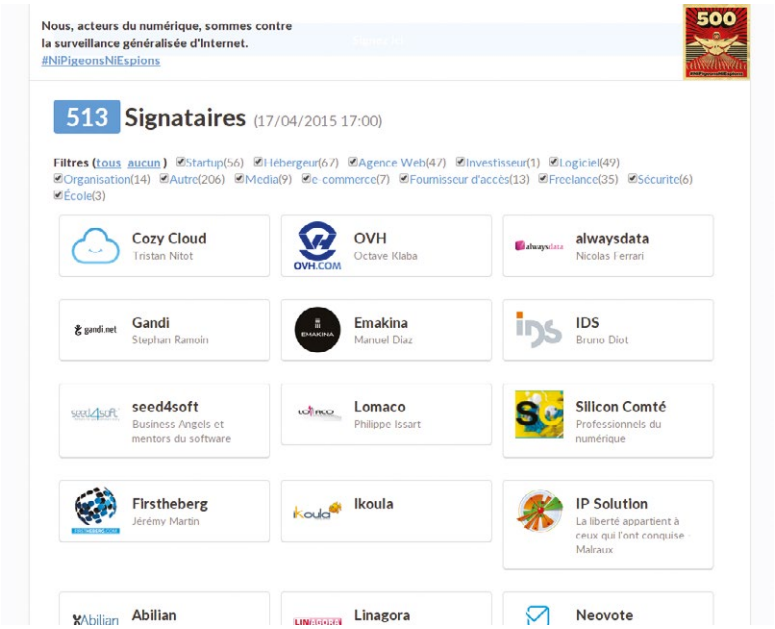
Rapidement, dès le début des débats, une pétition en ligne a été mise en place sur change.org pour que les opposants puissent faire entendre leur voix (https://www.change.org/p/retirez-le-pjlrenseignement-le-big-brother-fran%C3%A7ais-stoploirenseignement).

En ce qui me concerne, caractériser les prémices d'une action future (une menace donc) repose sur deux choses : une détermination motivant un acte et la capacité d'accomplir ce même acte. Évaluer la détermination ou la motivation d'une personne n'est pas simple, les humains étant ce qu'ils sont. Internet en est la démonstration permanente, il est difficile de faire la différence entre l'expression d'idées fortes dans le seul but de se soulager ou choquer et une vraie préméditation exposée publiquement.

En me faisant l'avocat du diable, la conclusion logique qui s'impose est donc que les personnes techniquement capables de réalisations étonnantes (par rapport à la grande majorité) méritent la plus grande attention, car ont les moyens d'agir.

Et nous voici arrivés au point de convergence : les personnes techniquement capables de réalisations étonnantes, c'est vous, c'est moi !

Vous en doutez ? Nous sommes au sixième numéro de *Hackable*.



Les utilisateurs et citoyens n'ont pas été les seuls à vouloir marquer leur opposition au projet de loin. Ainsi le site ni-pigeons-ni-espions.fr a recueilli plus de 500 signatures d'acteurs du numérique demandant « une révision en profondeur du projet de loi sur le renseignement, à commencer par le retrait pur et simple du dispositif nous imposant l'installation des « boîtes noires », parmi lesquels OVH, Gandi, l'INRIA, NBS System, la Fédération FDN, Mediapart, Framasoft, etc.

La capacité à accomplir un acte, elle, est sans doute plus sérieuse, car en vérité tout le monde peut déblatérer longtemps et publiquement sur le fait que telle ou telle action doit être accomplie. Si la personne n'en est pas capable, la probabilité du passage à l'acte est quasi nulle. Inversement, par contre, une personne capable d'accomplir énormément, car disposant des compétences et des connaissances, n'a que sa sagesse et son éthique comme garde-fou.

Au bout de presque une année de publication, qu'avons-nous appris, exploré ou expérimenté ? Nous avons acquis des techniques de base applicables aux systèmes embarqués, à l'électronique, aux microcontrôleurs... Nous avons détourné l'utilisation d'un récepteur DVB-T pour inspecter le trafic aérien ou interprété des signaux de pagers... Nous avons modifié des équipements destinés à une utilisation précise pour les adapter à un autre usage... Nous avons appris à utiliser des systèmes autonomes programmables... Nous avons joué avec les signaux IR des télécommandes... Nous avons énervé nos collègues... Nous avons utilisé des systèmes de communication sécurisés entre machines... Nous avons inspecté des bus de données... Et finalement dans ce numéro-ci, nous avons découvert les limitations des télécommandes de garages et exploré le monde des radiocommandes...

Ce ne sont là que quelques sujets des numéros publiés, mais les faits sont là : l'objectif de ce magazine est d'apprendre et se donner la capacité de maîtriser la technologie par une utilisation réelle et non conventionnée. En termes de prédictibilité, nous sommes donc, vous et moi, une population à risque, tout comme les radioamateurs, les programmeurs, les informaticiens, les chimistes... car nous avons la capacité de changer ce qui nous entoure et de l'utiliser à des fins que seule notre morale limite. Dois-je le dire autrement ? D'un point de vue des connaissances pures et des informations acquises, qu'est-ce qui fait la différence entre vous et moi, et un terroriste avec un livre d'électronique ou de chimie niveau licence ou master ?

Mais ne vous inquiétez pas. Le simple fait de lire *Hackable* ne fait pas de vous un suspect, du moins pas plus que n'importe quelle personne curieuse, créative et prompt à expérimenter pour enrichir ses connaissances techniques.

Les algorithmes permettant de classifier les comportements « à risque » sont incapables de faire la différence entre informations, connaissances et sagesse. La sagesse exige une conscience qui est et restera encore longtemps le saint Graal de l'intelligence artificielle. De la même manière que vous recevez toujours du SPAM, que les malwares/virus se propagent encore et que les attaques sur des serveurs ne sont pas systématiquement stoppées, la classification des comportements est prompte à montrer ses limites.

coromonadalix

Les systèmes, si performants et rapides soient-ils, ne peuvent tout simplement pas « comprendre » une idée, un concept, un scénario, une stratégie ou une personne.

Pire encore, les erreurs et faux positifs sont monnaie courante. Qui n'a jamais eu un mail légitime classé dans SPAM ? Qui n'a jamais eu un avertissement non pertinent de son antivirus ? Quel administrateur n'a jamais eu d'alerte intrusion infondée ? Qui sera sélectionné automatiquement, et verra l'anonymat de ses données levé, pour avoir consulté un site douteux, utilisé Tor, chiffré un mail ou tenté par un moyen ou un autre d'utiliser une alternative ?

Car cela aussi a été exprimé en parlant des terroristes : « *Ces personnes utilisent des procédés clandestins, souvent des outils spécifiques ou détournés de leurs usages originels [...] Ce sont ces procédés, une fois repérés, qui permettent de déterminer les algorithmes* » a dit le député Le Drian devant l'Assemblée nationale le 15 avril au soir. De « peu connu » à « clandestin », il n'y a qu'une légère différence d'appréciation. IRC, VPN, Tor ou Tails, dans ce sens sont-ils des procédés clandestins ? Mais ce qui me choque le plus c'est la mention d'outils « détournés de leurs usages originels » et je pense que vous savez parfaitement pourquoi. En particulier quand, dans la même séance, le ministre de l'Intérieur, Bernard Cazeneuve, soutenant ce projet de loi, dit ouvertement à propos des inquiétudes relayées dans les médias que ce qu'il y a dans les articles de presse, par principe, il ne le croit pas, par nature et par essence (affirmation qu'on retrouve dans la vidéo de la 217ème séance à 49mn50s **[1]**, mais étrangement absente du compte-rendu sur le site officiel de l'AN **[2][3][4]**). Lorsqu'on est journaliste et rédacteur en chef d'un magazine de presse qui s'est donné comme mission d'aider ses lecteurs à se montrer créatif par le détournement d'outils (en vérifiant tout ce qui est écrit), comprenez que c'est là quelque chose de difficile à entendre, surtout de la part d'un représentant du peuple.

Mise à jour de dernière minute (19/04 22:15) : Depuis, monsieur le ministre Bernard Cazeneuve s'est exprimé dans un article en ligne **[5]** et le CR est corrigé. Dans les grandes lignes, il s'agit d'une erreur de retranscription et ses propos ont été mal interprétés. Il y précise par ailleurs « *[...]ces informations de journalistes, je ne les prenais pas pour argent comptant, mais les vérifiais[...]* » à propos de son utilisation d'articles de presse en guise d'argumentaire par le passé (2009 - affaire Karachi). Je ne saurai donc que trop conseiller à notre ministre de l'Intérieur de faire de même lorsque ses services lui assurent que des algorithmes peuvent détecter et identifier efficacement des terroristes sans une avalanche de faux positif, alors même que beaucoup de spécialistes et d'experts en IA et en sécurité affirment le contraire.

Le gouvernement pour sa part a également souhaité clarifier sa position en proposant une page pour faire « le point sur les infos et les intox sur ce projet de loi essentiel à notre démocratie », car « le texte fait l'objet de nombreuses rumeurs et d'autant de fantasmes » (http://www.gouvernement.fr/le-vraifaux-du-gouvernement-sur-le-pjlrenseignement). Un vrai/faux que LeMonde.fr s'est empressé de vérifier et mitiger.

C'est précisément là que l'argument souvent avancé par les personnes ne prêtant pas suffisamment attention à la protection de leur vie privée s'effrite : « *Qu'ils me surveillent donc, je n'ai rien à me reprocher* ». La vérité est que nous n'en savons rien. Nul n'est censé ignorer la loi, certes, mais il y a une différence entre respecter la loi et avoir sans le savoir un comportement « numériquement » assimilé à celui d'un terroriste. L'utilisation des termes « sans le savoir » n'est pas une figure de style, puisque justement, les algorithmes étant non publics, on ne peut qu'ignorer le pourquoi d'une éventuelle similarité et, pire encore, ignorer le simple fait que cette similarité existe.

Combien de personnes déclencheront un « ciblage » ? Combien de personnes le sauront ? Quel sera le taux de faux positif avéré après vérification humaine ? Quel volume de ressources sera engagé pour rien ? En d'autres mots : quelle sera vraiment l'efficacité d'un tel système ? Difficile, sinon impossible de répondre. Il n'y a pas eu d'études, de tests, d'expérimentations techniques dignes d'une véritable démarche scientifique. Je vous le dis comme je le pense, cela ne fonctionnera pas, car ces algorithmes « magiques » n'existent tout simplement pas. Si je me trompe, faire la preuve de mon erreur est tout aussi simple : il suffit de les soumettre à une évaluation publique comme cela doit toujours être le cas, par exemple, pour les algorithmes de chiffrement et leurs implémentations (et même là, des bugs peuvent subsister, cf Heartbleed, shellshock, etc.).
« Rendre public l'algorithme, c'est la garantie de son inefficacité. »

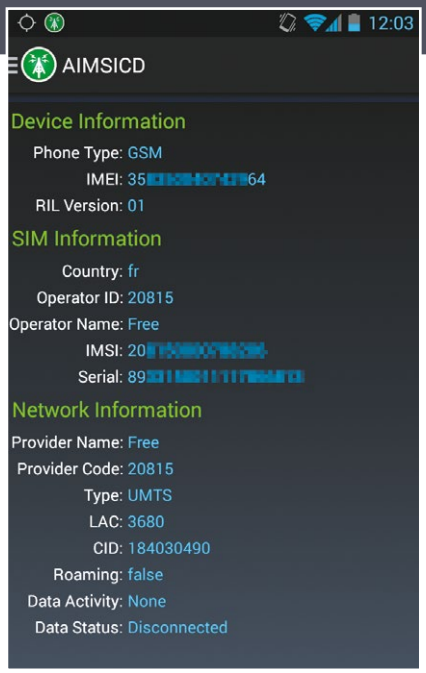
N'importe quelle personne ayant un minimum de connaissances techniques et d'expérience peut se rendre compte que ce ne sont pas les outils et les systèmes qui forment le fond du problème, mais l'interprétation même de l'information. Une machine n'interprète pas les données, elle les traite, les quantifie, les regroupe et les classe.

Combien de défenseurs de cette loi envisagent avec justesse l'abysse séparant la géolocalisation d'une photo

et la difficulté de comprendre ce qu'elle représente. Un ordinateur est capable d'extraire des données EXIF GPS d'une photo en quelques millisecondes pour les afficher sur une carte, mais il aura bien du mal à faire la différence entre un masque et un visage, chose qui, pour nous humains, est quasi instantanée. La géolocalisation a pourtant des airs de science-fiction alors que l'interprétation effective et juste d'une image, qui nous paraît si intuitive, relève encore et toujours du pur fantasme dès lors qu'on parle de machine.

Mais qui est capable de faire la part des choses si ce n'est celui qui s'est déjà frotté à cet exercice ? N'avez-vous jamais eu l'idée un jour d'écrire un programme qui se comporte comme un humain avant de vous rendre compte de l'ampleur de la tâche et finir avec un simple bot facile à démasquer ? Oui ? Alors vous savez. Ce n'est pas le cas pour des personnes ayant pour principales connaissances le fonctionnement des rouages de l'État au détriment de quelques bases techniques et, de ce fait, extrêmement sensibles aux promesses naïves et aux illusions qui accompagnent souvent la technologie. Comment voulez-vous accorder une quelconque crédibilité ou confiance à des députés tant éloignés des réalités techniques qu'ils osent affirmer, tout à fait confiants, qu'un IMSI catcher permet « d'attraper les numéros de toutes les cartes SIM alentour » ?! C'est pour cela que les instances gouvernementales font normalement appel à des experts. Et ici les experts comme RSF, la FDN, la Ligue des Droits de l'Homme, la CGT Police, le Syndicat de la magistrature, l'USM, et la CNIL... ont clairement émis des doutes, des craintes et des avis négatifs sur la question.

« *Le développement progressif de l'humanité dépend largement de ses inventions qui sont les produits par excellence de son esprit créateur. Son but ultime est la maîtrise totale du monde matériel, l'exploitation des forces de la nature pour les besoins de l'homme. C'est en cela que réside la tâche difficile de l'inventeur qui est souvent incompris et mal récompensé.* » — Nikola Tesla, « Mes inventions », Chapitre 1.



Android IMSI-Catcher Detector (AIMSICD) est une application Android en logiciel libre qui permet potentiellement de détecter la présence d'IMSI-Catcher, ces équipements se présentant comme des « tours GSM » et pouvant, entre autres choses, intercepter le trafic des téléphones mobiles.

Souvent incompris, mal récompensé... et rarement entendu dès lors qu'il ne va pas dans le sens d'attentes arbitrairement définies (on pourrait se dire qu'à défaut d'être entendu, on sera écouté, mais la gravité de la situation laisse malheureusement peu de place à l'humour).

Se passera donc ce qui devra se passer, car la technologie, comme la nature et la psyché humaines, apportent toujours des réponses et rétablit systématiquement l'équilibre. Le chiffrement et la signature des mails ne sont pas à la portée de tous, car le domaine n'est pas suffisamment intuitif et simple, pas plus que la mise en place de VPN. L'utilisation de systèmes adaptés à la défense de la vie privée, à la liberté d'expression et à la lutte contre les régimes d'oppression, en revanche, étrangement, formera une solution facile à utiliser, et ce presque sans effort. Tor (et le *bundle* Tor Browser) est sans doute le plus connu de ces systèmes et celui-ci va, sans l'ombre d'un doute, énormément gagner en popularité en France dans les prochains mois.

Et le plus paradoxal en cela est le fait que parmi les utilisateurs du réseau Tor se trouve déjà une population des plus éclectiques, mais ayant parfaitement assimilé les implications d'une perte d'anonymat. Elle se compose aussi bien d'utilisateurs prudents que de délinquants, de journalistes, de trafiquants, de reporters ou d'agents des organismes de renseignement (hé oui, l'anonymat est aussi indispensable lorsqu'on est un agent du FBI chargé de pénétrer un réseau pédophile).

La technologie va toujours plus vite que la capacité d'assimilation des structures et des mœurs. Qui sait ce qu'il se passera lorsque la vitesse limite sera atteinte et quelle forme prendra alors l'onde de choc, tel un bang supersonique. Mais le mouvement dans ce sens est engagé depuis longtemps et l'histoire suivra son cours. Comme le dit un vieux proverbe : « Prenez ce que vous voulez... puis payez-le ».

Denis Bodor

[1] http://videos.assemblee-nationale.fr/video.6695.1ere-seance--renseignement-suite-article-2-inscrits-a-l-apres-l-article-3-15-avril-2015

[2] http://www.assemblee-nationale.fr/14/cri/2014-2015/20150217.asp

[3] Cette différence entre vidéo et retranscription était détectée et reportée au moment de la rédaction de cet édito par votre humble serviteur sur Twitter et mes confrères de L'Obs/Rue89 ont poussé les investigations **[4]**. Le ministère de l'Intérieur nie toute intervention, il s'agirait d'une simple erreur.

[4] http://rue89.nouvelobs.com/2015/04/17/tiens-petite-phrase-anti-presse-bernard-cazeneuve-a-disparu-258723

[5] http://rue89.nouvelobs.com/2015/04/19/cazeneuve-repond-a-rue89-double-mauvais-proces-258746

WS2812 : LA LED INTELLIGENTE

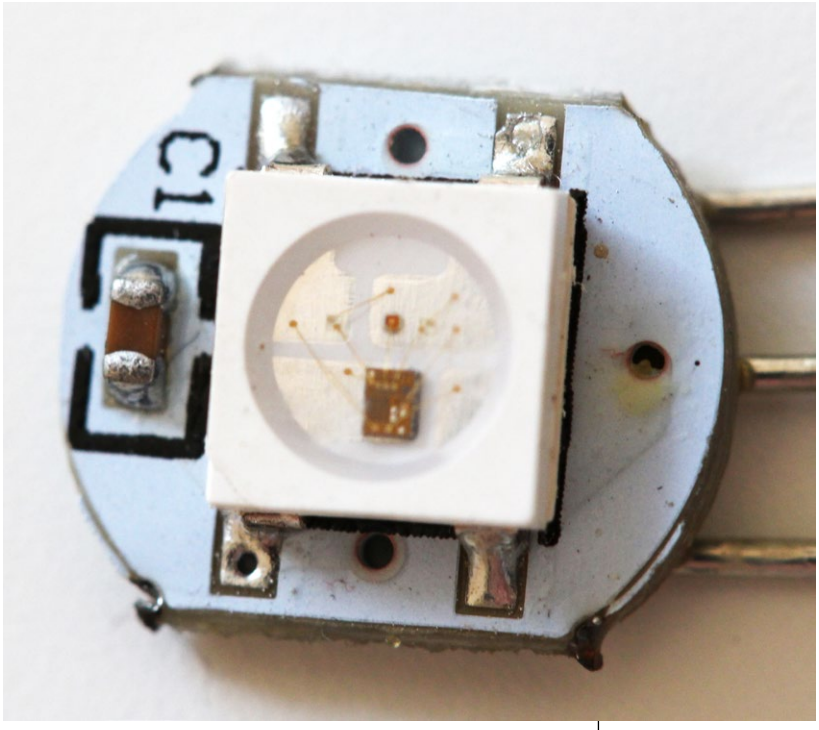
Denis Bodor

Elles sont belles, elles sont excitantes, elles sont captivantes... Et maintenant elles sont aussi intelligentes. Je parle bien entendu des leds qui illuminent nos vies et qui sont un facteur multiplicateur de l'intérêt pour n'importe quel projet. Mais qu'est-ce qu'une led intelligente après tout ? Imaginez simplement que vous puissiez à l'aide d'un seul signal ordonner à une led de prendre une couleur et une intensité donnée : voici la nouvelle génération de leds. Démonstration avec la WS2812.

coromonadalix

Les leds multicolores ne sont pas une nouveauté. D'abord bicolores, vertes et rouges et donc équipées de trois pattes, celles-ci permettaient déjà d'obtenir plusieurs couleurs. Vert, rouge et vert+rouge = jaune dans un premier temps puis, en jouant avec la PWM, toute une gamme de teintes entre vert et rouge. Il a fallu cependant attendre l'arrivée des leds bleues pour compléter la palette et pouvoir enfin obtenir toute la gamme de couleurs par synthèse additive. Ce procédé, consistant à combiner les lumières de plusieurs intensités de couleurs pour obtenir une teinte précise, est inverse de la synthèse soustractive qu'on utilise, par exemple, en peinture ou en impression. Ici, cyan, magenta et jaune sont mélangés et combinés pour obtenir une gamme complète. En imprimerie comme en peinture, on ajoute généralement le noir pour former un mélange CMJN (ou CMYK en anglais pour *Cyan Magenta Yellow Key*). Notez au passage que les éléments rouges dans ce magazine ne proviennent pas d'un pigment rouge, pas plus que le vert. C'est un mélange de magenta et de jaune, et un mélange de cyan et de jaune. Le noir est nécessaire, car très difficile à obtenir à partir des trois autres couleurs (arriver à un noir acceptable avec de la gouache par exemple peut vous occuper tout un week-end).

Le plus amusant dans tout cela, et pour la synthèse additive en particulier, est le fait que ceci n'est possible qu'en raison de la façon



La fameuse WS2812B telle que vendue sur circuit imprimé et accompagnée d'un condensateur. On voit ici clairement le circuit intégré dans le composant ainsi que les connexions internes aux leds rouges, vertes et bleues.

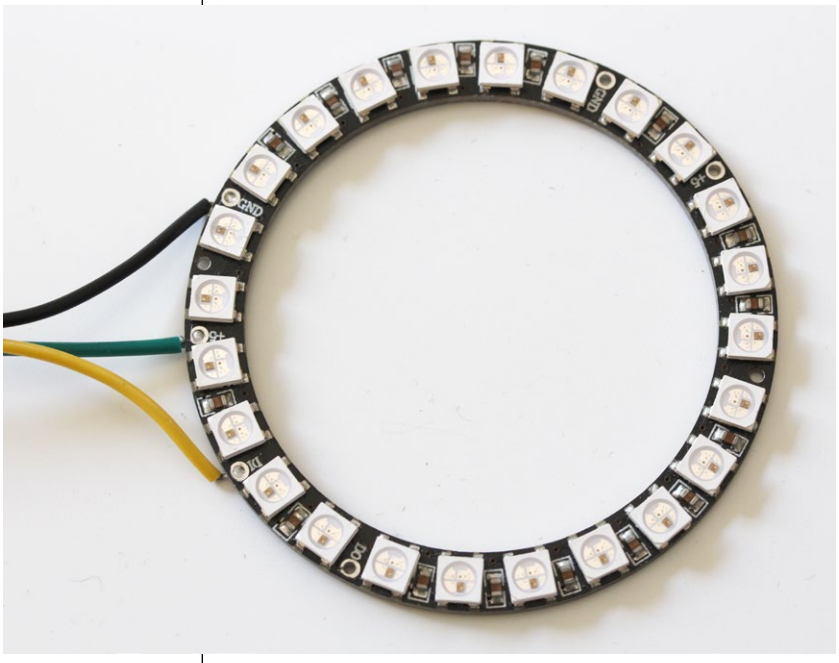
dont nous, humain, percevons les couleurs. Le fait d'obtenir du jaune en combinant des lumières vertes et rouges n'est pas une propriété physique, mais un « défaut » dans le fonctionnement de notre rétine. Celle-ci, en effet, est composée de cônes qui sont des récepteurs réagissant à une longueur d'onde donnée. Une lumière jaune possède une longueur d'onde entre 573 nm et 584 nm (nanomètres). Plus courte c'est du vert, plus grande c'est de l'orange, puis du rouge. Lorsque ce rayonnement rencontre notre rétine, il excite plusieurs types de cônes de façon bien précise. Nous n'avons pas de cône pour le jaune, mais uniquement des types B (437 nm, bleu), des types V (533 nm, vert) et R (564 nm, rouge).

Et là, cela devient très amusant, car si on utilise judicieusement un mélange en bonne quantité de chaque lumière, on arrive à faire croire à la rétine que c'est effectivement une lumière de couleur équivalente qui lui arrive. Autrement dit, en ajustant les quantités de rouge, de vert et de bleu, on peut faire croire à l'œil qu'il voit du jaune, de l'orange, du rose ou encore du violet. Pour le rouge et le vert qui se combinent, on ne voit donc pas de jaune, mais l'œil réagit de la même manière : on a la sensation de voir du jaune.

Vous trouvez ça fou ? Pourtant ce n'est pas nouveau et surtout c'est très largement utilisé partout autour de nous. Votre téléviseur, votre écran de PC,

coromonadalix

votre smartphone, le projecteur dans le salon... Tous ces appareils vous trompent, car aucun n'émet réellement un rayonnement entre 573 nm et 584 nm. Méfiez-vous ! Ils essayent tous de vous faire croire que vous voyez du jaune alors que ce n'est que du vert et du rouge !



Un clone chinois du NeoPixel Ring 24 d'AdaFruit Industries. Nous avons ici 24 leds WS2812B chaînées sur un circuit imprimé circulaire. Inutile de se creuser la tête bien longtemps pour trouver des idées de projets avec ce type de modules. L'anneau existe également en versions 12, 16 et 60 leds.

Quoi qu'il en soit, le fait de pouvoir obtenir une gamme complète de teintes à partir de ces trois couleurs n'est pas réservé aux appareils vendus dans le commerce. Les leds RVB (en anglais RGB pour *Red Green Blue*) sont composées de trois blocs de semi-conducteurs émettant une lumière d'une longueur d'onde différente lorsqu'un courant les traverse. La led en question comporte donc quatre pattes, une commune (cathode ou anode selon le type de led) et une par couleur. En utilisant la technique de modulation en largeur d'impulsion (PWM), on arrive à jouer sur l'intensité de chaque lumière en ajustant les rapports cycliques et donc à produire la couleur de son choix, même si le blanc est parfois difficile à obtenir en fonction du positionnement des blocs dans la led.

Le problème ici est bien entendu le fait de devoir gérer trois signaux PWM par led (ou le multiplexage) et donc de faire appel soit à un microcontrôleur soit à un circuit intégré dédié. Et voici qu'arrive la nouveauté et une très bonne idée : intégrer directement ce circuit de gestion dans la led elle-même et donc créer une led intelligente (*smart led*) qui reçoit des instructions et se charge de presque tout le travail.

1. WS2812B, UN PEU PLUS QU'UN UNIQUE MODÈLE

Parmi les différents modèles de smart leds existant, il en est un dont le nom revient souvent : WS2812. On retrouve cette dénomination dans presque toutes les descriptions de produits équivalents sur eBay par exemple et ce, quelle que soit la forme de l'objet : simples leds, leds sur support (*breakout board*), barrettes de leds, rubans de leds adressables, matrices, etc.

Mais en réalité, les choses sont un peu plus complexes. Avant tout, faisons un point sur quelle désignation correspond à quoi :

- WS2811 n'est pas une led, mais un circuit intégré de gestion de leds ;
- WS2801 est aussi une puce, mais contrairement à la WS2811, celle-ci utilise une liaison avec deux fils DIN et SCK, les données et le signal d'horloge ;
- WS2812 est une led RVB intégrant une puce proche de la WS2811, pilotable avec un seul signal (pas de CLK), mais à une fréquence de 800 KHz et non 400 Khz comme la WS2811 ;

- et enfin, la WS2812B qui est également une led avec une puce intégrée, mais qui est une évolution compatible de la WS2812 ajoutant diverses améliorations et protections. Vous l'avez compris, ce modèle est précisément celui généralement préconisé et traité ici.

Tous ces éléments sont fabriqués par la société WorldSemi basée à Guangdong en Chine et le site du constructeur en référence encore bien d'autres modèles. Ce a quoi il faut ajouter les incontournables leds et puces compatibles, dont celles de BaiCheng proposant un format de led différent (5mm et 8mm) sous le nom PL9823-F8, pilotables avec le même protocole que les WS2812 et WS2812B. Un autre exemple, plus « évolué » est la APA102©, mais utilisant un protocole différent puisqu'il s'agit de SPI (avec un signal d'horloge supplémentaire donc). Ce modèle sort du cadre de cet article, car se pilotant de manière totalement différente.

Les WS2812B se présentent sous la forme d'un composant à souder en surface (SMD) de 5mm de côté (format communément appelé 5050 PLCC4). Elles sont généralement distribuées par des boutiques en ligne, des sites d'achats directs (type Alibaba) ou sur eBay. Il ne sera pas forcément nécessaire de vous diriger vers un vendeur en Asie et donc d'attendre un temps non négligeable pour la livraison. Certains distributeurs situés en Allemagne ou en Angleterre commercialisent des packs de 100 WS2812B pour moins de 30 euros, port gratuit (kt-elektronic). Un petit conseil au passage, la led WS2812B se différencie de la WS2812, plus fragile, par son nombre de connecteurs, 4 pour le B contre 6 pour l'ancienne WS2812. Vérifiez bien le descriptif du produit et les photos, les erreurs et tromperies sur la marchandise ne sont pas courantes, mais le risque est réel.

Cependant, à moins de savoir précisément ce que vous faites, je vous recommande de tourner votre attention non pas vers les composants bruts (*bulk*), mais vers leurs déclinaisons montées sur un petit circuit imprimé ou moulées dans un ruban sécable.

Les WS2812B sont des composants fragiles et à traiter avec soin, et le modèle WS2812 est encore plus sensible, une simple inversion de polarité suffit à le détruire. Ainsi, à la différence des composants vendus seuls, les versions montées sont accompagnées d'un condensateur de découplage et d'une résistance de protection qui sont plus que recommandés (en plus d'utiliser le support comme dissipateur thermique). Tout cela en restant de taille réduite et dans un prix acceptable. J'en ai trouvé sur eBay aux tarifs de 200 pour 70 euros, 100 pour 40 euros, 30 pour 15 euros, 10 pour 5 euros... auxquels s'ajoute une dizaine d'euros de port (depuis l'Angleterre, vendeur *transnova*).

Notez que les WS2812 et WS2812B sont souvent désignés sous le nom « NeoPixel » qui est utilisé et popularisé principalement par le site AdaFruit proposant

Gros plan sur une partie du NeoPixel Ring. On distingue un des condensateurs de découplage à proximité d'une des leds. Il faudra cependant toujours ajouter un autre condensateur entre la masse et Vcc au niveau de l'alimentation. Précisons qu'il est hors de question d'alimenter 24 WS2812B avec une carte Arduino.

coromonadalix

ses propres déclinaisons sous forme de rubans, d'anneaux (12, 16, 24, 60 leds), de shields Arduino, etc., s'accompagnant d'une bibliothèque NeoPixel et d'un guide en ligne. Les tarifs cependant sont loin d'égaler les offres des sites d'enchères, mais le niveau de qualité de la fabrication n'est pas le même. L'alignement des WS2812B sur un anneau de 24 leds à 15 euros provenant d'eBay, par exemple, est assez aléatoire par rapport aux produits similaires chez AdaFruit.

2. COMMENT COMMUNIQUER AVEC UN SEUL FIL ?

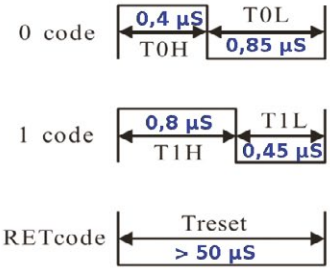
Les WS2812B et donc les circuits, rubans et barrettes qu'elles équipent disposent d'uniquement quatre connecteurs : l'alimentation en +5V, la masse, les données en entrée (DIN) et les données en sortie (DOUT). Il n'y a donc pas de signal d'horloge permettant de valider l'arrivée d'un bit. En prenant le cas d'une seule led, un connecteur suffit donc à transmettre au circuit intégré les informations sur l'intensité lumineuse des trois leds, rouge, verte et bleu, placées dans le composant.

Comme pour les registres à décalage et d'autres circuits logiques similaires, il est possible de chaîner les WS2812B en reliant le DOUT du premier composant au DIN du suivant, etc. Ainsi, en envoyant la bonne quantité de bits, il est possible de définir la couleur et l'intensité de tous les WS2812B de la chaîne. La quantité de composants qu'il est ainsi possible de chaîner n'est pas spécifiée dans la documentation, très concise et vaguement anglaise. Ce n'est pas un problème logique, mais physique qui limite la quantité, par la distance même que doit parcourir l'information. Ce à quoi s'ajoutent, bien entendu, les problèmes d'alimentation puisque chaque composant comprend trois leds de forte puissance. Il est très facile de dépasser le watt, voire la dizaine de watts (~100 leds, ce qui semble beaucoup, mais finalement n'est qu'un carré de 10×10). Enfin, il faut prendre en compte les paramètres temporels. Comme l'envoi de l'information est basé sur la durée des impulsions, le temps nécessaire pour envoyer les 24 bits à chaque led est fixe. Entre 1,2 µS et 1,3 µS par bit et donc une trentaine de microsecondes par led, ce à quoi il faut ajouter le temps de calcul nécessaire côté Arduino. Si vous comptez générer une animation sur *n* leds, ces délais impacteront la fréquence de rafraîchissement, car il faut que les informations aient le temps d'arriver à la dernière led avant de commencer un nouveau cycle.

La documentation du composant détaille comment une telle communication sur une broche peut être réalisée. Celle-ci parle de NZR qu'on pourrait interpréter comme étant une faute de frappe

Extrait amélioré de la documentation de Worldsemi à propos de leur WS2812B. On comprend ici clairement qu'envoyer 1 bit prend 1,25 µS et chaque led utilise 24 bits, ce qui donne un temps de 30 µS par composant.

Data transfer time(TH+TL=1.25µs±600ns)



T0H	0 code ,high voltage time	0.4us
T1H	1 code ,high voltage time	0.8us
T0L	0 code , low voltage time	0.85us
T1L	1 code ,low voltage time	0.45us
RES	low voltage time	Above 50µs

coromonadalix

sur l'acronyme NRZ comme *Non-Return to Zero* (un type de codage utilisé avec RS232 par exemple), mais en réalité, ceci se rapproche plus d'un encodage basé sur une sorte de PWM. Un bit 0 est 0,4 µS (micro seconde) à l'état haut, suivi de 0,85 µS à l'état bas. Un bit 1 est encodé par 0,8 µS à l'état haut et 0,45 µS à l'état bas. Un signal de plus de 50 µS à l'état bas est considéré comme un reset. Ces différents états sont nommés respectivement T0H, T0L, T1H, T0L et Treset.

Pourquoi vous raconter tout cela puisque, vous vous en doutez, nous ne comptons pas piloter directement les WS2812B, mais utiliser une bibliothèque ? C'est simple, la communication repose intégralement sur des signaux d'une durée donnée (+/- 150 nanosecondes) qui doit impérativement être respectée. Ceci signifie entre autres choses que tout ce qui peut perturber la communication et impacter ces délais provoquera des problèmes, à commencer par la longueur des fils. Un environnement riche en nuisance électromagnétique (tubes fluorescents), une mauvaise alimentation, l'absence de condensateur de découplage... sont autant de choses qui risquent non seulement de perturber le fonctionnement de ces leds, mais peuvent aussi les détruire.

Comme le décrit admirablement Fabien Batteix sur son blog Skyduino (coucou au passage), il est de bon ton de prendre soin des WS2812B. Comme des lemmings, ces composants ne semblent en effet vouloir qu'une chose : mourir (concernant les lemmings c'est absolument faux, ils ne sont pas suicidaires, mais juste complètement stupides lorsqu'ils sont en masse (comme les humains)).

Il faut impérativement donc prendre en compte les faits suivants. Les WS2812B :

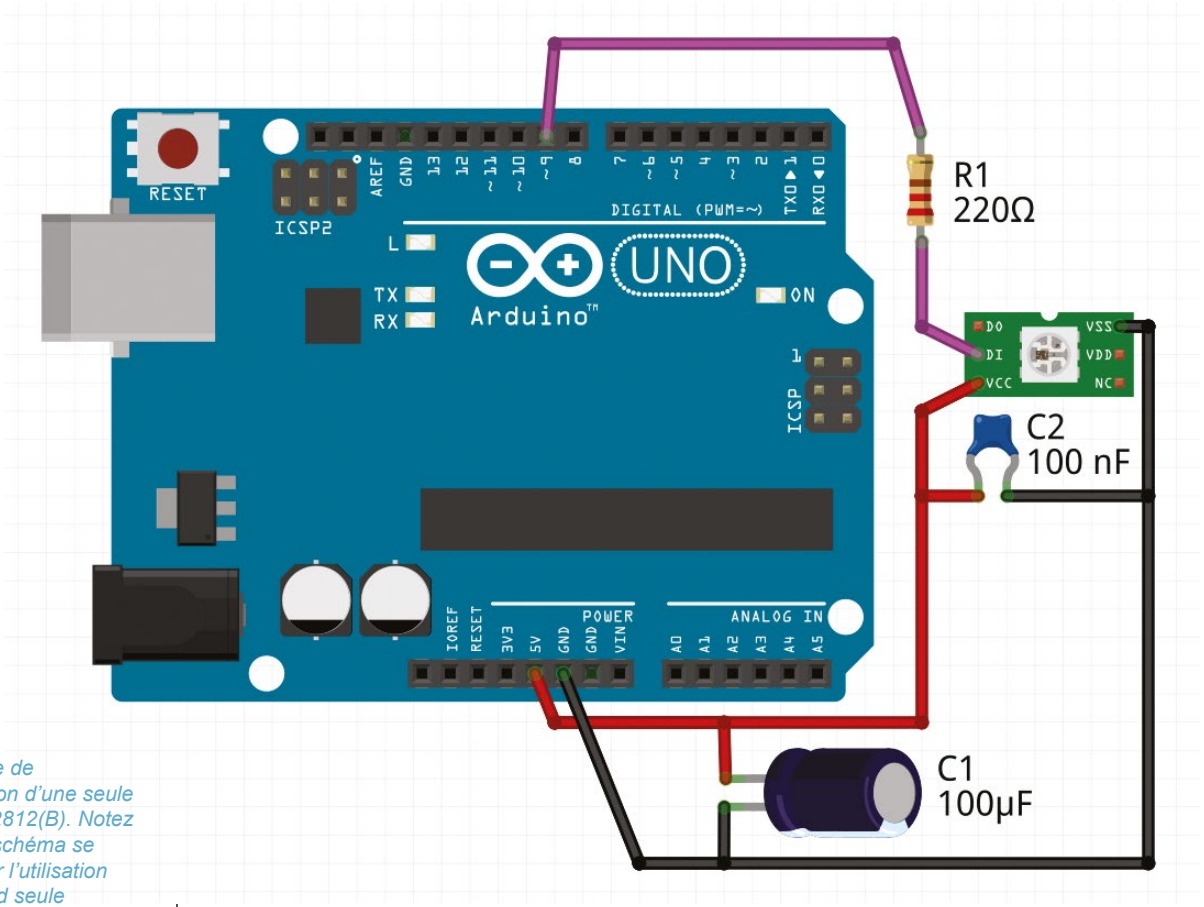
- sont hypersensibles à l'électricité statique ;
- ne supportent pas l'humidité ;
- doivent être alimentées en 5V précisément ;
- doivent être équipées d'une manière ou d'une autre d'un système de dissipation thermique ;
- ne supportent pas les alimentations non stables (il faut des condensateurs partout, et par led) ;
- ne supportent pas une inversion de polarité (ce qui est réglé avec le modèle B en principe) ;
- ne doivent pas recevoir de courant sur DIN si la led n'est pas alimentée ;
- etc.

En utilisant des leds montées sur platine ou en ruban, vous réglez potentiellement une partie du problème puisque ces circuits intègrent un condensateur entre l'alimentation et la masse ainsi qu'une résistance sur DIN pour limiter le courant. Les rubans sont souvent moulés dans une résine souple qui met les leds à l'abri de l'humidité, mais ne règle pas le problème de dissipation thermique. Les platines elles, servent de dissipateur thermique (dans une certaine mesure), mais ne règlent pas le problème lié à l'humidité. Selon votre projet, il faudra donc peser le pour et le contre de chaque solution et surtout respecter les besoins des composants en toute occasion. Tester un ruban de 5m avec 150 leds avec une intensité maximum pour chaque couleur et le tout enroulé dans son dévidoir est une très, très, très mauvaise idée !

3. PASSONS À LA PRATIQUE

La connexion d'une led ou une série de leds à une carte Arduino (ou autre) est relativement simple. N'importe quelle sortie numérique peut être utilisée pour relier la carte à DI. Il n'est pas nécessaire que celle-ci dispose de fonctionnalités « analogiques » (PWM et contrôlées par *timer*). Le seul élément important concerne les recommandations que nous avons déjà précisées : gardez les fils les plus courts possible, ajouter une résistance de protection et assurez-vous de filtrer correctement l'alimentation avec des condensateurs de découplage.

coromonadalix



Exemple de connexion d'une seule led WS2812(B). Notez que ce schéma se base sur l'utilisation d'une led seule montée sur platine. De ce fait, une résistance de protection est ajoutée sur DI, un gros condensateur (100 µF) stabilise la ligne d'alimentation et un plus petit (100 nF), plus près de la led, ajoute une protection contre les perturbations électromagnétiques (condensateur de découplage). R1 et C2 peuvent se trouver intégrés directement sur le circuit, le ruban ou la platine selon le produit acheté.

Connecter plusieurs leds en série se fait selon le même principe, il suffit de relier le DO d'une led au DI de la suivante. Gardez toutefois à l'esprit que plus vous avez de leds, plus il y aura besoin de courant. Connecté en USB, un montage dans sa totalité ne pourra pas utiliser plus de 500mA. De la même manière, le régulateur de tension monté sur les cartes Arduino possède également ses limites dépendantes de la tension en entrée, de la dissipation thermique et du courant circulant. Pour des montages utilisant une dizaine de leds WS2812B ou plus, il est recommandé (sinon nécessaire) d'utiliser une alimentation adaptée et de qualité, exactement comme pour le pilotage de leds de forte puissance (cf article sur l'émission infrarouge dans *Hackable n°3*).

Côté logiciel, il existe plusieurs bibliothèques en mesure de gérer une ou plusieurs leds WS2812B ou compatibles. La plus complète et la plus utilisée est sans doute celle d'AdaFruit Industries disponible sur GitHub (**https://github.com/adafruit/Adafruit_NeoPixel**) sous le nom *Adafruit NeoPixel* (référence aux produits correspondants chez ce distributeur). L'un des avantages de cette bibliothèque, bien que provenant d'un vendeur de composants, est le fait qu'il est possible de l'utiliser avec plusieurs composants.

En effet, si nous prenons l'exemple des leds WS2812B et PL9823, bien que relativement similaires, les deux composants possèdent une différence notable. La WS2812B attend une suite de bits sous la forme **GGGGGGGGRRRRRRRRBBBBBBBB** avec **G**, **R** et **B** correspondant respectivement à l'intensité du vert, du rouge et du bleu, soit trois paquets de 8 bits GRB. La PL9823 en revanche utilise le format **RRRRRRRRGGGGGGGGBBBBBBBB** ou, en d'autres termes, trois paquets de 8 bits RGB. Utiliser une PL8323 avec une bibliothèque ne gérant que la WS2812B fonctionnera, mais les couleurs seront fausses.

coromonadalix

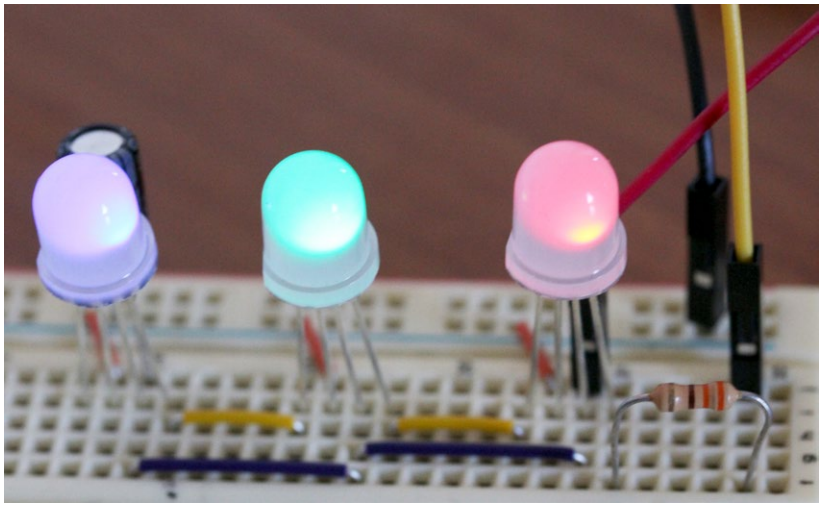
L'autre différence pouvant exister entre les composants (leds ou circuits de pilotage seuls) concerne les délais utilisés pour représenter les bits. Certains composants (WS2812B, WS2812, PL9823) utilisent un *timing* permettant d'atteindre 800 Kb/s, alors que d'autres comme le WS2811 (qui est un contrôleur sans led) utilisent une vitesse de 400 Kb/s.

La bibliothèque Adafruit NeoPixel est heureusement en mesure de gérer ces différentes combinaisons (et même le format BRG) et ce, pour la totalité des cartes Arduino utilisant un AVR (plus l'Arduino Due et quelques cartes Teensy de PJRC). Il ne vous sera pas possible, en revanche, d'utiliser la bibliothèque avec une plateforme comme une MSP Launchpad de TI, car la bibliothèque, évitant soigneusement d'utiliser les *timers*, repose sur des instructions en assembleur *inline* (de l'assembleur embarqué dans le code C/C++). Bien évidemment, qui dit « assembleur » dit « jeu d'instructions » et donc implicitement une dépendance complète à une famille précise de microcontrôleurs (pour les développeurs : sans surprise, la bibliothèque est bourrée de `ifdef`). Il existe cependant des portages/adaptations de la bibliothèque disponibles pour TI MSP430, STM32, cartes compatibles mbed ou encore Microchip PIC.

3.1 Premier exemple : la base

Le plus simple des exemples est le « helloworld » de l'électronique : allumer une led. Ceci sera également la base de l'utilisation de la bibliothèque :

```
#include <Adafruit_NeoPixel.h>

// création de "pixel" représentant
// notre chaîne de leds
// (ici, une chaîne de 1 led)
Adafruit_NeoPixel pixels =
  Adafruit_NeoPixel(1, 9, NEO_GRB + NEO_KHZ800);

void setup() {
  // mise en route
  pixels.begin();
}

void loop() {
  // led en position 0 :
  // 50 % de rouge, pas de vert, pas de bleu
  pixels.setPixelColor(0, pixels.Color(128,0,0));
  // envoi
  pixels.show();
}
```



Pour mon expérimentation, j'ai préféré utiliser des leds PL9823 dont le pilotage est quasi-identique à celui des WS2811B. Une led, même intelligente, qui a la forme d'une led c'est tout de même plus sexy, surtout en 8 mm.

coromonadalix

La chaîne de leds (ici une seule led) est représentée par la variable `pixel` de type `Adafruit_NeoPixel`. Celle-ci est initialisée grâce à la fonction éponyme en précisant, en argument, le nombre de leds chaînées, le port utilisé et le type. `NEO_GRB` correspond à un encodage vert/rouge/bleu adapté au modèle WS2812B. Pour des PL9823, on utilisera `NEO_RGB`. Il n'est pas possible, bien entendu, de mélanger WS2812B et PL9823 sur une même chaîne, mais il est parfaitement possible de créer autant de variables `Adafruit_NeoPixel` que nous avons de chaînes sur des ports différents. `NEO_KHZ800` précise la vitesse utilisée (800Khz) et une autre option possible est `NEO_KHZ400` (pour les WS2811+leds par exemple).

La fonction `setup()` se contente d'initialiser la chaîne avec `begin()`. Ceci a pour effet de préparer les attributs privés et publics de la classe, mais réinitialise également toutes les leds. En effet, lors de la mise sous tension, l'état des leds et donc la couleur et l'intensité lumineuse sont des éléments indéterminés (même si mes PL9823 semblent systématiquement s'allumer en bleu). Ainsi, passé `setup()`, toutes leds doivent être éteintes.

Notre fonction `loop()` se contente d'envoyer les données à notre unique led via la méthode `setPixelColor()`. Celle-ci prend en argument la position de la led ciblée (ici `0`, la première et la seule) et les données formatées spécifiquement par la méthode `Color()`. Cette méthode prend en argument les valeurs de rouge, de vert et de bleu. Ceci peut paraître complexe, mais présente l'avantage d'avoir une seule fonction de définition de couleur, au format RVB/RGB, indépendamment de l'encodage utilisé réellement par le composant.

Mais ce n'est pas tout, `setPixelColor()` ne provoque pas le changement de couleur sur le ou les composants. Cette méthode se contente de changer une représentation interne des données. Il faut donc « pousser » ces données sur les leds. C'est le travail de la méthode `show()`.

Sur cette base, vous pouvez déjà explorer les différentes possibilités et, par la même occasion, découvrir l'aspect ludique de la synthèse additive. Vous remarquerez aussi sans doute les limites physiques du procédé. `pixels.Color(32,32,32)` ne vous donnera pas un blanc de faible intensité, mais une espèce de mélange douteux entre le violet clair et le rose. Ceci provient à la fois de la conception même des leds, mais également de la sensibilité de l'œil humain à certaines longueurs d'onde et donc certaines couleurs (le vert/jaune à 555nm est extrêmement bien perçu alors que le rouge ou le bleu nécessitent une plus grande intensité). Un bon exercice consiste à comparer les couleurs HTML format `#RRVVBB` avec les mêmes valeurs appliquées à une led (en commençant par le jaune).

3.2 Jouons un peu

Une led c'est bien amusant, mais puisqu'on peut en ajouter sans que cela ne nous coûte des ports supplémentaires, pourquoi se priver ? Voici donc une petite déclinaison à 3 leds sur la base précédente, mais ajoutant un peu d'animation aléatoire :

```
#include <Adafruit_NeoPixel.h>

// trois leds sur la chaîne
Adafruit_NeoPixel pixels =
  Adafruit_NeoPixel(3, 9, NEO_GRB + NEO_KHZ800);

void setup() {
  pixels.begin();
  // initialisation du générateur de nombres
  randomSeed(analogRead(0));
}

void loop() {
  // première led
  pixels.setPixelColor(0, pixels.Color(
    random(255),random(255),random(255)));
  // seconde led
  pixels.setPixelColor(1, pixels.Color(
    random(255),random(255),random(255)));
  // troisième led
```

```
    pixels.setPixelColor(2, pixels.Color(
      random(255),random(255),random(255)));
    // envoi
    pixels.show();
    // pause
    delay(1000);
}
```

Cette fois, nous laissons le hasard décider des couleurs. Après avoir initialisé le générateur de nombres pseudo-aléatoire à partir d'une valeur lue sur la broche **A0** (non connectée), nous pouvons utiliser la fonction **random()** en précisant en argument la valeur maximale (moins 1) que nous tolérons. Notre fonction **loop()** se charge de décider seule des valeurs de rouge, de vert et de bleu pour les leds de 0 à 2.

Notez le fonctionnement global et l'intérêt de disposer de la méthode **show()**. Nous préparons nos données et ensuite seulement mettons à jour la couleur des leds en un seul mouvement.

Vous trouvez que c'est un peu éprouvant pour votre rétine ? Pas de problème, ajoutez simplement une ligne **pixels.setBrightness(32)**à la fin de la fonction **setup()** et la luminosité globale (théorique) en sera divisée par 8 (256/32). Cette méthode prend en argument une valeur entre 0 et 255 permettant de régler l'intensité lumineuse pour l'ensemble des leds. La bibliothèque se charge, en interne, d'adapter les valeurs effectives RVB que vous spécifiez en fonction de la luminosité choisie.

3.3 Vers une gestion plus pratique des couleurs

À ce stade, nous sommes en mesure d'ordonner à chaque led de prendre la couleur de notre choix et ce, avec l'intensité voulue. Quelques couleurs sont relativement faciles à obtenir avec 255 en guise de valeur :

- rouge ;
- vert ;
- bleu ;
- jaune/orange : rouge + vert ;
- cyan ou bleu verdâtre : vert + bleu ;
- magenta/rose : rouge + bleu ;
- blanc rosâtre : rouge + vert + bleu.

Pour le reste des couleurs intermédiaires, c'est un peu plus délicat, car il faut jouer sur l'intensité de chaque composante RVB. Pour de l'orange par exemple, 255 de rouge et 128 de vert, fonctionneront assez bien. Il est donc possible de se créer une palette, peut-être sous la forme d'un tableau stocké en flash, et ainsi piocher les valeurs pour chaque nuance souhaitée. Reste cependant à construire cette palette et cela, soyons clairs, relève en grande partie du tâtonnement. On pourrait utiliser des potentiomètres reliés aux entrées analogiques de l'Arduino pour régler manuellement les quantités R, V et B. En ajoutant un bouton provoquant l'affichage des trois valeurs sur le moniteur série, on disposera d'une solution pour créer plus facilement une palette.

Mais il existe une meilleure solution et pour en mesurer tout l'intérêt, imposons-nous un cahier des charges simple : piloter une led de manière à ce qu'elle parcoure de manière fluide toutes les couleurs de l'arc-en-ciel, dans l'ordre (si déjà). Avec une led qui peut afficher quelques 16 millions de couleurs (256 x 256 x 256) et donc 16 millions de combinaisons de valeurs RVB sur 24 bits en tout, la palette risque d'être un peu difficile à ajuster...

La bonne solution consiste à ne pas utiliser le modèle RVB, mais TSV pour Teinte Saturation et Valeur (HSV en anglais). C'est un modèle très pratique qu'on retrouve systématiquement dans les logiciels de création graphique par exemple. Comme pour RVB, TSV fait correspondre une valeur par élément contrôlé :

- Teinte : une valeur sur 360 correspondant à un angle dans le cercle des couleurs ou cercle chromatique avec le rouge à 0° (ou 360°), le vert à 120° et le bleu à 250°. Il suffit alors de choisir un angle pour sélectionner une teinte. Et comme c'est un cercle, on peut tourner autour à loisir ;

coromonadalix

- Saturation : est exprimée en proportion, généralement sur cent ou 256, et fixe l'intensité de la couleur. Le terme « désaturation » est souvent utilisé pour décrire une image fade/délavée ou comme technique pour passer une image en niveaux de gris ;
- Valeur : c'est la brillance d'une couleur, également exprimée en proportion. Pour comprendre l'intérêt de la V en TSV, il suffit de se rappeler que plus V est petit, plus la couleur est sombre. Avec la valeur à zéro, on obtient du noir, quelles que soient la teinte et la saturation.

Ce qui nous intéresse ici est bien entendu surtout T. En disposant d'un algorithme de conversion TSV vers RVB, nous pouvons alors pousser la saturation au maximum ainsi que la valeur et parcourir le cercle de couleurs de 0 à 359.

Il existe des centaines d'implémentations de l'algorithme de conversion dans tous les langages possibles et imaginables. En voici une en C parfaitement adaptée à un code pour Arduino (que des entiers, pas de **float** et donc une économie notable de flash) :

```
void HSVtoRGB(int  *r, int *g, int
*b, int h, int s, int v) {
  int c;
  long l, m, n;

  // saturation zéro, pas de couleur
  if(s == 0) {
    *r = *g = *b = v;
    return;
  }

  // chroma
  c = ((h%60)*255)/60;
  h /= 60;

  // intermédiaire
  l = (v*(256-s))/256;
  m = (v*(256-(s*c)/256))/256;
  n = (v*(256-(s*(256-c))/256))/256;

  // choix dominante
  switch(h) {
    case 0:
      *r = v; *g = n; *b = l;
      break;
    case 1:
      *r = m; *g = v; *b = l;
      break;
    case 2:
      *r = l; *g = v; *b = n;
      break;
    case 3:
      *r = l; *g = m; *b = v;
      break;
    case 4:
      *r = n; *g = l; *b = v;
      break;
    default:
      *r = v; *g = l; *b = m;
      break;
  }
}
```

Notez que toutes les couleurs décrites par le modèle TSV ne peuvent être représentées en RVB, mais c'est un point de détail ici puisque nous travaillons de toute façon avec une led qui physiquement fonctionne en RVB.

Cette fonction prend en argument des pointeurs sur les variables contenant les valeurs de rouge, de vert et de bleu, ainsi que les valeurs TSV. Sans compliquer l'article plus que nécessaire en déviant de notre objectif, limitons-nous à son utilisation. Après avoir ajouté cette déclaration de fonction dans notre croquis, nous pouvons l'utiliser. Notre **loop()** devient alors :

```
void loop() {
  // valeur RVB
  int r, v, b;
  // on tourne dans le cercle de couleurs
  for (int i = 0; i < 360; i++) {
    // TSV = position, 100% S et 100% V
    HSVtoRGB(&r, &v, &b, i, 255, 255);
    // choix de la couleur de la première led
    pixels.setPixelColor(0, pixels.Color(r, v, b));
    // affichage
    pixels.show();
    // courte pause pour animation
    delay(15);
  }
}
```

coromonadalix

Notre boucle `for` va incrémenter `i` de 0 à 359. Nous utilisons cette variable avec notre fonction toute neuve. Notez le `&` précédant les noms de variables `r`, `v` et `b`. Nous spécifions ici les adresses (pointeurs) auxquelles se trouvent ces variables et non leur contenu. Ainsi la fonction `HSVtoRGB()` pourra les utiliser pour leur affecter une valeur. Après appel à `HSVtoRGB()`, les trois variables contiennent les valeurs de rouge, vert et bleu que nous pouvons directement utiliser avec la bibliothèque NeoPixel.

Le résultat sera exactement celui attendu. Nous définissons successivement 360 nouvelles couleurs en tournant d'un degré à chaque fois dans le cercle. Arrivés à 359, on sort de la boucle `for` et on repart pour un tour de `loop` sans le moindre problème puisque 360 et 0 correspondent exactement à la même couleur/position. Notre led va donc, en à peu près 6 secondes, faire en douceur le tour des 360 couleurs.

CONCLUSION

Arrêtons là les expérimentations, car cet article, sinon, pourrait bien finir par occuper l'ensemble du magazine (voir tout ou partie du numéro suivant). L'algorithme proposé pour la conversion TSV vers RGB n'est qu'un exemple, il existe même quelques bibliothèques incluant cette fonctionnalité. Cependant, quelle que soit la solution que vous adopterez, le fait de travailler en TSV facilite énormément la mise en œuvre de ce type de leds dans un projet réel. Voici quelques exemples :

- En n'utilisant que les positions entre 120 et 0 dans le cercle de couleurs, nous obtenons un dégradé entre le vert et le rouge en passant par le jaune et l'orange : idéal pour indiquer un niveau de gravité, d'importance ou de température. En trichant un peu, on peut utiliser 128 positions et nous faciliter la tâche pour directement utiliser des valeurs sur 8, 9 ou 10 bits avec une simple division.
- En enregistrant une palette de couleurs basée sur la teinte, non seulement on économise de la mémoire, mais il devient possible de créer des transitions douces. Ainsi, plutôt que de passer brutalement du cyan au rouge on peut, à l'aide d'une simple boucle, glisser vers la couleur cible. Écrire une fonction dédiée pouvant utiliser n'importe quelle couleur cible (et source) ne demande que peu de travail.
- En utilisant plusieurs leds chaînées, il devient simple de créer un « effet de persistance » avec n'importe quelle couleur. La première led verra sa valeur poussée au maximum puis proportionnellement, on diminuera cette valeur pour les leds suivantes tout en conservant la même teinte et la même saturation. Résultat, un magnifique dégradé d'intensité.
- Il devient bien plus simple de contrôler la couleur d'une led avec une entrée analogique. Un seul potentiomètre connecté à la carte Arduino fixera l'angle (teinte) à utiliser et donc la couleur. Il sera possible d'en ajouter un second pour la saturation et ainsi accéder à des lumières plus « pastel » et au blanc.

Une prise en charge ainsi rendue simple par le duo led intelligente + TSV permet d'apporter beaucoup d'informations dans une « interface utilisateur ». Il deviendra même possible dans certains cas de remplacer purement et simplement un afficheur LCD. Ces composants sont, à mon sens, une véritable aubaine malgré leurs fragilités.

Allez, une dernière idée de projet pour la route : vous avez toujours voulu un réveil simulateur d'aube, mais sans ces horribles et pénibles sons d'oiseaux guillerets ? Une carte Arduino, une grosse alimentation, une RTC DS1307 et quelques rubans de WS2812B et voici votre chambre à coucher devenue elle-même le réveil de vos rêves... **DB**

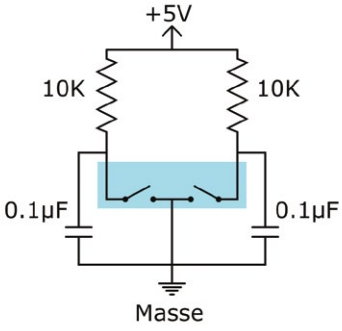
coromonadalix

SUR LE POUCE : AJOUTEZ UN CODEUR ROTATIF

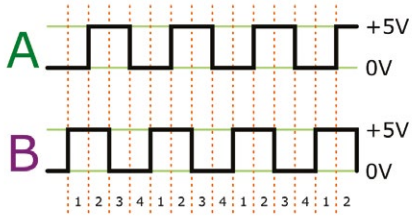
Denis Bodor

Potentiomètres, boutons poussoirs, interrupteurs, surfaces capacitives, LDR... Il existe beaucoup de solutions pour interfacer un humain avec une carte Arduino. Chacune d'elles possède ses avantages en fonction du projet en cours. Ici, nous avons dans l'idée d'ajouter à notre système une méthode de contrôle de la teinte des WS2812B. La solution retenue est le codeur rotatif.

Les encodeurs rotatifs sont des éléments mécaniques ne fournissant ni position ni valeur. On ne peut donc pas simplement lire l'état des broches à un instant t pour connaître la position courante. Il faut plutôt les voir comme des incrémenteurs/décrémenteurs. Schématiquement, ils se composent de deux interrupteurs qui, en fonction de leurs états successifs, permettent de connaître le sens de rotation et le nombre de pas.



Il faut donc utiliser deux entrées de l'Arduino pour obtenir les informations nécessaires. Les deux interrupteurs sont activés par la rotation, mais ne fonctionnent pas en même temps. Leurs successions d'états sont décalées l'une par rapport à l'autre. On dit que les signaux en sortie sont en quadrature ou déphasés de 90° (cela vous rappelle quelque chose ? Jetez un œil au premier article sur la radiocommande Futaba).

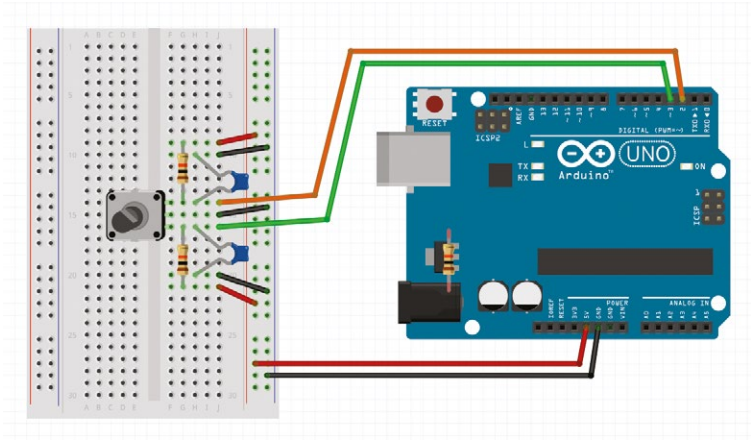


Les sorties d'un codeur rotatif n'ont pas réellement de désignation attitrée. Nous avons simplement un connecteur commun et une broche pour chaque interrupteur interne qu'on désigne arbitrairement par A et B. Comme il s'agit avant tout d'interrupteurs mécaniques, il est important de se rappeler qu'il faut faire face à un phénomène de rebond puisque le changement d'état n'est pas franc. La lame de l'interrupteur rebondie et envoie une succession d'états haut et bas avant de se stabiliser.

Il nous faut donc prévoir une solution pour régler ce problème. Ici, nous choisissons celle du réseau RC (résistance/condensateur) agissant comme un filtre passe-bas. Des résistances de 10 Kohms

coromonadalix

relient A et B à +5V. Elles sont à la fois un élément du filtre et les résistances de rappel à Vcc. Des condensateurs de 0.1µF complètent le circuit. D'autres solutions anti-rebond existent comme le fait de le prévoir dans le croquis ou encore d'utiliser un circuit intégré comme un 74HC14 ou n'importe quel autre circuit logique contenant un trigger de Schmitt.



Il suffit ensuite de relier l'ensemble à l'Arduino, le reste se passe dans le code (je n'ai pas ici repris l'ensemble du croquis, mais uniquement les parties faisant intervenir l'encodeur) :

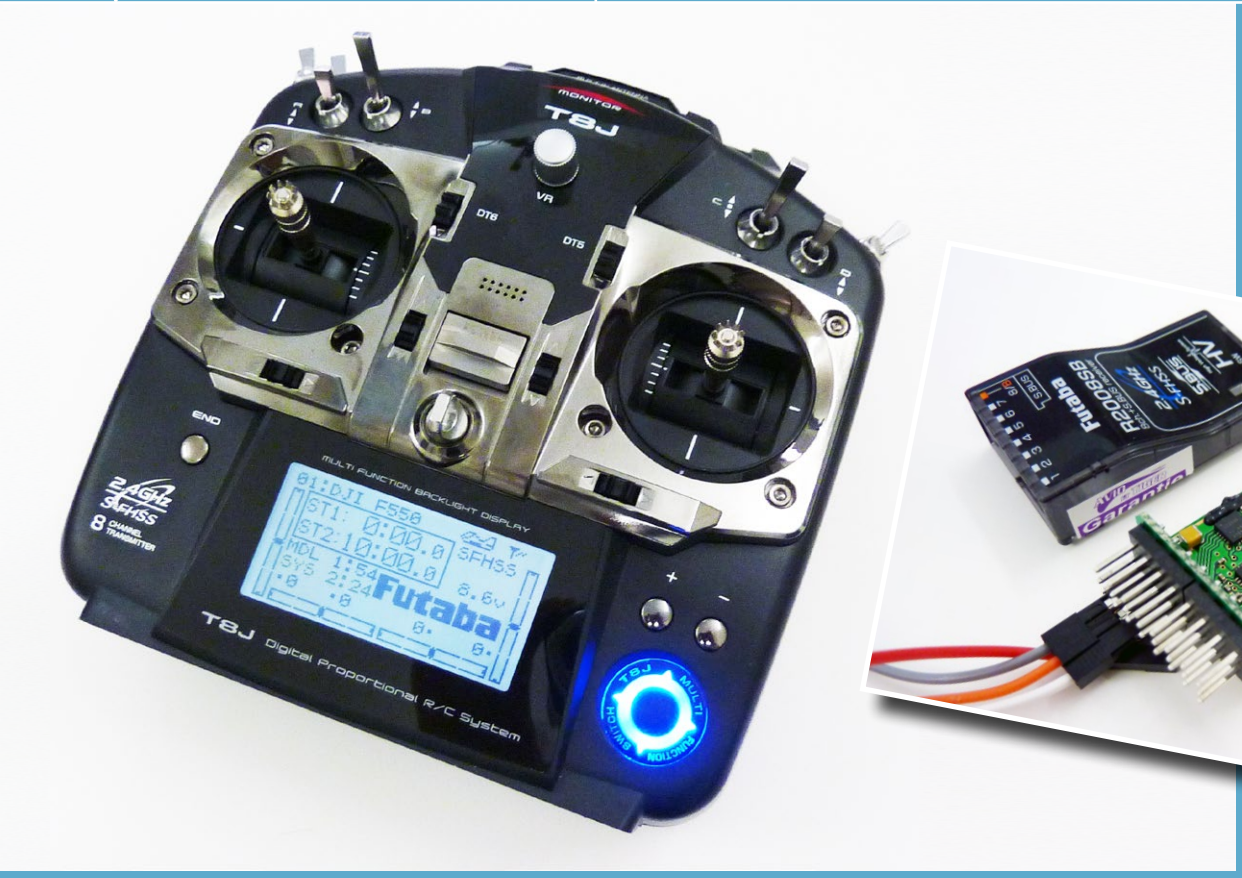
```
// broches utilisées
int brocheA = 3;
int brocheB = 2;
// valeur stockée
int actuelle = 0;
// dernier état de A
int dernièreA = LOW;
// état actuel
int n = LOW;

void setup() {
  // A et B (2 &3) en entrée
  pinMode(brocheA,INPUT);
  pinMode(brocheB,INPUT);
  pixels.begin();
}

// valeur d'incrémentation
int STEP=5;

void loop() {
  int r, v, b;
  // lecture A
  n = digitalRead(brocheA);
  if ((dernièreA == LOW) && (n == HIGH)) {
    if (digitalRead(brocheB) == LOW) {
      // on décrémente
      actuelle-=STEP;
      // on teste pour tourner dans le cercle
      if(actuelle<0) actuelle=359;
      // envoi de la couleur
      HSVtoRGB(&r, &v, &b, actuelle, 255, 255);
      pixels.setPixelColor(0, pixels.Color(r, v, b));
      pixels.show();
    } else {
      // on incrémente
      actuelle+=STEP;
      // on teste pour tourner dans le cercle
      if(actuelle>359) actuelle=0;
      // envoi de la couleur
      HSVtoRGB(&r, &v, &b, actuelle, 255, 255);
      pixels.setPixelColor(0, pixels.Color(r, v, b));
      pixels.show();
    }
  }
  // l'état actuel devient l'état précédent
  dernièreA = n;
}
```

Je vous laisse le soin de suivre la logique en la rapprochant du diagramme de la succession d'états qui peut se lire de gauche à droite et inversement, en fonction du sens de rotation de l'encodeur. Ceci est relativement facile à analyser et constitue un excellent exercice (et je n'ai plus de place, c'est le dernier article du numéro).

Notez que cette approche n'est pas la seule utilisable. En cherchant sur le Web, vous trouverez des croquis utilisant les interruptions et vous pourrez adapter votre code à d'autres utilisations (d'où la connexion aux broches 2 et 3). Mais dans le cas présent, j'ai préféré éviter d'interférer avec les routines de contrôle du *timing* des leds.

Notez que la plupart des encodeurs rotatifs disposent d'un troisième interrupteur, activé par la pression et non par rotation. Deux pattes supplémentaires lui sont dédiées et nous pouvons, avec un autre circuit RC, l'utiliser comme entrée pour, par exemple, passer du contrôle de la teinte à celui de la saturation, puis de la valeur. Il est, bien entendu également possible de multiplier les encodeurs et d'utiliser les pressions pour passer en mode sélection de leds (si vous en pilotez plusieurs). Les idées ne manquent pas... **DB**

coromonadalix



TÉLÉCOMMANDEZ VOS MONTAGES ARDUINO

Denis Bodor

Lorsqu'on aime bidouiller, vous le savez aussi bien que moi, il est généralement difficile de se limiter à un seul domaine d'activité. Peut-être avez-vous comme moi, porté votre intérêt sur le modélisme RC, l'aéromodélisme et/ou les multirotors communément et faussement désignés par le terme « drones ». Il est amusant de remarquer comme ces centres d'intérêt finalement peuvent converger et se mélanger. Vous possédez une télécommande d'aéromodélisme Futaba ? Vous avez un récepteur en trop actuellement inutilisé ? Pourquoi ne pas vous en servir pour communiquer avec votre Arduino ?

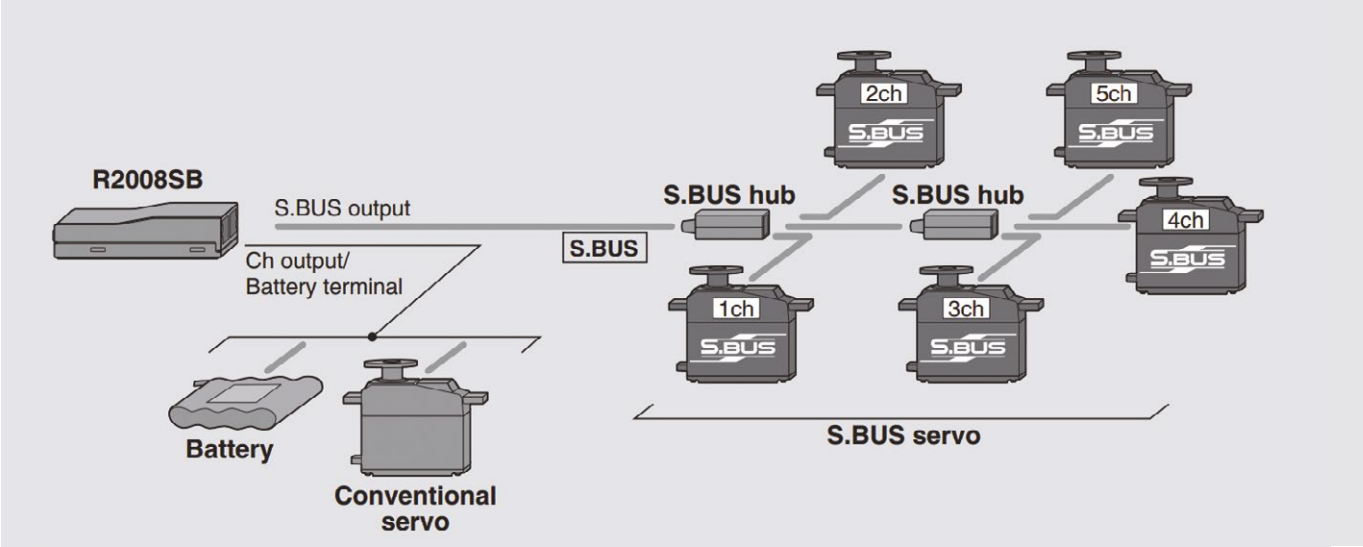
NIVEAU	TEMPS	BUDGET
	15 minutes	environ 400 € (Arduino inclus)

Le récepteur RS2008SB de Futaba utilisé pour nos expérimentations. C'est un modèle relativement classique et courant, compatible S-FHSS/FHSS 2.4 Ghz et permettant de contrôler jusqu'à 8 servomoteurs classiques ou S.BUS. C'est le modèle livré en standard avec la radiocommande Futaba T8J.

Il existe bien des modèles de télécommandes et de récepteurs, cet article ne portera cependant que sur une marque et une famille de modèles bien spécifique. Futaba est un fabricant parmi les plus populaires, les autres étant Graupner, Walkera ou encore FrSky (une valeur qui monte et qui a choisi la voie du logiciel libre pour sa radiocommande Taranis X9D). Ce qui nous intéresse ici est une fonctionnalité bien précise des récepteurs Futaba. Traditionnellement, le récepteur, d'un modèle compatible avec l'émetteur, contrôle un ensemble de servomoteurs directement. Il y a donc autant de connecteurs sur le récepteur que de servos à commander. On se retrouve donc avec une architecture en étoile, exactement comme pour les réseaux Ethernet actuels.

Pour rappel, un servomoteur est un moteur asservi qui prend et maintient une position en fonction de la durée d'une impulsion sur sa broche de contrôle. Il ne s'agit pas à proprement parler de PWM (*Pulse Width Modulation* ou modulation de largeur d'impulsions) où c'est le rapport cyclique qui est important. On parle ici de PPM pour *Pulse Position Modulation,* car ce n'est que la durée où le signal est à l'état haut qui compte et non le rapport entre le haut et bas.

Pour simplifier le câblage, Futaba a eu, comme d'autres, l'idée de travailler sur une architecture sensiblement différente en mettant en œuvre un bus. Ceci s'avère extrêmement pratique pour les multirotors. En effet, qu'il



Exemple de connexion d'un récepteur RS2008B de Futaba. À droite, 5 servos connectés en S.BUS et à gauche en bas un servo standard (PPM) et une alimentation (source : documentation Futaba).

La radiocommande Futaba T8J. Un classique du genre parmi les modèles d'entrée de gamme. Si vous avez, comme moi, eu des jouets radiocommandés en étant enfant, ceci est un saut technologique considérable. Plein de boutons, un écran LCD, tout un système de configuration embarqué, une antenne interne... Terminé les interférences avec « papa castor et son tonton qui demande s'il y a des cowboys sur le grand ruban » !

s'agisse de solutions propriétaires comme les contrôleurs Naza de DJI ou de solutions open source comme le Pixhawk Autopilot (successeur de l'APM (alias Ardupilot)), il est nécessaire de connecter le récepteur au module de contrôle dès lors qu'on souhaite contrôler son engin à distance (et non en auto-pilote comme un drone, un vrai). Le contrôleur de vol assure la stabilité de l'appareil, la gestion de la télémétrie et des différents capteurs, mais il reçoit des ordres du récepteur. Dans le cas d'un récepteur « classique » utilisant la PPM, ceci suppose donc autant de connexions que de canaux (la vitesse des rotors est contrôlée à la manière de servomoteurs). C'est très pénible et surtout chaque gramme qu'on économise sur le poids de l'appareil se traduit pas des secondes et des minutes de vols supplémentaires.

Futaba propose donc un connecteur utilisant un protocole appelé S.BUS qui est une liaison série. Initialement développé pour une nouvelle génération de servos « intelligents », ce protocole s'avère idéal pour les multirotors, car trois malheureux fils suffisent à établir un dialogue entre le récepteur RF et le contrôleur de vol.

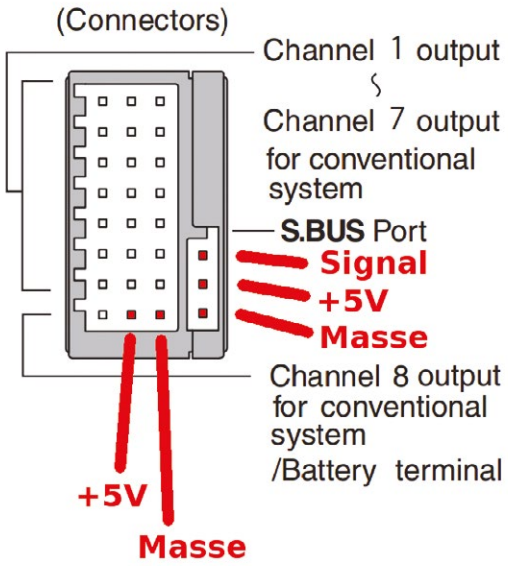
Le modélisme RC (radiocommandé) est un monde à part avec ses valeurs, ses produits vedettes, ses coutumes, etc. Comme pour d'autres activités, rares sont les amateurs du domaine qui « descendent » dans les entrailles du fonctionnement électronique des éléments qu'ils utilisent. Il y a déjà suffisamment à faire, à savoir et à apprendre quant à la mise en œuvre de ces modules, et pour qui s'intéresse au pilotage radiocommandé, d'autres connaissances doivent être acquises avant d'avoir le temps de se pencher sur le fonctionnement électronique. En ce qui me concerne, ce serait plutôt l'inverse : m'intéresser au domaine comme la conséquence d'une certaine obsession pour l'électronique et la bidouille. La question vient alors naturellement : comment utiliser le récepteur Futaba R2008SB pour contrôler n'importe quel montage à base d'Arduino avec ma radiocommande Futaba T8J ?

Remarquez une chose importante ici : une telle radiocommande est un accessoire à part entière, relativement coûteux (~300 €). Dans mon cas, son utilité principale est de contrôler un hexacoptère DJI F550 équipé d'un contrôleur de vol Naza M Lite (en attendant de passer à une solution open source). Il se trouve simplement que la radiocommande T8J était livrée avec deux récepteurs, dont un placé dans le F550. Étant donné le budget, je ne saurai donc vous conseiller un tel achat si le seul usage envisagé est de jouer avec une carte Arduino. La section « *Ce qu'il vous faut* » liste cependant l'ensemble des éléments et affiche donc un coût total prohibitif.

coromonadalix

LE PRINCIPE

Le récepteur R2008SB dispose d'un ensemble de connecteurs permettant le branchement de servomoteurs ou d'un S.BUS. Seul un certain ensemble de broches nous intéresse ici, car notre objectif est de faire lire le S.BUS par la carte Arduino. Le connecteur S.BUS est celui placé horizontalement et nous utiliserons, en plus le connecteur 8 (vertical) afin d'alimenter le récepteur directement par la carte Arduino.



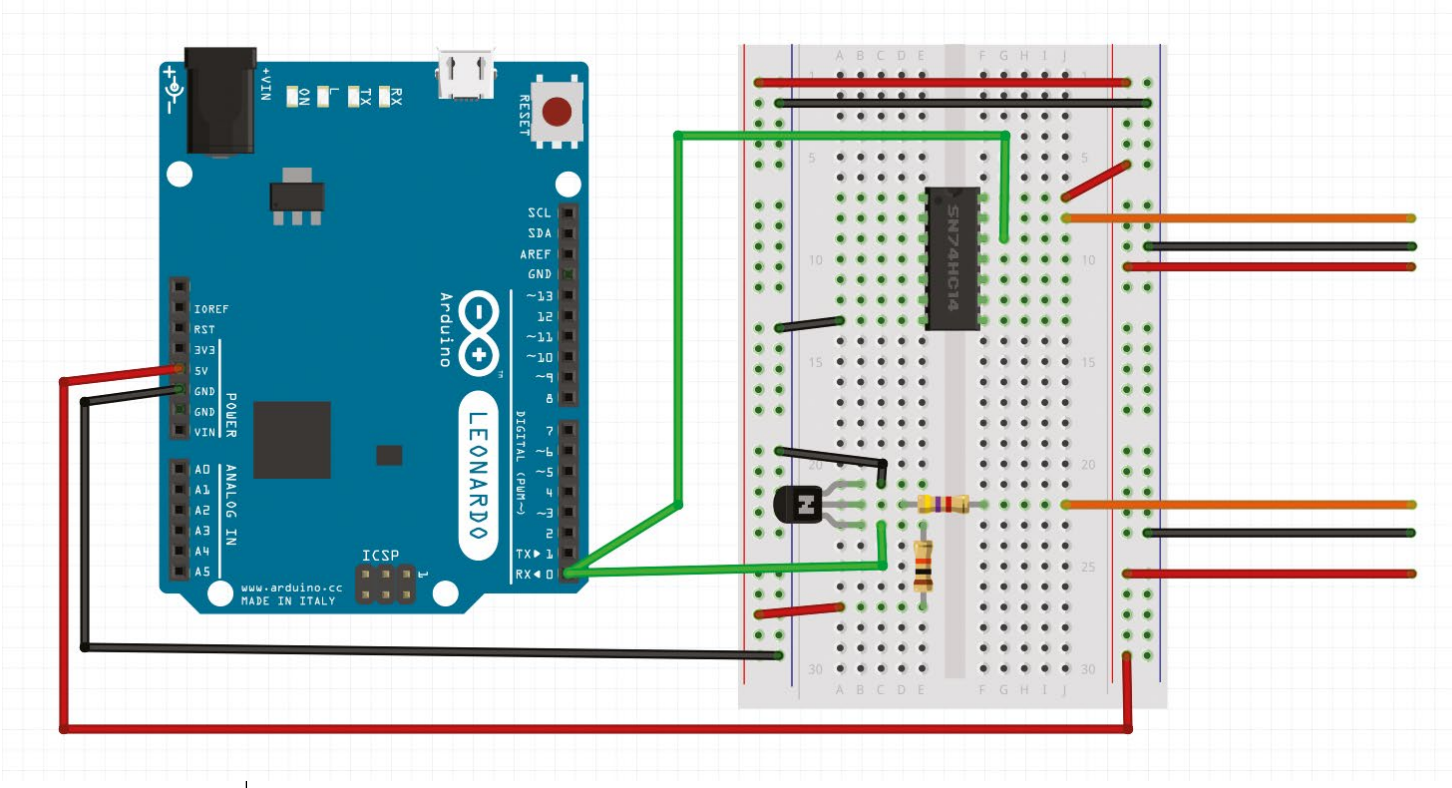
Le protocole S.BUS prend la forme d'une communication série un peu particulière et les trois broches formant le connecteur S.BUS se résument à une masse, une tension d'alimentation et un signal TX destiné normalement aux servos compatibles S.BUS. L'une des particularités du port S.BUS est d'utiliser des signaux inversés, il nous faut donc un composant supplémentaire. Ceci peut être un inverseur construit avec un transistor type BC547B et une paire de résistances, mais nous opterons ici par défaut pour la solution de facilité : un circuit intégré 74HC14 contenant 6 inverseurs. Notez que n'importe quel circuit inverseur fera l'affaire, j'ai ici utilisé un 74AC14 restant d'un autre projet (un réflectomètre ou TDR permettant de faire différentes mesures intéressantes). Il s'agit simplement d'une version « rapide » d'un 74HC14 ou 74HCT14 (version compatible TTL).

CE QU'IL VOUS FAUT

- **Une radiocommande Futaba. Peu importe le modèle du moment qu'elle peut utiliser un récepteur disposant d'un port S.BUS. Notez qu'il existe de nombreuses compatibilités et combinaisons compatibles puisque le protocole S.BUS a énormément gagné en popularité ces dernières années. Vous pouvez avoir une radiocommande Futaba avec un récepteur d'une autre marque et inversement. Le modèle utilisé ici est le T8J (8 canaux).**
- **Un récepteur compatible avec la radiocommande et disposant d'un port S.BUS. Le modèle utilisé dans le tutoriel est un R2008SB de chez Futaba. La même remarque s'applique que pour l'élément précédent en termes de compatibilité. Assurez-vous simplement de disposer d'un port S.BUS et vérifiez si le signal est effectivement inversé ou non (il semblerait que les récepteurs X8R 8/16Ch de chez FrSky n'inversent pas le signal).**
- **Une carte Arduino avec une préférence pour une Mega ou une Leonardo. Le récepteur est connecté par un port série à l'Arduino, mais nous pouvons souhaiter utiliser un autre port pour afficher des informations sur la console. Il est donc de bon ton de choisir une carte avec deux ports série (*Serial* et *Serial1*) pour ne pas avoir à utiliser la bibliothèque *SoftwareSerial*.**
- **Un sextuple inverseur 74HC14, 74HCT14 ou 74AC14. Ceci peut être remplacé par un simple transistor BC547B et deux résistances (10 Kohms et 4,7 Kohms), ou n'importe quel transistor NPN d'usage générique.**
- **Une platine à essais et quelques câbles.**

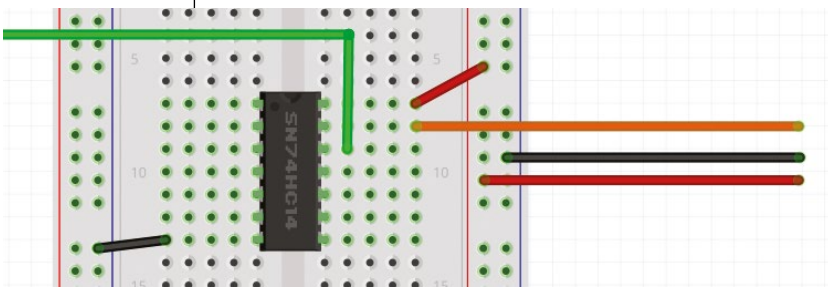
coromonadalix

LE MONTAGE



Le montage complet est relativement basique. Il consiste simplement à utiliser un circuit permettant d'inverser le signal envoyé par le récepteur Futaba. Nous avons ici deux versions du circuit sur une même platine à essais. À vous de choisir l'une ou l'autre selon les composants dont vous disposez.

En haut, nous avons un exemple d'utilisation d'un composant 74HC14 contenant six inverseurs. Comme la plupart des circuits logiques, la broche en haut à droite (14) est Vcc, l'alimentation. En bas à gauche (7) se trouve la connexion à la masse. Le signal parvient via le câble orange sur la broche 13 et ressort sur la broche 12. Le principe de fonctionnement d'un inverseur est simple : si la broche 13 est à la masse, la broche 12 est à Vcc, et inversement. Le 74HC14 ne se contente pas de ce fonctionnement en bascule, il intègre 6 circuits logiques appelés triggers de Schmitt.



Ce mécanisme peut être utile dans plusieurs situations et en particulier pour se débarrasser du bruit et des parasites dans les signaux logiques. Il repose sur des tensions de seuil. Ici, comme il s'agit d'un inverseur, lorsque la

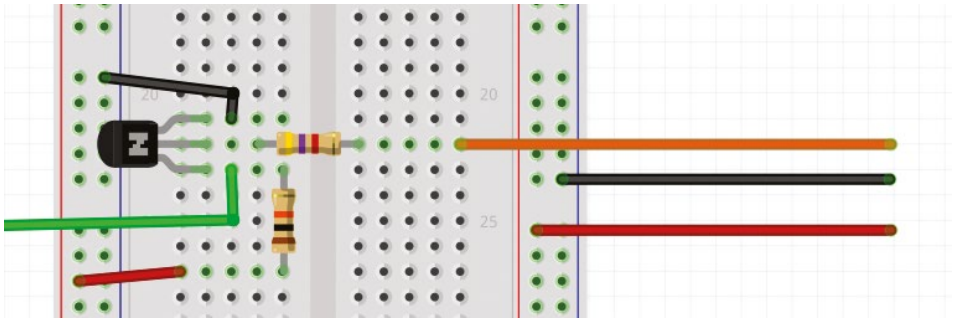
tension entrée est en dessous du seuil haut, la sortie est à l'état logique haut (Vcc). Lorsque la tension dépasse le seuil, la sortie passe à bas. Mais un autre seuil est également utilisé. Si la tension en entrée repasse sous le seuil haut, rien ne change. Il faut en effet que celle-ci passe sous un seuil bas pour que la sortie passe alors au niveau logique haut.

L'avantage de ce mécanisme est d'éviter une transition non franche entre les deux états. Il est faux d'imaginer les tensions d'un circuit logique comme étant parfaitement stables. Le bruit dans le signal au moment de la transition haut/bas par exemple, provoque généralement une succession de changements d'état avant de se stabiliser définitivement. En utilisant un trigger de Schmitt et donc la mise en place de niveaux de tension différents pour l'activation et la désactivation, on met en place une sorte de marge. Si le bruit reste dans la marge, la transition en sortie se fait alors sans « sauts ».

Remarquez que nous n'utilisons ici qu'un seul des 6 inverseurs à trigger de Schmitt présents dans le composant. Il n'est pas rare de voir ce composant utilisé sans inversion, dans l'unique but de se débarrasser du bruit. Il suffit en effet de relier la sortie inversée sur l'entrée d'un autre inverseur pour revenir à la logique de départ, mais avec un signal propre. Les inverseurs à trigger de Schmitt sont également utilisés en compagnie d'un condensateur et d'une résistance pour la connexion de boutons poussoirs par exemple. Ceci formant ce qu'on appelle un circuit RC, l'ensemble permet d'annuler l'effet de « rebond » (sauts d'un état à un autre) provenant de la mécanique même des boutons (lamelles de métal qui se touchent et rebondissent avant de former un contact franc). Un 74HC14 est donc bien plus utile qu'il n'y paraît. À moins de 50 centimes d'euros pièce, vous pouvez donc vous permettre d'en acheter un petit lot, ils seront toujours utiles à un moment ou un autre.

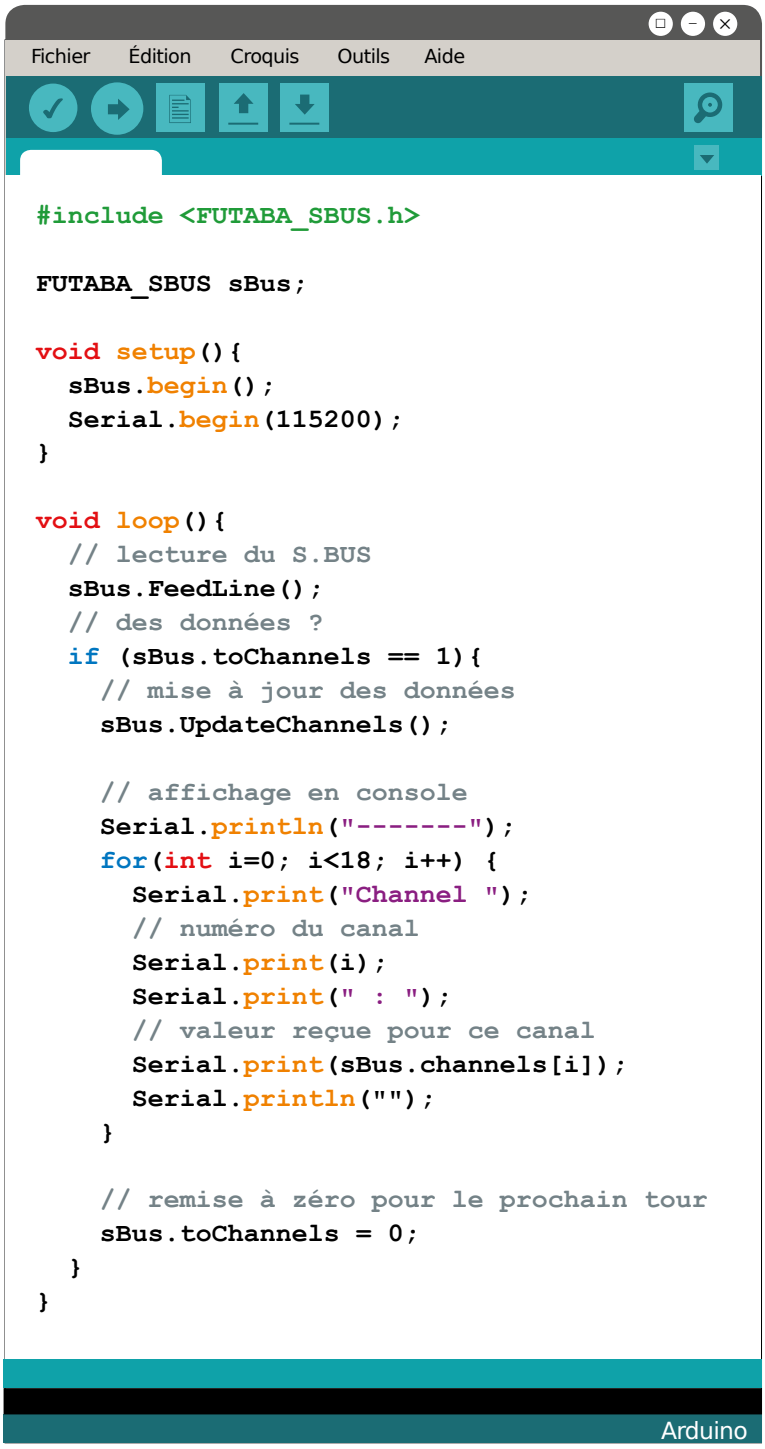
En bas, le schéma présente l'utilisation d'un transistor NPN (BC547 ou 2N2222) pour obtenir un résultat équivalent. Le signal envoyé sur le S.BUS passe par une résistance de 4,7 Kohms reliée à la base du transistor qui donc contrôle le passage du courant entre le collecteur et l'émetteur de ce dernier. Lorsqu'un courant est ainsi appliqué, le transistor est en saturation et le courant passe de Vcc, au travers de la résistance de 10 Kohms, à la masse. La tension entre le collecteur et la masse est donc à 0. En revanche, si la sortie S.BUS est à l'état bas, aucun courant ne passe par la base du transistor, celui-ci n'est donc plus conducteur et le collecteur présente un niveau de tension haut. En reliant le collecteur à l'Arduino, nous avons donc bien un signal qui est inversé par rapport à celui émis par le récepteur Futaba.

Quel que soit le montage utilisé, nous récupérons le signal inversé sur la broche RX1 de l'Arduino Leonardo et pouvons alors le lire et l'analyser. Le reste n'est qu'une affaire de programmation.



coromonadalix

LE CROQUIS




```
#include <FUTABA_SBUS.h>

FUTABA_SBUS sBus;

void setup(){
  sBus.begin();
  Serial.begin(115200);
}

void loop(){
  // lecture du S.BUS
  sBus.FeedLine();
  // des données ?
  if (sBus.toChannels == 1){
    // mise à jour des données
    sBus.UpdateChannels();

    // affichage en console
    Serial.println("-------");
    for(int i=0; i<18; i++) {
      Serial.print("Channel ");
      // numéro du canal
      Serial.print(i);
      Serial.print(" : ");
      // valeur reçue pour ce canal
      Serial.print(sBus.channels[i]);
      Serial.println("");
    }

    // remise à zéro pour le prochain tour
    sBus.toChannels = 0;
  }
}
```

À PROSPOS DU CROQUIS

Ce croquis repose sur la bibliothèque **FUTABA_SBUS** disponible sur **https://github.com/mikeshub/FUTABA_SBUS**. Il vous suffira de pointer votre navigateur sur l'adresse et de cliquer sur « *Download ZIP* » pour récupérer le nécessaire. Le fichier **FUTABA_SBUS-master.zip** contient un répertoire **FUTABA_SBUS-master/FUTABA_SBUS** qu'il vous faudra copier dans le répertoire **libraries** de votre carnet de croquis.

Un exemple, **sbus_example**, est livré avec **FUTABA_SBUS**, mais il utilise une bibliothèque supplémentaire (streaming : **http://arduiniana.org/libraries/streaming/**) permettant d'utiliser l'opérateur de concaténation **<<** pour faciliter l'envoi de données sur le moniteur série. C'est une fonctionnalité intéressante pour les programmeurs coutumiers des langages Java/VB/C#/C++, mais ceci implique d'installer une autre bibliothèque, et ce pour une seule ligne de l'exemple. Personnellement, je trouve que ce genre de dépendance dans un exemple est une mauvaise idée, car bien des utilisateurs, comme en témoignent les messages sur les forums, obtiennent une erreur parce que ladite bibliothèque n'est simplement pas installée (alors que le but d'un exemple est de permettre une utilisation/démonstration rapide). Notre croquis utilise donc la bonne vieille méthode sur **Serial.print** à répétition. C'est moins joli d'un point de vue de la concision du code, mais c'est bien plus explicite.

Notre connexion S.BUS est représentée par l'objet **sBus** qu'on initialise avant toutes choses dans **setup()**

avec la méthode **begin** comme c'est le cas pour bien d'autres bibliothèques (LCD, SPI, etc.). Par défaut, la bibliothèque est faite pour utiliser le second port série de la carte Arduino (**Serial1**) ce qui explique le fait d'utiliser de préférence une Leonardo ou une Mega pour ce type de mise en œuvre. Si vous souhaitez changer cela, il faudra directement modifier la bibliothèque et en particulier le fichier **FUTABA_SBUS.h** où une ligne définie le port utilisé :

```
#define port Serial1
```

Remarquez que juste à la ligne du dessus est définie la vitesse de ce port :

```
#define BAUDRATE 100000
```

C'est là l'autre spécificité du S.BUS. Non content d'avoir des signaux inversés, ce protocole impose une vitesse non standard pour un port série (souvent 115200 normalement). La raison pour laquelle Futaba a fait ces choix étonnants est un mystère, car en réalité nous avons affaire à un bus asynchrone très similaire aux ports séries des cartes Arduino, d'une majorité de microcontrôleurs et de systèmes embarqués ou nano-ordinateurs comme la Raspberry Pi. Dommage, Futaba aurait eu ici l'occasion de faire de son S.BUS un véritable standard ou de simplement reposer sur un standard existant.

Quoi qu'il en soit, une fois l'objet initialisé, le reste de notre travail se passe dans la boucle **loop()** où nous collectons les données avec **FeedLine()** puis testons leur disponibilité en vérifiant la valeur de l'attribut **toChannels**. Si celle-ci est à **1**, nous avons des données à lire et utilisons la méthode **UpdateChannels()** pour les obtenir.

Dès lors, l'ensemble des valeurs de chaque canal est disponible via **sBus.channels[0]** à **sBus.channels[17]** sous forme d'entiers sur 16 bits (**int16_t**). Le récepteur R2008SB ne gère que 8 canaux et nous pourrions donc nous limiter à la lecture de **sBus.channels[0]** à **sBus.channels[7]**, mais autant rendre le croquis plus générique.

Pour traiter les données, nous utilisons une simple boucle **for** affichant sur le moniteur série (via **Serial** donc) le numéro de chaque canal et la valeur lue associée. Pour finaliser le traitement des données, accuser réception de ces dernières et pouvoir en obtenir de nouvelles, nous finissons par repasser la valeur de l'attribut **toChannels** à **0**. Nous sommes prêts pour un nouveau tour dans la boucle et un nouveau test.

Il ne reste plus qu'à essayer l'ensemble en allumant tout d'abord l'émetteur puis à connecter l'Arduino et *uploader* notre croquis compilé (c'est une règle de base : toujours allumer la radiocommande en premier). En activant le moniteur série, nous devons alors voir défiler, à répétition, la liste des canaux et des valeurs, démontrant le bon fonctionnement de l'ensemble. Il suffit alors de manipuler les commandes de l'émetteur pour voir les valeurs changer en direct.

La suite et donc l'évolution du croquis de base présenté ici ne dépendront que de vous. Une fois en mesure de lire les valeurs pour chaque canal, il devient possible d'imaginer n'importe quelle mise en œuvre. Il faudra cependant, avant toutes choses, procéder à une calibration, exactement comme pour la configuration d'un contrôleur de vol pour un multirotor. L'opération consiste à tout d'abord lire la valeur dans la position par défaut puis à réitérer l'opération pour la valeur maximum et minimum suite aux actions sur la radiocommande. On obtient alors une échelle de valeurs qu'il suffira de convertir en pourcentages par exemple, en valeurs sur 8 bits signées (entre -128 et 127) ou non signées (de 0 à 255), pour nous en servir comme bon nous chante. On peut également envisager de stocker les valeurs de calibration en EEPROM nous évitant ainsi cette étape à chaque mise sous tension du montage.

Dès lors il deviendra possible de contrôler l'intensité lumineuse de leds, la vitesse de moteurs, un tracé sur sur un écran LCD... ou tout simplement la position de servomoteurs. Il devient également possible d'envisager une utilisation plus générique en créant la base d'un véhicule radiocommandé comme un *rover* ou un aéronef. Bien entendu, dans ce cas, d'autres composants et modules seront nécessaires (contrôle des moteurs, gestion de l'alimentation, pilotage des capteurs). Selon le cahier des charges prévu, il faudra alors prévoir une carte Arduino plus conséquente ou, tout simplement, une carte dédiée compatible Arduino comme celles supportées par APM Autopilot (ex-Ardupilote) intégrant de base les fonctionnalités attendues (gyroscope, accéléromètre, baromètre, interfaces, mémoire flash, etc.).

coromonadalix

OH LE PETIT CURIEUX !

Difficile de résister à la tentation d'en savoir un peu plus sur ce récepteur. Il est inutile d'espérer trouver quelque chose de significatif en ligne. En effet, étant donné la difficulté pour simplement obtenir une description précise et détaillée du protocole S.BUS, il est plus qu'improbable de trouver des informations concernant le fonctionnement et la conception même du récepteur (quelqu'un chez Futaba doit se dire qu'il est bien plus rentable de vendre de nouveaux ensembles radiocommande/récepteur que de permettre une mise à jour de firmwares). Mieux vaut alors attaquer le sujet « à la dure » en utilisant une bonne vieille méthode très simple : le démontage.

Hmmmm... exactement le même effet qu'un gros bouton rouge « ne pas appuyer » ! Notez au passage que le choix des mots n'est pas très rassurant pour un récepteur trouvant généralement place dans un appareil censé voler...

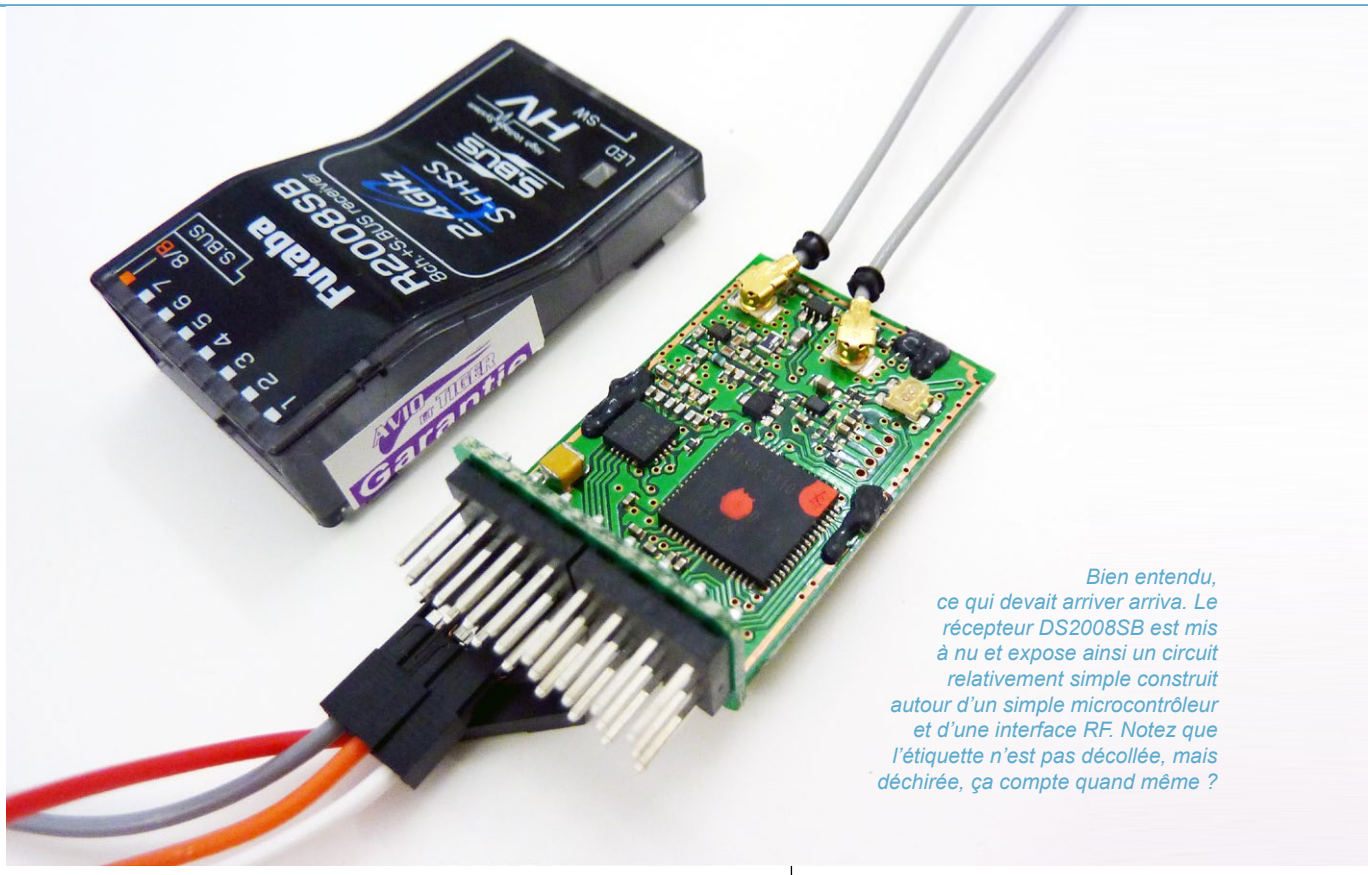
Précisons bien entendu que même si l'ouverture du récepteur est un jeu d'enfant, celle-ci est « sécurisée » par une classique étiquette de garantie qui, une fois endommagée, vous prive de toutes possibilités de retour/échange en cas de problème. À quelques 50 euros le récepteur R2008SB, c'est là quelque chose qui mérite réflexion... sauf si vous êtes moi, auquel cas l'instant même du déchirage de l'étiquette est un moment de bonheur et la promesse d'une aventure. De plus comme j'ai cédé à la tentation, vous n'avez pas à le faire pour satisfaire votre curiosité.

Après ouverture, on constate en premier lieu que l'aspect mécanique est à la hauteur de la qualité générale de l'ensemble. Les antennes par exemple ne sont pas simplement soudées au circuit, mais disposent d'un système de connexion similaire à ce qu'on peut voir sur les cartes Wifi PCI Express Mini intégrées aux ordinateurs portables. Cela est une bonne nouvelle car, si suite à un crash, une antenne est endommagée, il est relativement simple de la changer. Il doit également être possible, avec quelques compétences dans le domaine, d'étendre la portée de réception en jouant sur cet élément.

Mais la surprise vient surtout du composant principal autour duquel est construit le récepteur. Surprise qui s'avère finalement relativement logique en y réfléchissant un instant. L'élément central n'est autre qu'un microcontrôleur Texas Instrument MSP430F5310. C'est exactement la même famille composants basse consommation qui équipe les TI Launchpad MSP (sauf le récent MSP-EXP432P401R équipé d'un MSP432 ARM Cortex M4F 32 bits). On notera d'ailleurs la présence sur le circuit d'un connecteur de 4 contacts qui semble être une interface de programmation JTAG/Spy-Bi-Wire typique des microcontrôleurs MSP430.

La présence d'un MSP430 n'est pas une vraie surprise, car c'est une solution bien moins coûteuse que le développement d'un composant dédié ou d'un ASIC. Ce genre d'approche technique est maintenant très courante, quel que soit le type de matériel « intelligent » qui est produit. Il est, par exemple, plus que probable que votre réveil ou votre autoradio intègre un microcontrôleur d'un modèle bien connu. Le fait que la famille MSP430 soit spécialisée dans la basse consommation est l'autre « non-surprise ». En effet, la diversité des modes de fonctionnement couplée à la plage de tensions utilisables fait de ces microcontrôleurs des spécialistes du genre (même si Freescale s'évertue à démontrer le contraire). Le MSP430F5310 par exemple, à plein régime (mode actif, ou AM) affiche en 3V et en exécution d'un programme en flash, un courant consommé de moins de 5 mA. Courant qui chute littéralement à 52 µA en mode LPM3 (veille de premier niveau).

coromonadalix

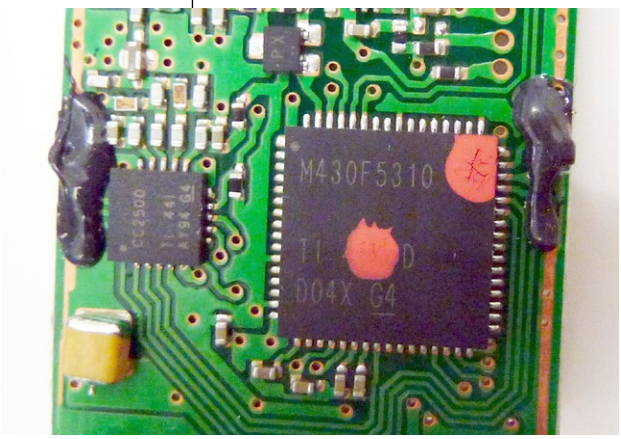


Bien entendu, ce qui devait arriver arriva. Le récepteur DS2008SB est mis à nu et expose ainsi un circuit relativement simple construit autour d'un simple microcontrôleur et d'une interface RF. Notez que l'étiquette n'est pas décollée, mais déchirée, ça compte quand même ?

L'autre composant intéressant du R2008SB est celui placé juste à côté du MSP430. C'est également une puce Texas Instrument, mais spécialisée dans la communication radio : un CC2500. Il s'agit d'un émetteur/récepteur (ou *transceiver* en anglais) qualifié de *low-cost low-power* (faible coût, faible consommation) par TI. Le composant intègre tous les éléments permettant le traitement des données, la gestion des paquets, la modulation/démodulation et la conversion analogique/numérique. Il est en mesure de travailler avec des fréquences entre 2400 et 2483,5 MHz en différentes modulations (OOK, 2-FSK, GFSK, et MSK). Le tout en un petit composant de quelques millimètres de côté auquel s'ajoute simplement un circuit d'amplification RF.

Pour résumer, ce que nous apprend cette rapide exploration bonus n'est rien de moins que : en principe, avec les efforts adéquats et du temps, il doit être parfaitement possible, à l'aide d'un shield ou un module CC2500 (très courants), à la fois de prendre la place du récepteur, mais également de la radiocommande. Et pour les plus curieux d'entre vous, non, il n'est pas possible d'utiliser un récepteur DVB-T RTL-SDR dans cette gamme de fréquences. Mais un *HackRF One* le pourra (10 MHz à 6 GHz), à condition d'arriver à gérer la méthode de transmission S-FHSS/FHSS, puis de décoder le protocole de Futaba, ce qui est une autre paire de manches... **DB**

Après inspection sommaire, il s'avère que le récepteur est construit autour d'un microcontrôleur basse consommation MSP430F5310 de Texas Instruments disposant de 32 Ko de flash intégrée et d'un ensemble de périphériques parfaitement adaptés à la tâche. À gauche sur la photo, un autre composant TI, le CC2500 chargé de la communication radio en 2,4 Ghz et connecté en SPI au MSP430 (on voit d'ailleurs clairement 3 pistes sortant au bas du composant et rejoignant le MSP430 par la droite).



coromonadalix

ARDUINO 1.6 : UTILISER L'EEPROM INTERNE N'A JAMAIS ÉTÉ AUSSI SIMPLE !

Denis Bodor

Nous avons vu dans le précédent numéro qu'il était possible d'utiliser la mémoire flash comme espace de stockage pour les données/ variables qui ne changent pas et ainsi économiser la précieuse mémoire vive. Seul problème, il n'est pas possible d'écrire en flash depuis un croquis, il s'agit simplement de stockage statique. Mais les cartes Arduino proposent une autre solution pour conserver des données entre deux mises sous tension : l'EEPROM intégrée !

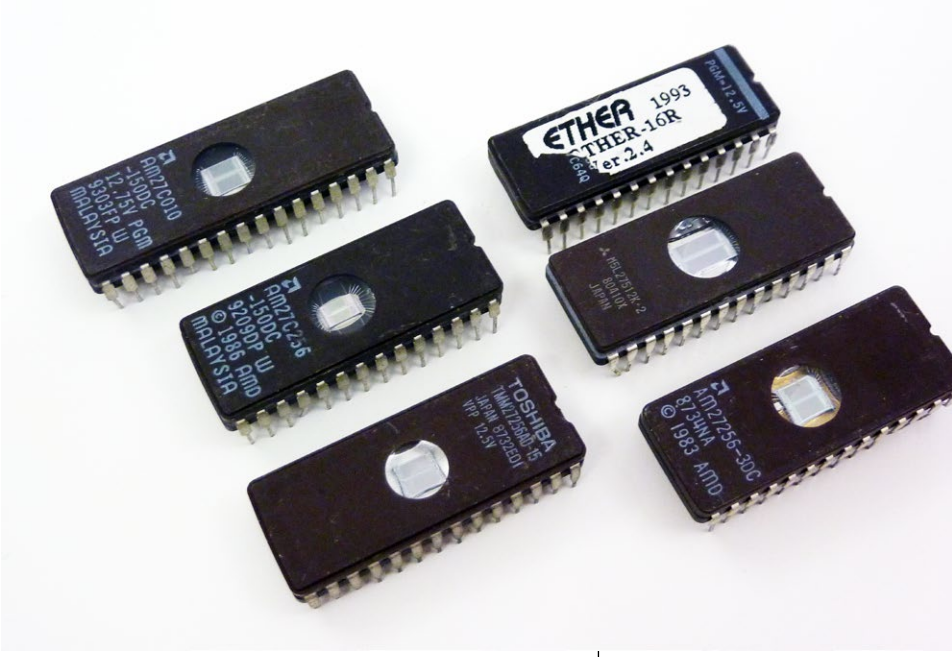
coromonadalix

Dans un précédent article, nous avons vu qu'il était possible de placer des informations dans la mémoire normalement utilisée pour le programme. L'utilisation de **PROGMEM** nous a permis d'économiser énormément de mémoire, mais ceci n'est utilisable qu'au moment de l'écriture du croquis, seule la lecture peut être faite au moment de l'exécution. Les données ainsi stockées ne peuvent donc pas être changées au cours de la vie du programme.

Ceci est relativement problématique, car on se retrouve dans une situation où le seul intérêt est finalement d'économiser de la mémoire vive (SRAM). Mais la solution est là, et ce depuis très longtemps. Le microcontrôleur formant le cœur de la grande majorité des Arduino, l'Atmel AVR, existait bien avant Arduino et disposait déjà de cette solution qui tient en un mot : EEPROM.

1. EEPROM ?

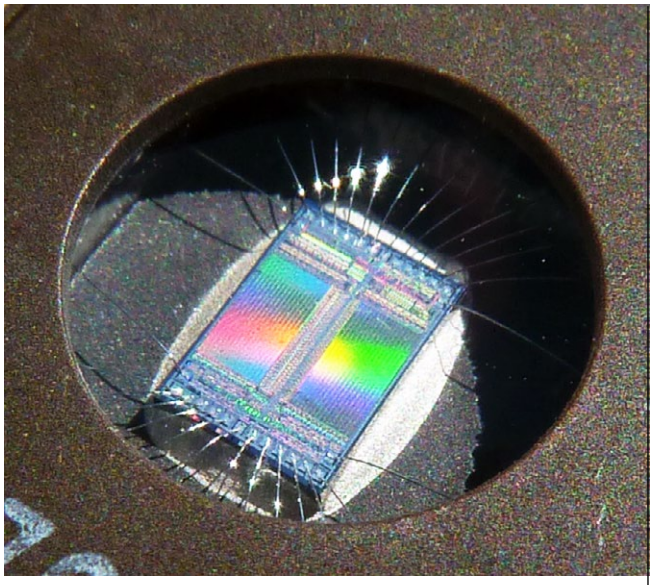
EEPROM est l'acronyme de *Electrically Erasable Programmable Read-Only Memory* qui est une désignation pleine d'ambiguïté et de contradiction. En effet, ceci peut se traduire en « mémoire en lecture seule programmable et effaçable électriquement ». Voilà qui peut paraître surprenant puisque le concept de lecture seule est relativement opposé à celui d'effacement ou de programmation. En réalité, cette dénomination découle directement d'un enchaînement



Une petite sélection d'UV-EPROM d'environ 30 ans d'âge. Remarquez, en haut à droite, l'une d'elles possède encore son étiquette de protection qui, par la même occasion, témoigne de son usage. Ce type d'EPROM était souvent utilisé pour permettre aux cartes réseaux de démarrer sans disque en tant que simple terminal Netware IPX/SPX (Ethernet 10base2 coax). Ceci existe toujours, mais avec des EEPROM intégrés et en PXE.

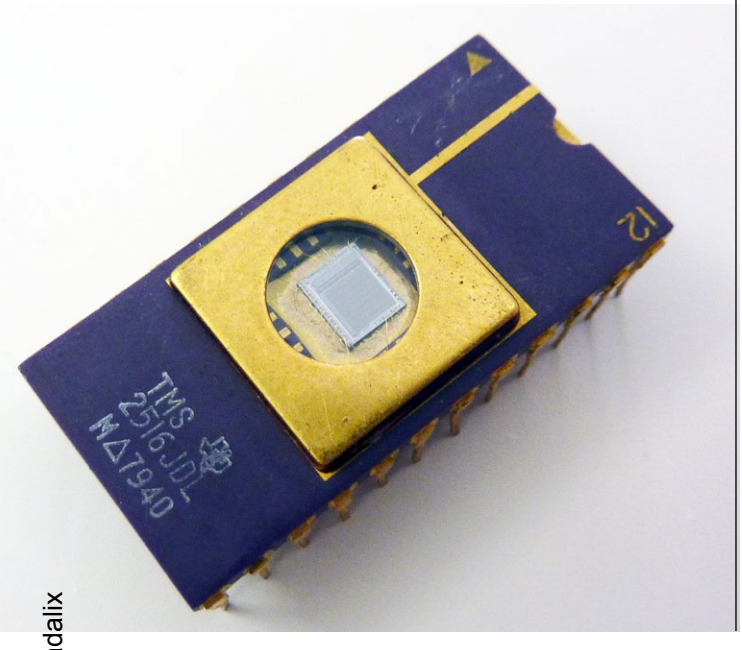
d'évolutions concernant ce type de composants. Avant l'EEPROM était l'EPROM (ou UV-EPROM), une mémoire programmable une fois, mais effaçable intégralement par une exposition à un rayonnement ultraviolet. La technique consistait à exposer la puce aux UV pour l'effacer totalement, pour ensuite mettre en œuvre un programmeur utilisant une tension importante (de l'ordre de 12V) pour inscrire les données. Le composant était ensuite protégé par une étiquette pour empêcher tout effacement accidentel par exposition à la lumière du soleil par exemple et pouvait être lu en utilisant une tension plus « normale » de 5V.

L'effacement par UV était un progrès important, car avant cela n'existaient que les PROM. Ces composants étaient à usage unique ou plus exactement à programmation unique (OTP pour *One-Time Programmable*, programmable une seule fois). Les PROM reposent sur une technologie utilisant un procédé irréversible. La PROM était livrée vierge et ne contenait que des bits à **1**. Pour inscrire les données, on utilisait les bits comme des fusibles en « brûlant » les **1** en **0**. Il était donc toujours possible d'écrire dans le composant, mais uniquement sur les bits encore à **1**, impossible de revenir en arrière. Les PROM étaient souvent utilisées dans les bornes d'arcades ou parfois dans les cartouches de jeu des premières consoles. Ce qui nous amène au dernier type de mémoire,



La caractéristique la plus fascinante même aujourd'hui est le fait de pouvoir littéralement voir le circuit intégré dans le composant ainsi que les minuscules connexions de la puce aux pattes du boîtier. Cette EPROM est une AM27C256 32 Ko de chez AMD (1986), les deux zones qu'on distingue clairement sont les matrices de stockage.

La plus vieille pièce de ma collection, une EPROM TMS-2516 de fin 1979 de chez Texas Instruments en boîtier céramique et pattes plaquées or. Cette petite chose permet de stocker royalement 16384 bits. Oui, ça ne fait que 2 Ko, mais à cette époque les supports de stockage les plus courants étaient des disquettes de 5"1/4 de 360 Ko et les disques durs de 10 Mo étaient un véritable luxe. Autre temps, autre échelle...



coromonadalix

les ROM. Celles-ci ne sont pas programmables et fabriquées en usine directement avec les données à stocker. Généralement, les ROM étaient utilisées pour les grosses productions et les PROM pour des quantités moindres.

Au fil du temps et des progrès de la technologie, il semblerait donc que la souplesse, l'espace de stockage et la désignation aient tendance à s'accroître. Ceci est d'autant plus vrai que la plupart des EEPROM actuelles sont des SEEPROM, des EEPROM série auxquelles ont accède via un protocole comme SPI ou i2c, par opposition aux EEPROM parallèles utilisant un bus de 8 bits ou plus.

Aujourd'hui, ce qu'on appelle EEPROM n'est pas très différent technologiquement d'autres mémoires comme les flash série de bien plus grandes capacités. Ce n'est pas tant le procédé de fabrication ou le système interne de rétention des données qui importe que la manière d'utiliser ces mémoires. Une SEEPROM est généralement de plus faible capacité (quelques Ko), mais permet la modification (effacement et écriture) octet par octet. Une mémoire flash série (SPI) affiche généralement une taille de 256 Ko, 512 Ko ou 1 Mo (ou plus), mais s'utilise bloc par bloc avec des unités (*erase size*) de 4, 32 ou 64 Ko.

De ces quelques rapides explications vous pouvez conclure qu'il existe des dizaines de modèles d'EEPROM répartis entre les différents constructeurs comme Microchip, Atmel ou encore Texas Instruments. Des noms qui évoquent certainement d'autres composants et en particulier des microcontrôleurs. Il n'est donc pas surprenant d'apprendre que la plupart de ces microcontrôleurs intègrent des EEPROM directement dans la puce. Il s'agit de petites zones de stockage accessibles en lecture comme en écriture qui viennent compléter les autres espaces déjà disponibles (flash, registres, SRAM). Les cartes actuelles comme la Uno, la Nano ou la Leonardo utilisent un AVR intégrant une EEPROM de 1 Ko. La Mega 2560 est plus généreuse, grâce à son ATmega2560 et offre 4 Ko d'EEPROM. Les anciens modèles de cartes, reposant sur des ATmega168, ne disposaient que de 512 octets. Précisons que la future Arduino Zero, d'une architecture totalement différente (tout comme la Due ou la Yùn) pourra proposer 16 Ko d'EEPROM, mais il ne s'agira que d'une émulation.

2. AVANT 1.6 : UTILISATION BASIQUE

Dès les premières versions de l'environnement Arduino, les bibliothèques de base permettaient déjà un accès à l'EEPROM intégrée. La situation resta inchangée durant une longue période et des bibliothèques supplémentaires permettaient d'ajouter un peu de souplesse dans l'utilisation assez basique proposée par défaut (EEPROMex par exemple).

Ces fonctionnalités simples sont toujours présentes et encore largement utilisées par de nombreux projets, bibliothèques et croquis de démonstration. Elles peuvent également s'avérer utiles dès lors qu'il s'agit de simplement enregistrer quelques octets. La bibliothèque **EEPROM** est en effet uniquement capable de lire ou d'écrire **byte**. Voici un exemple :

```
#include <EEPROM.h>

byte mesdata[16] =
  {42,15,78,95,
   69,25,17,13,
   74,19,06,18,
   66,93,86,12};

void setup() {
  for(int i=0; i<16; i++) {
    EEPROM.write(i,mesdata[i]);
  }
}

void loop() {
}
```

Nous avons ici un croquis ayant pour but de simplement enregistrer en EEPROM interne une série de 16 valeurs. Nous créons donc une variable **mesdata** étant un tableau de 16 **byte**. Notre fonction **setup()** se contente, à l'aide d'une boucle **for**, de parcourir chaque case du tableau et d'utiliser la valeur avec la méthode **write()**. Celle-ci prend en argument la position (adresse) dans l'EEPROM et la valeur à écrire.

N'importe quelle valeur peut ainsi être enregistrée, à n'importe quel endroit/moment du croquis. Tout ce dont nous avons besoin est de choisir une position dans l'EEPROM et de travailler octet par octet avec des boucles.

La lecture quant à elle n'est pas plus compliquée :

```
#include <EEPROM.h>

byte madonnee;

void setup() {
  Serial.begin(9600);
  delay(2000);

  for(int i=0; i<16; i++) {
    madonnee = EEPROM.read(i);
    Serial.print("[");
    Serial.print(madonnee, DEC);
    Serial.print("]");
  }
  Serial.println();
}

void loop() {
}
```

Le même principe s'applique ici, mis à part que nous nous passons joyeusement de tableau. Il nous suffit de parcourir les octets de l'EEPROM en utilisant une boucle sur les adresses 0 à 15 et la méthode **read()** en fournissant la position à lire. La méthode retourne directement la valeur lue et nous pouvons alors l'utiliser pour l'envoyer sur le moniteur série avec **Serial.print()**.

Pour tester ces croquis, il vous suffit de programmer le premier dans votre Arduino. Après redémarrage automatique, celui-ci va s'exécuter et inscrire les valeurs en EEPROM. L'étape suivante sera de programmer un autre croquis, quelconque ou vide. À ce stade, la mémoire flash du microcontrôleur sera réécrite, mais pas le contenu de l'EEPROM. Le but de cette programmation est purement démonstratif afin de mettre en évidence la conservation des données. Enfin, chargez le second croquis et surveillez le moniteur série. Vous aurez le plaisir d'y voir apparaître les 16 valeurs initialement enregistrées en EEPROM.

Terminons cette partie en précisant qu'il est possible de connaître, dans le croquis, la taille de l'EEPROM disponible avec **EEPROM.length()**. Ceci nous permettrait par exemple de boucler sur tout l'espace de stockage pour lire les données ou les effacer. Notez cependant

coromonadalix

que cette approche « à la barbare du donjon de naheulbeuk » est une mauvaise idée. En effet, nous devons prendre en compte deux éléments importants :

- l'écriture en EEPROM est plus lente que la lecture. Lire l'intégralité de l'EEPROM ou une grosse partie de celle-ci et la copier dans une variable (tableau), pour ensuite y faire des modifications et tout réécrire sera très lent (plus de 3 ms par écriture, je vous laisse faire le calcul) ;
- la durée de vie de l'EEPROM en cycles d'effacement/écriture est d'environ 100000. Écrire à répétition à une adresse ou un ensemble d'adresses va donc user prématurément la zone en question qui finira par ne plus garantir l'intégrité des données.

Pour contourner partiellement ces problèmes, la bibliothèque **EEPROM** permet d'utiliser **update()**. Cette méthode prend en argument les mêmes paramètres que **write()**, mais va procéder à une lecture avant de déclencher une écriture. Si la donnée lue est déjà identique à celle qu'on souhaite enregistrer, l'opération n'aura pas lieu. Ceci permet simplement de mettre à jour la valeur à une adresse et donc d'utiliser un cycle effacement/écriture que lorsque c'est vraiment nécessaire.

3. AUJOURD'HUI : UTILISATION ÉLÉGANTE ET PRATIQUE

La version stable actuellement diffusée de l'environnement de développement (IDE) Arduino est la 1.6.3. Jusqu'à fin 2014, la version stable était la 1.0.x et des versions beta 1.5.x étaient les seules à prendre en charge les cartes Arduino Due basées sur un ARM Cortex-M3 de chez Atmel (SAM3X8E) très différent des microcontrôleurs AVR. Les versions 1.5.x ont introduit un grand nombre de nouveautés aussi bien dans la prise en charge des cartes que dans la chaîne de compilation ou les bibliothèques de base.

La version 1.0.6 est maintenant de l'histoire ancienne, car la stabilisation de la série 1.5.x en 1.6.x est effective depuis début février 2015. En ce qui nous concerne, la nouvelle fonctionnalité intéressante est arrivée avec la 1.6.2 et une timide mention dans les *Release Notes* : « *EEPROM : Replaced existing library with more complete implementation* » (« remplacement de la bibliothèque par une implémentation plus complète »). En réalité, le changement mériterait quelque chose comme « énorme et fantastique avancée dans la gestion de l'EEPROM » !

Pourquoi une telle affirmation de ma part ? Jugez plutôt : deux nouvelles méthodes ont été ajoutées permettant respectivement d'écrire (**put()**) et lire (**get()**) n'importe quel type de données (**int**, **float**, **char**, **struct**, etc.). Vous n'êtes donc plus restreint à l'utilisation de **byte**. Et en bonus cadeau, **put()** utilise non pas **write()** pour faire ce travail, mais **update()**, vous n'avez donc plus à vous soucier outre mesure de l'usure de l'EEPROM (c'est toujours un problème, mais la bibliothèque **EEPROM** fait déjà tout ce qu'elle est possible de faire à ce niveau).

Pour comprendre tout l'intérêt de ces ajouts, le plus simple est encore un bon exemple :

```
#include <EEPROM.h>

struct montype {
  float virgule;
  byte entier;
  char texte[16];
};

void setup() {
  montype mesdata = {
    3.14f,
    42,
    "Coucou monde"
  };

  EEPROM.put(0, mesdata);
}

void loop() {
}
```

Nous utilisons **EEPROM.put(0, mesdata)** pour enregistrer les données de **mesdata** à partir de

coromonadalix

l'adresse **0** en EEPROM. L'élément clé de ce code est le type de donnée que nous utilisons. Il ne s'agit pas d'un octet, mais d'une structure complète. En C/C++, la directive **struct** permet de créer un type de donnée complexe. La syntaxe est la suivante :

```
struct nom {
  type membre1;
  type membre2;
  type membre3;
};
```

Voyez cela comme un moule permettant de créer des variables pour y placer des données de type différents. Nous créons littéralement un nouveau type de données qui se compose d'un nombre de membres quelconques, mais un type précis. Dans notre exemple, nous créons **montype** qui est une structure contenant un **float**, un **byte** et un tableau de 16 **char** avec :

```
struct montype {
  float virgule;
  byte entier;
  char texte[16];
};
```

Notre structure **montype** n'est **PAS** une variable, mais un **type** de données composé. Pour nous en servir, nous devons déclarer une variable utilisant ce type :

```
struct montype mesdata;
```

que nous pouvons ensuite utiliser avec :

```
mesdata.virgule=3.14f;
mesdata.entier=42;
mesdata.texte="Coucou monde";
```

Notez l'utilisation du point (**.**) pour préciser le membre utilisé dans la structure pour l'affectation d'une valeur. Nous pouvons cependant simplifier tout cela. En premier lieu, en C++, il n'est pas nécessaire de préciser **struct** dans la déclaration, **montype mesdata** est suffisant pour déclarer **mesdata** comme structure de type **montype**.

Il n'est pas non plus nécessaire de séparer la déclaration de l'initialisation. Nous pouvons tout faire en une seule fois avec :

```
montype mesdata = {
  3.14f,
  42,
  "Coucou monde"
};
```

ou encore, si vous préférez :

```
montype mesdata = {3.14f, 42,
"Coucou monde"};
```

Ceci évite la multiplication des lignes de code. Il faut simplement préciser les valeurs dans l'ordre des membres dans la structure. Mais ce qui est prodigieusement pratique dans ces nouveautés concernant l'EEPROM est le simple fait que, quelle que soit la complexité de la structure, une seule et unique ligne nous permet de sauvegarder tout cela : un simple **EEPROM.put(adresse, variable)**.

En ce qui concerne la lecture des données depuis l'EEPROM, le croquis est assez similaire :

```
#include <EEPROM.h>

struct montype {
  float virgule;
  byte entier;
  char texte[16];
};

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

  montype mesdata;

  EEPROM.get(0, mesdata);

  Serial.println("EEPROM lue !");
  Serial.println(mesdata.virgule);
  Serial.println(mesdata.entier);
  Serial.println(mesdata.texte);
}

void loop() {
}
```

On retrouve naturellement la même structure mais, bien entendu, la déclaration de la variable se passe d'initialisation. C'est la méthode **get()** qui se chargera de lire les données et

coromonadalix

ainsi remplir notre **mesdata**. Nous accédons ensuite simplement à chaque membre de la structure pour en afficher le contenu via le moniteur série.

Notez qu'il est important d'utiliser une structure correspondant aux données effectivement enregistrées. Les noms utilisés n'ont pas d'impact sur le comportement du croquis, mais les types et l'ordre des membres dans la structure doivent être parfaitement identiques. Conserver des noms identiques et cohérents est toutefois une excellente solution pour éviter les problèmes, même si, dans la pratique, il est fort probable que ce soit le même croquis qui lise et écrive dans l'EEPROM.

L'opération qu'applique **put()** et **get()** peut être qualifiée de sérialisation/désérialisation de la structure. Nous avons dans notre exemple une structure composée d'un **float** (32 bits), un **byte** (8 bits) et de 16 **char** (16 x 8 bits). Une fois stockées en EEPROM, ces données occuperont donc 32+8+(16*8) bits, soit 21 octets. Ceci signifie, comme nous avons écrit les données à partir de l'adresse 0 en EEPROM, qu'il est possible d'en écrire d'autres à l'adresse 21 sans risquer d'écraser quoi que ce soit.

Mieux encore, nous n'avons pas besoin de faire des calculs nous-mêmes puisqu'il existe un opérateur dédié qui nous retournera la bonne valeur : **sizeof(montype)**. Nous pouvons donc parfaitement faire ceci :

```
montype mesdata = {
  3.14f,
  42,
  "Coucou monde"
};

montype mesautresdata = {
  19.06f,
  23,
  "oh un texte !"
};

EEPROM.put(0, mesdata);
EEPROM.put(sizeof(montype), mesautresdata);
```

sizeof() pourra même être utilisé dans une boucle pour enregistrer une série de données de type **montype** avec **sizeof(montype)*position** où **position** est le numéro de l'enregistrement.

POUR CONCLURE

Ici, avec notre carte Uno, nous pourrions stocker en EEPROM quelques 48 occurrences différentes de la structure de l'exemple. En faisant un peu travailler notre imagination, on choisira de réserver le premier octet de l'EEPROM (adresse 0) pour enregistrer le nombre d'occurrences. Ajouter un enregistrement pourra donc se faire à n'importe quel moment en utilisant l'adresse **sizeof(montype)*EEPROM.read(0)+1** et en mettant à jour ensuite la valeur à l'adresse 0. D'autres techniques peuvent être utilisées comme le fait de tenir à jour une table des emplacements en début d'EEPROM (comme avec un système de fichiers FAT) ou encore de prévoir dans la structure une référence à la prochaine adresse libre (sorte de liste chaînée) ou un marqueur spécifiant l'état (utilisable ou non).

1024 malheureux octets d'EEPROM ne permettent pas cependant beaucoup d'originalité en termes d'algorithme d'allocation. Le plus souvent l'EEPROM sera utilisée pour stocker simplement des valeurs de configuration comme l'adresse MAC d'un shield Ethernet ou des valeurs de calibration d'un capteur ou d'un récepteur de radiocommande (voir article sur le sujet dans le présent numéro).

Pour une masse de données plus conséquente comme l'enregistrement d'une série de mesures (température, pression, position, etc.), il faudra se tourner vers une EEPROM externe, une flash SPI ou encore une carte SD.

Quoi qu'il en soit, cette évolution de la gestion d'EEPROM dans les nouvelles versions de l'IDE Arduino est un véritable bonheur, car il n'est tout simplement plus nécessaire de se torturer l'esprit sur la façon de découper nos variables avant de les enregistrer. **DB**

coromonadalix

PROFESSIONNELS !

DÉCOUVREZ NOS NOUVELLES OFFRES D'ABONNEMENTS ...

PDF COLLECTIFS

		PROFESSIONNELS					
		1 - 5 lecteurs		6 - 10 lecteurs		11 - 25 lecteurs	
OFFRE	ABONNEMENT	Réf	Tarif TTC	Réf	Tarif TTC	Réf	Tarif TTC
PROHK2	6ⁿ° HK	☐ PRO HK2/5	156,-	☐ PRO HK2/10	312,-	☐ PRO HK2/25	624,-

Prix TTC en Euros / France Métropolitaine

PROFESSIONNELS : N'HÉSITEZ PAS À NOUS CONTACTER POUR UN DEVIS PERSONNALISÉ PAR E-MAIL : abopro@ed-diamond.com OU PAR TÉLÉPHONE : 03 67 10 00 20

ACCÈS COLLECTIFS BASE DOCU

		PROFESSIONNELS					
		1 - 5 connexion(s)		6 - 10 connexions		11 - 25 connexions	
OFFRE	ABONNEMENT	Réf	Tarif TTC	Réf	Tarif TTC	Réf	Tarif TTC
PROOS+3	OS	☐ PRO OS+3/5	90,-	☐ PRO OS+3/10	180,-	☐ PRO OS+3/25	360,-
PROH+3	GLMF + HS + LP + HS + MISC + HS + OS	☐ PRO H+3/5	447,-	☐ PRO H+3/10	894,-	☐ PRO H+3/25	1788,-

Prix TTC en Euros / France Métropolitaine

OS = Open Silicium
LP = Linux Pratique
HS = Hors-Série
LM = GNU/Linux Magazine France

coromonadalix

...EN VOUS CONNECTANT À L'ESPACE DÉDIÉ AUX PROFESSIONNELS SUR :

www.ed-diamond.com

DES FEUX DE CIRCULATION AVEC ALARME

Ludovic Grossard

Les feux de circulation, c'est un peu le « Hello World » de l'Arduino (enfin juste après la diode clignotante). Mais allons plus loin, et programmons des feux de circulation directement dans un microcontrôleur à l'aide des bibliothèques C AVR ! Et puis, ajoutons-y également une alarme, et regardons comment on peut jouer avec les interruptions...

coromonadalix

L'objectif est de faire des feux de circulation pour les LEGOs de mes enfants. Le principe est relativement simple. Il suffit d'allumer trois leds, verte, jaune et rouge, en séquence. On pourrait utiliser directement un Arduino pour cela. Pour que ce soit plus compact, moins cher, et plus fun, on va utiliser un microcontrôleur. J'ai choisi un ATTINY85 de chez Atmel. Il dispose de suffisamment d'entrées/sorties pour allumer les différentes leds, et on peut le programmer facilement avec un Arduino.

Pour que ce soit plus sympa (et parce que les pompiers LEGO en ont besoin), nous ajouterons également une alarme. Lorsqu'on appuie sur un bouton poussoir, la led rouge se met à clignoter, et une alarme sonore retentit. Un petit buzzer piezo fera l'affaire ici.

Côté alimentation, on utilisera une pile de 9V. Le microcontrôleur s'alimentant en 5V, un régulateur de tension du type AP78L05 est nécessaire. Il permet d'obtenir en sortie une tension de +5V pour une tension d'entrée allant de +7V à +20V.

Enfin, pour ne pas griller les leds, une résistance de 330 ohms limitera le courant les traversant. Une seule résistance est nécessaire ici, car les leds ne sont allumées qu'une à la fois.

1. LE MONTAGE

Sans transition, voici d'abord le schéma du circuit électrique ci-dessous.

Détaillons chacune des broches de l'ATTINY85 :

- Pin 8 : c'est le +Vcc de l'alimentation du microcontrôleur. Il faut y mettre du +5V. Le régulateur de tension IC1 permet de passer de 9V à son entrée (IN) à 5 V à la sortie (OUT). La borne − de la pile est reliée à l'entrée masse du régulateur de tension (GND).
- Pin 4 : à mettre à la masse.
- Pin 5, 6, 7 : nous utiliserons ces broches en sorties numériques (0V ou 5V) pour allumer les trois leds.

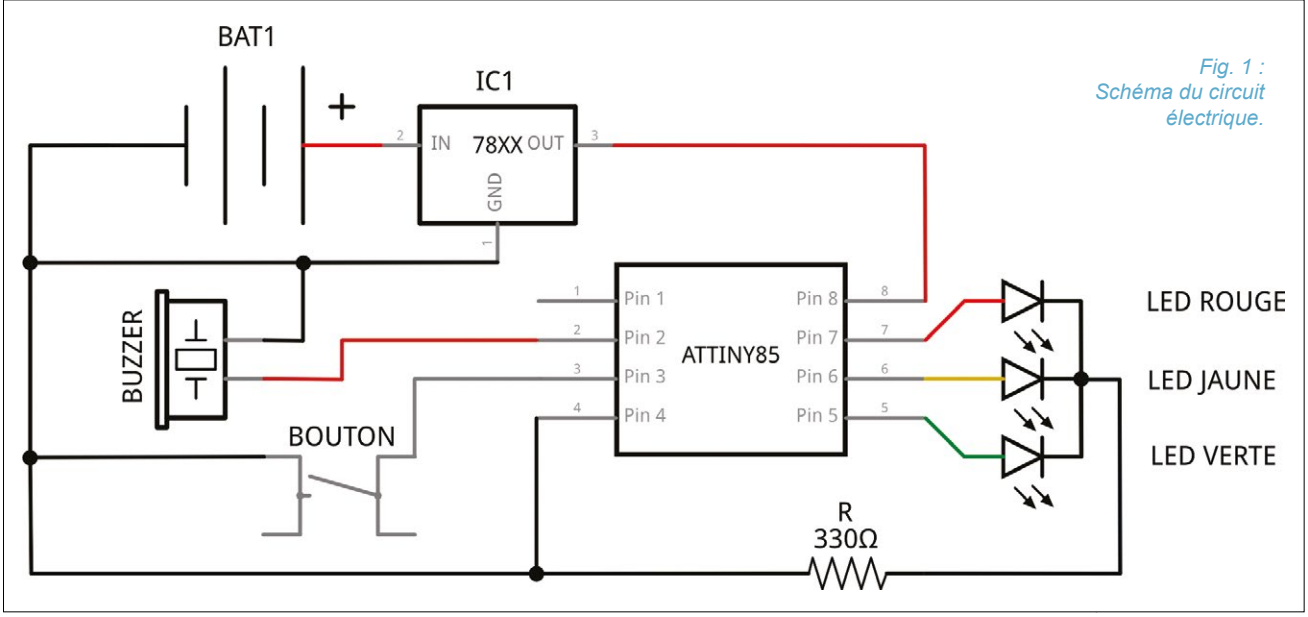


Fig. 1 : Schéma du circuit électrique.

coromonadalix

- Pin 3 : cette broche sera configurée en entrée numérique. Elle sera connectée à un bouton poussoir, qui permettra de la mettre à la masse lorsque l'on appuie dessus. Dans le cas contraire, une résistance de rappel (*Pull Up resistor*) interne à l'entrée 3 permettra de maintenir l'entrée à 5V.
- Pin 2 : utilisée en sortie, cette broche sera reliée au buzzer piézo-électrique pour l'alarme. Il faudra mettre alternativement cette sortie à la masse et à 5V à une fréquence donnée, pour générer un signal acoustique de forme carrée, et de même fréquence.
- Pin 1 : c'est la broche RESET du microcontrôleur. Il ne faut pas l'utiliser si on veut pouvoir reprogrammer le microcontrôleur plus tard.

Dans un premier temps, le montage est réalisé avec une platine d'essai pour pouvoir le modifier très facilement. Voici à quoi ça ressemble :

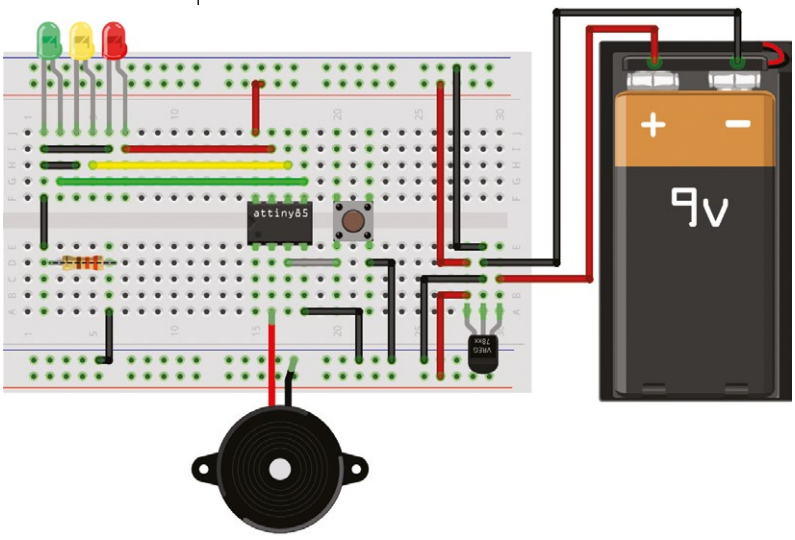


Fig. 2 : Montage sur une platine d'essai pour faciliter les connexions. Le microcontrôleur ATTINY85 est placé au centre de la platine.

2. LE CODE SOURCE

Et maintenant le code du programme, écrit en C. Nous allons ensuite en expliquer le contenu par petits morceaux.

2.1 Instructions préprocesseur

D'abord les instructions préprocesseur :

```
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
```

- la bibliothèque **io** définit les différentes entrées et sorties du microcontrôleur ;
- la bibliothèque **delay** est elle nécessaire pour utiliser des boucles d'attente, qui seront utiles pour gérer l'alarme ;
- enfin, nous utiliserons la bibliothèque **interrupt**, la temporisation des feux et le bouton d'alarme étant gérés par des interruptions.

2.2 Quelques définitions

```
#define DELAY_VERT            10
#define DELAY_JAUNE            2
#define DELAY_ROUGE           10
#define DEMI_PERIODE_ALARME    1
```

Afin de rendre le code plus lisible et plus facile à maintenir, on définit ici les durées pour les feux : 10 s pour le feu vert, 2 s pour le jaune, et 10 s pour le feu rouge. On indique également que la période de l'alarme est de 2 s : pendant une seconde, le feu rouge est allumé et l'alarme retentit. Puis, pendant la seconde suivante, le feu rouge est éteint, et l'alarme est coupée, etc.

2.3 Configuration des ports

Entrons maintenant un peu plus dans le vif du sujet, et attachons-nous à configurer les broches du microcontrôleur. Pour simplifier les choses, les

coromonadalix

différentes entrées/sorties des microcontrôleurs sont regroupées dans des ports. L'ATTINY85 ne dispose que d'un seul port, le PORTB. Sur les huit broches du microcontrôleur, deux sont réservées à l'alimentation (VCC et GND). Il reste donc six broches utilisables, numérotées PB0 à PB5. Cette dernière n'est pas utilisable non plus, car il s'agit de la broche RESET.

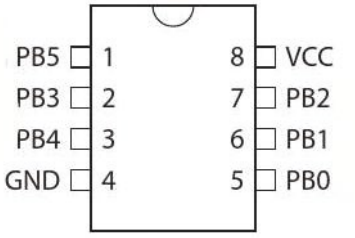


Fig. 3 : Configuration des broches du microcontrôleur. Seules les broches PB0 à PB4 sont utilisables.

Nous avons donc à notre disposition cinq broches, numérotées PB0 à PB4, que l'on peut configurer à loisir en entrée ou en sortie. Attribuons ces broches aux trois leds, au bouton poussoir et au buzzer, avec les définitions suivantes :

```
#define LED_VERTE  PB0
#define LED_JAUNE  PB1
#define LED_ROUGE  PB2
#define BUZZER     PB3
#define BOUTON     PB4
```

Cette façon de procéder permet de rendre le programme indépendant de la configuration matérielle. Dans le code, plutôt que de faire référence au port PB3 (qui correspond à la broche 2), nous écrirons BUZZER, et c'est beaucoup plus parlant.

Nous allons maintenant définir quels sont les ports en entrée et en sortie. Il faut pour cela manipuler le registre de directions du port B, **DDRB**.

Des explications très claires sont disponibles dans ce billet de blog **[1]**. Je vous invite à le lire, car il contient des informations intéressantes pour comprendre la suite.

```
int main(void)
{
// on configure les entrées/sorties 0, 1 et 2
du port B en mode sortie
DDRB = 0;
DDRB |= (1<<LED_VERTE);
DDRB |= (1<<LED_JAUNE);
DDRB |= (1<<LED_ROUGE);

// on configure la 4 en entrée
DDRB |= (0<<BOUTON);
// et on active le PULL UP
PORTB |= (1<<BOUTON);

// et la 3 en sortie pour le buzzer
DDRB |= (1<<BUZZER);

// on allume la led verte et on éteint les
leds jaune et rouge
PORTB |= (1<<LED_VERTE);
PORTB &=~ (1<<LED_JAUNE);
PORTB &=~ (1<<LED_ROUGE);
```

- La première ligne correspond au début du corps de la fonction principale du programme.
- Ensuite, on met tous les bits du registre de direction du port B à 0. Toutes les broches sont alors configurées en entrée. Les lignes 29, 30 et 31 permettent de placer les ports PB0, PB1 et PB2 en sortie en mettant les bits correspondant à 1.
- On met alors PB4 en entrée pour le bouton poussoir. Ça n'est pas obligatoire puisqu'on avait déjà tout mis en entrée ligne 28. Mais bon, c'est pour être bien sûr !
- On met le port 4 au niveau haut (*HIGH*). Ça peut paraître étrange, car le port 4 n'est pas une sortie et donc ça n'a pas de sens de vouloir lui imposer un état. En fait, lorsqu'un port est utilisé en entrée, le bit correspondant dans le registre de données **PORTB** est utilisé pour configurer la résistance de rappel (pull up). Cela signifie ici qu'en l'absence de tout signal sur le port PB4, il sera mis à 5V. Lorsqu'on appuie sur le bouton poussoir, on le met à la masse, et on lira 0V sur PB4.

- On met le port PB3 en sortie pour le buzzer.
- Et on termine enfin en allumant la diode verte, et en éteignant les deux autres.

2.4 Mesurer le temps

Il va falloir gérer le temps, ne serait-ce que pour allumer chaque feu pendant une durée déterminée. Il y a deux façons de faire :

- utiliser la fonction `_delay_ms(T)`. Cette solution a un inconvénient majeur. On demande au processeur d'attendre pendant une durée déterminée T en ms. Pendant ce temps-là, le programme est comme arrêté, car occupé à ne rien faire ! C'est assez embêtant, car si je dois déclencher l'alarme pour une urgence, il n'est pas question d'attendre plusieurs dizaines de secondes que le feu change de couleur. Il faut que ça réagisse tout de suite.
- l'autre solution permet de déclencher l'alarme à n'importe quel moment. Plutôt que de faire tourner en boucle le processeur en attendant que le temps passe, nous allons utiliser le *timer* du microcontrôleur. Il s'agit d'une horloge indépendante du microprocesseur.

Nous utiliserons bien évidemment le timer. Il n'est pas question que le camion de pompiers de mes gamins reste bloqué sur place simplement parce qu'un programmeur (en l'occurrence papa) a mal pensé son programme ! Voyons un peu comment cela fonctionne.

Le timer est simplement un registre 8 bits qui va compter de 0 à 255 au cours du temps, à une certaine vitesse définie entre autres par la fréquence d'horloge du microcontrôleur. Par exemple, si l'horloge du microcontrôleur est de 8 MHz, il faut 255 / 8×10^6 s = 32 µs pour compter de 0 à 255. Ensuite, le timer revient à 0 et se remet à compter. C'est bien beau tout ça, mais à quoi ça sert si le programme n'est pas informé de ce que fait ce compteur ? C'est là qu'interviennent les interruptions. Il est possible de demander au timer de prévenir le processeur chaque fois qu'il arrive à 255, et qu'il repasse à 0. On parle alors de débordement (*OVERFLOW*). Le processeur peut alors réagir à cette interruption, et exécuter du code lorsque cet événement se produit. On appelle ce morceau de code « *Interrupt Service Routine* » (**ISR**). Nous verrons plus loin quoi mettre dedans.

On a donc un moyen de faire quelque chose toutes les 32 µs. Ce n'est pas vraiment ce qu'on veut. Nous devons nous allumer un feu pendant plusieurs secondes ! Le timer va bien trop vite, il faut le ralentir. On utilise pour cela un diviseur (*prescaler*). Il s'agit d'un mécanisme permettant de générer une horloge pour le timer à partir de l'horloge du microcontrôleur, en la divisant par une certaine valeur, généralement 8, 64, 256 ou 1024. Pour cela, il va nous falloir configurer le registre de contrôle B du timer0 (*Timer Counter Control Register 0 B* ou `TCCR0B`, voir page 79 de la documentation du microcontrôleur **[2]**). Il s'agit d'un registre 8 bits, dont seuls les trois derniers (numérotés 0, 1 et 2) nous intéressent ici :

Fig. 4 : Structure du registre TCCR0B. Ce registre permet de contrôler la source d'horloge du timer.

TCCR0B – Timer/Counter Control Register B

Bit	7	6	5	4	3	2	1	0	
0x33	FOC0A	FOC0B	–	–	WGM02	CS02	CS01	CS00	TCCR0B
Read/Write	W	W	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 11-6. Clock Select Bit Description

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	$clk_{I/O}$/(No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on T0 pin. Clock on falling edge.
1	1	1	External clock source on T0 pin. Clock on rising edge.

Fig. 5 : Description des bits de sélection du diviseur pour l'horloge du timer.

Pour choisir le diviseur du timer 0, on utilise le tableau ci-dessus (Figure 5), issu de la page 80 de la documentation du microcontrôleur.

Et là, c'est relativement facile. Pour un diviseur de 256 (pour que ça compte 256 fois moins vite donc), il suffit de mettre le bit **CS02** à 1, et de laisser les bits **CS01** et bit **CS00** à 0.

On utilise alors simplement la ligne de code suivante :

```
// mesure du temps par interruption
// on configure le diviseur sur 256
TCCR0B |= (1<<CS02);
```

On aura donc une interruption toutes les 256x256 / $8x10^6$ = 8.192 ms. On est encore loin de la durée de plusieurs secondes pour la durée d'affichage des feux, mais on peut utiliser un compteur pour compter les interruptions. En effet, avec une interruption toutes les 8.192 ms, il y aura 1 / $8.192x10^3$ = 122 interruptions par seconde. Prenons le cas du feu vert par exemple, que l'on veut afficher pendant 10 secondes. Il suffira d'incrémenter un compteur à chaque itération, et de surveiller lorsque ce compteur atteint la valeur 10x122 = 1220. On sait alors que 10 s se seront écoulées, et qu'il est temps d'éteindre le feu vert, et d'allumer le feu jaune.

On commence par initialiser le compteur en début de programme :

```
// compteur pour les itérations d'interruptions
int its_counter = 0;
```

Bien, les choses commencent à se mettre en place. Nous avons un mécanisme pour mesurer le temps, qui ne fait pas travailler le microprocesseur. Nous avons vu que toutes les 8.192 ms, le timer atteint sa valeur maximale, et une interruption OVERFLOW est générée. Nous allons regarder maintenant comment activer les interruptions pour que le microprocesseur puisse exécuter le code qui va bien.

Pour activer les interruptions de débordement (OVERFLOW), il faut configurer le registre de masque du timer **TIMSK** (*Timer Interrupt Mask Register*). La page 81 de la documentation du microcontrôleur indique la signification de chaque bit de ce registre :

Fig. 6 : Structure du registre TIMSK. Le bit TOIE0 permet d'activer l'interruption lors d'un débordement du timer.

TIMSK – Timer/Counter Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
0x39	–	OCIE1A	OCIE1B	OCIE0A	OCIE0B	TOIE1	TOIE0	–	TIMSK
Read/Write	R	R/W	R/W	R/W	R/W	R/W	R/W	R	
Initial Value	0	0	0	0	0	0	0	0	

coromonadalix

Pour utiliser le débordement du timer 0, nous devons mettre à 1 le bit 1 **TOIE0** (*Timer Overflow Interrupt Enable 0*), avec la ligne suivante :

```
// on active les interruptions pour débordement du compteur 0
TIMSK |= (1<<TOIE0);
```

Il ne reste plus qu'à activer les interruptions globalement avec la fonction **sei()** :

```
// on active les interruptions
sei();
```

Et maintenant le programme principal. Eh bien en fait, il ne fait rien de particulier, car tout sera géré par les interruptions, dont nous verrons le détail un peu plus loin. On ne peut pas dire que le microprocesseur soit débordé !

```
// le programme est une boucle infinie, qui ne fait rien !
while(1){};

return 0;
}
```

2.5 La gestion des feux

Nous allons gérer notre programme comme une machine d'état. Le programme peut se trouver dans quatre états différents : feu vert, feu jaune, feu rouge, et alarme activée. Nous définissons pour cela une énumération, qui par défaut vaut vert :

```
// états possibles
enum {vert,jaune,rouge,alarme} etat = vert;
```

Les changements d'état seront gérés avec une instruction de branchement **switch**. Pour chaque état, on regarde si la durée prévue est écoulée. Si ce n'est pas le cas, on ne fait rien et on attend la prochaine interruption du timer. Si c'est le cas, on remet le compteur à zéro, on met à jour l'état des feux, et on passe à l'état suivant.

La macro **ISR(vector)** permet d'indiquer quel code exécuter lorsqu'une interruption est levée. **vector** permet de préciser le type d'interruption concernée. La liste des vecteurs possibles est donnée dans la référence **[3]**. Dans notre cas, il s'agit de **TIM0_OVF_vect**, qui correspond donc à un débordement du timer 0.

Voici le code gérant l'affichage des feux et les changements d'état :

```
// macro qui s'exécute lors d'un débordement du compteur 0
// (TIMer 0 Overflow)
ISR(TIM0_OVF_vect){
 // on désactive les interruptions pour ne pas boucler
 cli();

 // on teste les différents états
 switch(etat){
  case vert:
   if(++its_counter>=(122*DELAY_VERT)){
```

```
    its_counter = 0;
    // on éteint les leds verte et rouge et on allume la jaune
    PORTB &=~ (1<<LED_VERTE);
    PORTB |=  (1<<LED_JAUNE);
    PORTB &=~  (1<<LED_ROUGE);
    etat = jaune;
   };
   break;
  case jaune:
   if(++its_counter>= (122*DELAY_JAUNE)){
    its_counter = 0;
    // on éteint les leds verte et jaune et on allume la rouge
    PORTB &=~ (1<<LED_VERTE);
    PORTB &=~ (1<<LED_JAUNE);
    PORTB |=  (1<<LED_ROUGE);
    etat = rouge;
   };
   break;
  case rouge:
   if(++its_counter>=(122*DELAY_ROUGE)){
    its_counter = 0;
    // on éteint les leds jaune et rouge et on allume la verte
    PORTB |=  (1<<LED_VERTE);
    PORTB &=~ (1<<LED_JAUNE);
    PORTB &=~ (1<<LED_ROUGE);
    etat = vert;
   };
   break;
```

Notez au passage que l'on désactive toutes les interruptions avec la commande `cli()` en réinitialisant le masque global des interruptions. Il ne faudra pas oublier de les réactiver lorsque le traitement de l'interruption est terminé ! Remarquez également que l'on retrouve le facteur 122, qui est le nombre d'interruptions générées par seconde. Lorsque le feu vert est allumé, il faudra lever 1220 interruptions avant de véritablement faire quelque chose, ici, passer dans l'état jaune.

Il ne nous reste plus qu'à fermer l'instruction `switch`, et à réactiver les interruptions avec la fonction `sei()`.

```
  // on réactive les interruptions
  sei();
}
```

2.6 Le bouton d'alarme

Nous avons maintenant des feux de signalisation qui changent au cours du temps en fonction des durées qui ont été définies au début du programme. Tout cela est géré par les interruptions, et tout fonctionne bien. Reste maintenant à rendre possible l'activation de l'alarme. Nous avons prévu dans le montage un bouton poussoir pour cela. Encore faut-il aller vérifier régulièrement si le bouton est appuyé ou non. Nous pourrions le faire dans la boucle principale du programme (la boucle qui ne fait rien), mais ce ne serait pas bien malin. Autant utiliser là aussi les interruptions,

et en particulier les interruptions qui sont capables de détecter un changement de tension sur une des broches du microcontrôleur, car c'est exactement ce qu'il se passera lorsque l'on appuiera sur le bouton d'alarme.

Pour activer les interruptions liées à un changement sur une broche, il faut configurer le registre du masque d'interruption général **GIMSK** (*General Interrupt Mask*), présenté page 51 de la documentation du microcontrôleur :

GIMSK – General Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
0x3B	–	INT0	PCIE	–	–	–	–	–	GIMSK
Read/Write	R	R/W	R/W	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

Fig. 7 : Structure du registre GIMSK. Ce registre permet d'autoriser les interruptions en provenance des entrées d'interruption externes ou bien lors de changements d'état de certaines broches.

Il faut alors mettre à 1 le bit **PCIE** (*Pin Change Interrup Enable*) :

```
// on active les interruptions par changement d'état des broches du portB
GIMSK |= (1<<PCIE);
```

Bien. Reste à indiquer maintenant qu'on ne veut surveiller que la broche sur laquelle est connecté le bouton poussoir, et pas toutes les broches, sinon on risque de lever des interruptions inutilement. C'est le registre de masque de changement de broche **PCMSK** (*Pin Change Mask Register*) qui s'occupe de ça. C'est détaillé page 52 de la documentation du microcontrôleur :

PCMSK – Pin Change Mask Register

Bit	7	6	5	4	3	2	1	0	
0x15	–	–	PCINT5	PCINT4	PCINT3	PCINT2	PCINT1	PCINT0	PCMSK
Read/Write	R	R	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Fig. 8 : Structure du registre PCMSK. Ce registre permet d'indiquer les broches sur lesquelles il faut surveiller le changement d'état.

Il faut juste passer le bit **PCINT4** à 1, car il correspond à la broche PB4 sur laquelle le bouton poussoir est connecté :

```
// seul le changement d'état sur PB4 déclenchera une interruption
PCMSK |= (1<<PCINT4);
```

Dans ces conditions, une interruption sera donc levée à chaque changement d'état de la broche PB4, c'est-à-dire lorsqu'on appuie sur le bouton d'alarme, mais aussi lorsqu'on le relâche. Ce sera de la responsabilité du code prenant en charge l'interruption de détecter s'il s'agit d'un bouton enfoncé ou relâché, et d'agir en conséquence. Nous allons voir ça dans la suite.

2.7 Activation et désactivation de l'alarme

Comme tout à l'heure avec le timer, nous allons utiliser la macro **ISR()** qui permet d'indiquer quel code exécuter lorsqu'une interruption est levée. Cette fois, il s'agit du vecteur **PCINT0_vect**, qui correspond à un changement d'état sur l'une des broches du microcontrôleur. Notez que la broche sur laquelle le changement d'état s'est produit n'est pas indiquée, et que ce sera au code appelé dans la macro de déterminer ce qu'il s'est passé. Ici, pas de soucis puisque seul un changement d'état sur la broche PB4 déclenche une interruption (souvenez-vous du registre **PCMSK** défini plus haut).

Sans plus tarder, voici le code de la macro **ISR** gérant l'interruption :

```
ISR(PCINT0_vect){
 cli();
 // on attend 20 ms pour éviter les rebonds au niveau du contact
 // du bouton poussoir
 _delay_ms(20);

 // si c'est un appui et pas un relâchement
 if(!(PINB & (1<<PINB4)))
 {
   // si l'alarme était déjà activée, on la désactive et on revient
   // dans un état normal
   if(etat == alarme){
  etat = vert;
  // on allume la led verte et on éteint les autres
  PORTB |=  (1<<LED_VERTE);
  PORTB &=~ (1<<LED_JAUNE);
  PORTB &=~ (1<<LED_ROUGE);
  its_counter = 0;}
   // sinon, on active l'alarme
   else{
  ledRouge = 1;
  etat = alarme;}
 }
 sei();
}
```

Comme tout à l'heure, on commence par désactiver les interruptions avec la fonction **cli()**, le temps de traiter celle qui vient d'apparaître. On attend ensuite une vingtaine de millisecondes pour éviter les rebonds au niveau du contact du bouton poussoir et ainsi lire une valeur de tension fiable sur la broche 4.

La lecture de la tension sur la broche 4, parlons-en justement. Il faut que notre programme réagisse à une pression sur le bouton (passage de la broche 4 à la masse), mais ne fasse rien de spécial lors du relâchement de ce bouton (passage à 5V). Pour lire une valeur numérique sur les broches du microcontrôleur (valeur binaire **HIGH** ou **LOW**, 5V ou 0V), il suffit d'interroger le registre **PINB**. Sur les huit bits de ce registre, seuls les six premiers ont une signification. Inutile de donner la tension sur les broches VCC et GND du microcontrôleur, on les connaît déjà !

La page 60 de la documentation du microcontrôleur donne des informations sur ce registre :

PINB – Port B Input Pins Address

Bit	7	6	5	4	3	2	1	0	
0x16	–	–	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	PINB
Read/Write	R	R	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	N/A	N/A	N/A	N/A	N/A	N/A	

Fig. 9 : Structure du registre d'entrée du port B : PINB. Ce registre donne accès à l'état de chacune des broches du microcontrôleur.

Pour récupérer l'état logique de la broche 4, il faut récupérer l'état du bit 4 du registre PINB. Cette opération est réalisée en utilisant un masque. On effectue un ET logique (**&**) entre le registre PINB et le masque **1<<PINB4** (qui correspond au nombre binaire **00010000**). Le résultat correspond à l'état de la broche 4. Comme on ne veut traiter l'interruption que si cette broche est à la masse (appui sur le bouton), on considère l'opération contraire grâce à l'opérateur NOT (**!**).

Le reste du code est assez facile à comprendre. Si l'état antérieur du système était alarme, alors il faut couper l'alarme. L'état suivant devient donc vert, on gère l'affichage des feux, et on réinitialise le timer. Sinon, on active l'alarme.

Pour cette alarme justement, on voudrait faire clignoter la diode rouge. Pour cela, un booléen **ledRouge** va conserver l'état de la diode rouge (allumée ou éteinte) pour permettre le clignotement. Il faut évidemment déclarer cette variable en début de programme :

```
// booléen pour le clignotement de la led rouge
int ledRouge = 1;
```

Et voilà pour la gestion de l'activation ou la désactivation de l'alarme. Bien sûr, ça ne dit pas comment gérer l'alarme proprement dite. C'est ce que nous allons voir maintenant.

2.8 Définition de l'alarme

L'alarme est activée lorsque l'état du système vaut **alarme**. C'est ce que nous venons de gérer dans la partie précédente. Reste maintenant à définir ce qu'est cette alarme. Il nous faut juste ajouter un cas supplémentaire dans la fonction **switch()** de la macro **ISR(TIM0_OVF_vect)**, qui gérait déjà les feux.

On ajoute donc le code suivant à la suite des autres cas :

```
  case alarme:
   // on éteint les leds vert et jaune
   PORTB &=~ (1<<LED_VERTE);
   PORTB &=~ (1<<LED_JAUNE);
   // si la led rouge est allumée, on émet un son
   if(ledRouge){
    PORTB |= (1<<LED_ROUGE);
    int i;
    for(i=0;i<500*DEMI_PERIODE_ALARME;i++){
```

```
    PORTB |=  (1<<BUZZER);
    _delay_ms(1);
    PORTB &=~ (1<<BUZZER);
    _delay_ms(1);}
    PORTB &=~ (1<<LED_ROUGE);
    ledRouge = 0;
  }
  // sinon on attend
  else
  {
    _delay_ms(1000*DEMI_PERIODE_ALARME);
    ledRouge = 1;
  }
  break;
```

Pour rappel, l'alarme consiste à faire clignoter la led rouge à une fréquence de 0,5 Hz. Pendant une seconde, la led rouge est allumée et le buzzer émet un son, puis, pendant la seconde suivante, on éteint la led rouge et le son est coupé, puis on recommence.

Pour générer un son avec le buzzer, nous allons simplement le soumettre à un signal carré de fréquence 500 Hz environ. On met donc la broche 2 à 5V pendant une ms, puis à la masse pendant 1 ms également, puis on recommence 500 fois à l'aide d'une **`boucle for`**. Le buzzer émet donc un son pendant 500 x 1 x 2 ms = 1s, ce qui est bien ce que l'on souhaite. Ensuite, on éteint la led rouge, et on passe le booléen **`ledRouge`** à 0 pour la prochaine interruption. Dans ce cas, on se contente d'attendre la durée de la demi-période de l'alarme, soit une seconde également, puis on rallume la led rouge, etc.

Un autre appui sur le bouton permet de passer de l'état alarme à l'état vert, et donc de couper l'alarme. Notez que l'alarme n'est pas gérée ici avec des interruptions, mais par l'utilisation de la fonction **`_delay_ms()`**. Cela signifie que lors d'un appui sur le bouton, il faut peut-être attendre jusqu'à une seconde au maximum pour quitter le mode alarme. Il aurait été toutefois tout à fait possible d'utiliser des interruptions pour gérer l'arrêt de l'alarme.

3. COMPILATION ET ENVOI DU PROGRAMME DANS LE MICROCONTRÔLEUR

On peut facilement utiliser un Arduino Uno pour programmer le microcontrôleur attiny85. Je ne vais pas détailler ici la façon de procéder. De nombreux tutoriels existent sur le net, par exemple dans cet article de blog **[4]**, ou encore page 14 du numéro 1 de *Hackable Magazine* (juillet-août 2014).

Voici le schéma que j'ai utilisé : voir Figure 10, page suivante.

Sur mon système GNU/Linux Debian, j'ai également installé les paquets suivants : **`gcc-avr`**, **`binutils-avr`**, **`gdb-avr`**, **`avr-libc`** et **`avrdude`**.

coromonadalix

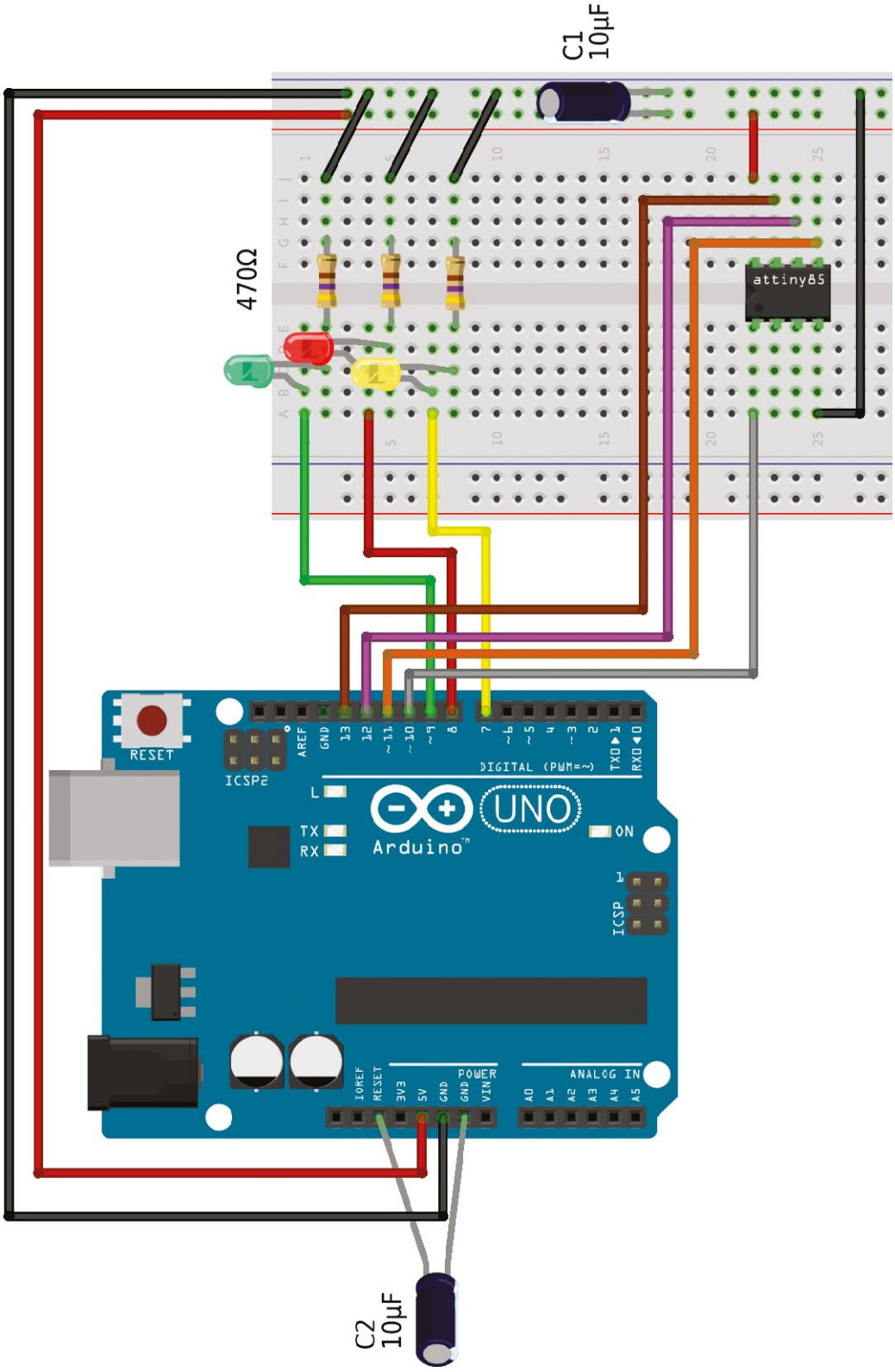


Fig. 10 : Montage utilisant un Arduino Uno pour programmer le microcontrôleur ATTINY85.

coromonadalix

Enfin, pour compiler le fichier source `feux.c`, j'ai utilisé un fichier `Makefile`. Les deux fichiers peuvent être téléchargés sur le GitHub du magazine : **https://github.com/Hackable-magazine**.

Assurez-vous que le fichier `Makefile` est bien dans le même dossier que le fichier source `feux.c`. Il suffit ensuite de lancer la compilation du code source avec la commande :

```
$ make
avr-gcc -g -DF_CPU=8000000UL -Wall -Os -mmcu=attiny85    -c -o feux.o feux.c
avr-gcc -g -DF_CPU=8000000UL -Wall -Os -mmcu=attiny85  -Wl,-Map,feux.map -o feux.elf
feux.o
avr-objdump -h -S feux.elf > feux.lst
avr-objcopy -j .text -j .data -O ihex feux.elf feux.hex
avr-objcopy -j .text -j .data -O binary feux.elf feux.bin
avr-objcopy -j .text -j .data -O srec feux.elf feux.srec
```

Pour configurer les fusibles de configuration du microcontrôleur (*fuses*), par exemple pour modifier la fréquence d'horloge du microcontrôleur, il suffit de saisir `make fuse`. Si tout se passe bien, vous devriez obtenir à l'écran quelque chose comme ceci :

```
$ make fuse
avrdude -p t85 -c avrisp -P /dev/ttyUSB0 -b 19200 -v \
-U lfuse:w:0xe2:m -U hfuse:w:0xdf:m -U efuse:w:0xff:m

...

avrdude: safemode: lfuse reads as E2
avrdude: safemode: hfuse reads as DF
avrdude: safemode: efuse reads as FF
avrdude: safemode: Fuses OK (E:FF, H:DF, L:E2)

avrdude done.  Thank you.
```

Enfin, pour terminer, l'envoi du programme compilé se fait avec la commande :

```
$ make install
```

C'est prêt. Le microcontrôleur est programmé. Il ne reste plus qu'à le mettre sur le circuit, à l'alimenter, et à appeler les enfants ! LG

LIENS

[1] http://blog.cicatrice.eu/119

[2] http://www.atmel.com/images/atmel-2586-avr-8-bit-microcontroller-attiny25-attiny45-attiny85_datasheet.pdf

[3] http://www.nongnu.org/avr-libc/user-manual/group__avr__interrupts.html

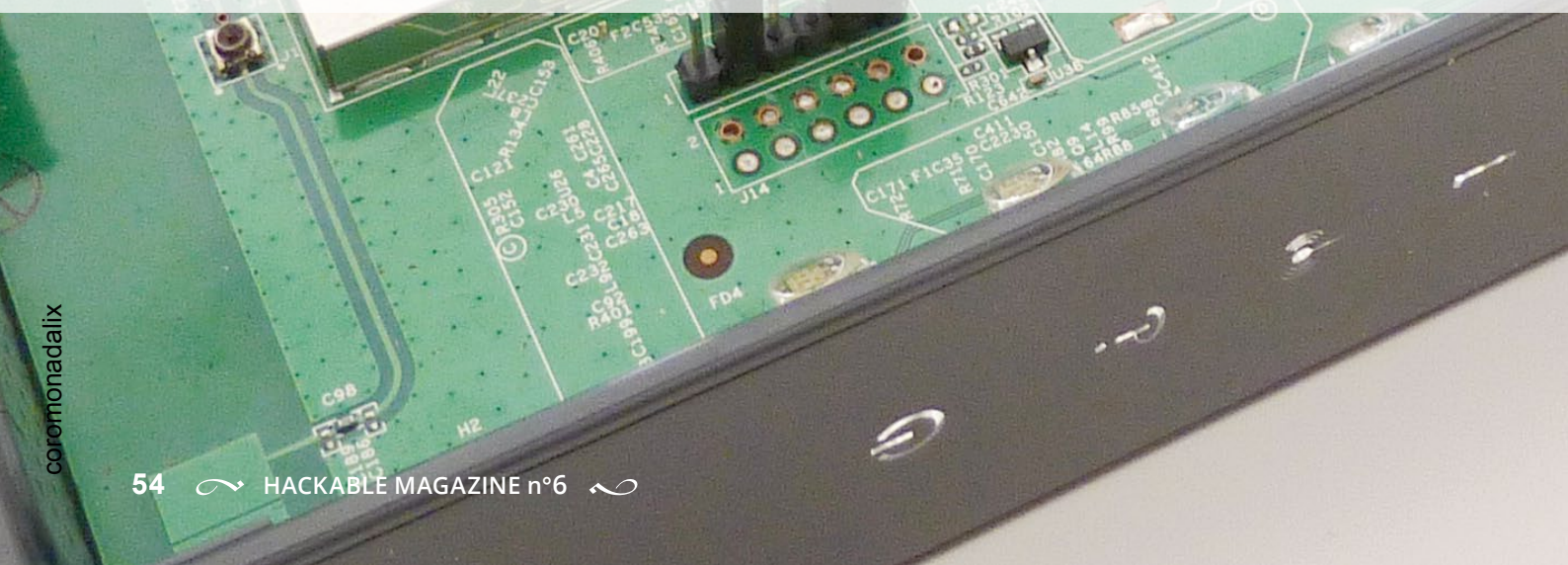
[4] http://blog.cicatrice.eu/140

coromonadalix

UNE RASPBERRY PI POUR PROTÉGER VOTRE VIE PRIVÉE : LA PI EN POINT D'ACCÈS WIFI

Denis Bodor

L'anonymat sur Internet est plus que jamais un sujet de polémique, de discussion et d'inquiétude. Les débats récents autour de la mise en place d'une nouvelle loi sur la surveillance, aux contours potentiellement flous, ne font que réaffirmer un réel besoin de protection et d'assurance. Que diriez-vous de disposer d'un point d'accès fournissant automatiquement une connexion (relativement) protégée et surtout anonymisée à Internet ? Mais avant cela encore faut-il préparer notre plateforme, une Raspberry Pi, en la transformant en point d'accès Wifi standard.



coromonadalix

Afin de ne pas rendre l'article trop volumineux, nous baserons nos explications sur une carte Raspberry Pi, une B+ pour l'exemple, d'ores et déjà fonctionnelle et connectée à Internet par une connexion filaire (Ethernet). L'ensemble de cet article concernera naturellement la ligne de commandes, puisqu'il s'agit de configuration relativement avancée, et nous ne ferons donc absolument aucun usage de l'interface graphique. Il est donc recommandé de configurer la Raspberry Pi en conséquence (mémoire vidéo, démarrage simple) et d'utiliser directement la ligne de commandes via la console (clavier + écran HDMI), un accès à distance avec SSH/PuTTY ou encore un adaptateur USB/série (cf article dans le numéro précédent).

Avant toutes choses, assurons-nous que le réseau et l'accès à Internet fonctionnent en *pingant* un site au hasard :

```
$ ping -c 2 www.google.com
PING www.google.com (216.58.211.68) 56(84) bytes of data.
64 bytes from par03s14-in-f4.1e100.net (216.58.211.68):
  icmp_req=1 ttl=57 time=18.7 ms
64 bytes from par03s14-in-f4.1e100.net (216.58.211.68):
  icmp_req=2 ttl=57 time=19.7 ms

--- www.google.com ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 18.763/19.273/19.783/0.510 ms
```

Une autre vérification possible consiste à simplement utiliser la commande **`sudo apt-get update`** afin de mettre à jour la liste des paquets (logiciels et bibliothèques) disponibles. Si tout fonctionne, nous pouvons aller plus avant et nous intéresser au Wifi.

1. DU WIFI ?

La Raspberry Pi n'est pas équipée en standard d'une interface Wifi. Ajouter cette fonctionnalité passe donc par l'utilisation d'un périphérique USB compatible. Au jour d'aujourd'hui et contrairement à la situation d'il y a quelques années, la grande majorité des adaptateurs Wifi fonctionnera quasi automatiquement sous GNU/Linux. Généralement, une clé USB Wifi repose sur un firmware, un petit bout de code que le pilote doit charger dans le périphérique avant de l'utiliser. Ceux-ci sont livrés avec la distribution Raspbian sous la forme des paquets **`firmware-realtek`**, **`firmware-brcm80211`**, **`firmware-ralink`**, etc.

Le plus simple pour savoir si votre clé Wifi est reconnue est de simplement la brancher et d'utiliser la commande :

```
$ ifconfig -a
eth0      Link encap:Ethernet  HWaddr b8:27:eb:1e:86:aa
          inet adr:192.168.10.236  Bcast:192.168.10.255
Masque:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1450  Metric:1
          RX packets:10679 errors:0 dropped:0 overruns:0 frame:0
          TX packets:5666 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 lg file transmission:1000
          RX bytes:7882972 (7.5 MiB)  TX bytes:508625 (496.7 KiB)
```

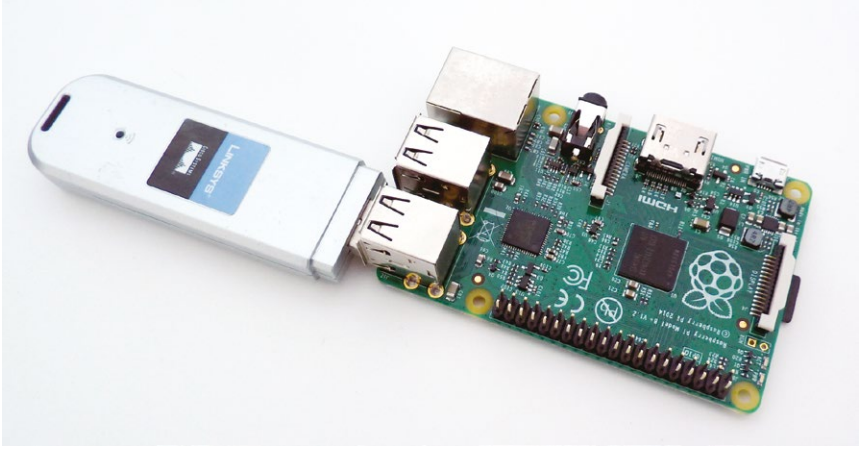
coromonadalix

```
lo        Link encap:Boucle locale
          inet adr:127.0.0.1  Masque:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 lg file transmission:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

wlan0     Link encap:Ethernet  HWaddr 00:22:6b:a9:32:90
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 lg file transmission:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

Nous avons ici trois interfaces réseau : **eth0** est la connexion filaire Ethernet, **lo** est l'interface locale (*loopback*) et enfin **wlan0** est notre périphérique Wifi. Notez que certains adaptateurs/pilotes ne chargent le firmware qu'à l'activation du périphérique. Un **ifconfig wlan0 up** (puis **down**) permet de rapidement vérifier ce point. Un petit coup de **dmesg | tail** apportera également quelques informations utiles :

```
[3.824819] usb 1-1.5: Product: Compact Wireless-G USB Adapter
[3.846788] usb 1-1.5: Manufacturer: Cisco-Linksys
[4.366236] udevd[158]: starting version 175
[6.081070] cfg80211: Calling CRDA to update world regulatory domain
[7.266880] usb 1-1.5: reset high-speed USB device number 4 using dwc_otg
[7.831620] ieee80211 phy0: rt2x00_set_chip: Info -
  Chipset detected - rt: 2573, rf: 0002, rev: 000a
[8.033477] usbcore: registered new interface driver rt73usb
```

Nous avons ici un adaptateur USB Cisco/Linksys WUSB54GC, supporté par le pilote Ralink RT73 (firmware **/lib/firmware/rt73.bin**). Il n'y a eu aucune erreur et on peut donc supposer que le périphérique fonctionnera parfaitement. On remarquera aussi qu'il est fait mention de **cfg80211**, chose qui nous sera utile par la suite puisque cela indique que le périphérique est pris en charge par l'interface/API **nl80211**. Sans entrer dans le détail, on résumera cela en disant que ce nom nous indique que le système gère l'interface d'une certaine manière. Beaucoup de pilotes aujourd'hui et donc de périphériques reposent sur **nl80211**/**cfg80211**.



Une Raspberry Pi B ou B+ et un adaptateur Wifi USB, il n'en faut pas plus pour créer un point d'accès permettant bon nombre d'applications. Il existe des adaptateurs USB bien plus compacts que le Cisco/Linksys WUSB54GC un peu ancien (2008) présenté ici. Il présentait l'avantage, à l'époque, d'être une valeur sûre en termes de compatibilité GNU/Linux et de fonctionnalité (mode monitor), j'en avais donc acheté 3...

coromonadalix

2. WIFI OUI, MAIS PAS COMME CLIENT

Nous n'avons pas l'intention de faire en sorte que la Raspberry Pi puisse se connecter à un point d'accès. Elle ne sera pas cliente, mais serveur. Pour cela, elle doit savoir faire deux choses : permettre les connexions Wifi sécurisées et donner des adresses aux machines qui les utilisent. La première passe par l'installation du paquet **hostapd** et la seconde par celle de **isc-dhcp-server**. On installera le tout avec **sudo apt-get install hostapd isc-dhcp-server**.

Il est probable que vous obteniez un message d'erreur :

```
Starting ISC DHCP server: dhcpdcheck syslog for diagnostics. ...
failed!
failed!
invoke-rc.d: initscript isc-dhcp-server, action "start" failed.
```

Ceci ne signifie pas que l'installation a échoué, mais simplement que le serveur DHCP (celui qui donne les adresses) ne s'est pas lancé. C'est parfaitement normal puisqu'il n'est pas configuré par défaut. C'est la première chose que nous devrons faire. Pour cela, vous devez éditer (avec **sudo nano** par exemple, mais je préfère largement Vi) le fichier **/etc/dhcp/dhcpd.conf**. Les lignes débutant par **#** sont des commentaires :

```
# pas de gestion DDNS
ddns-update-style none;
# adresses données pour 600s
default-lease-time 600;
# maximum de temps accepté
max-lease-time 7200;
# c'est moi le chef
authoritative;
# les messages vont dans le journal sys
log-facility local7;

# voici mon réseau
subnet 192.168.74.0 netmask 255.255.255.0 {
  # je donne des adresses dans cette plage
  range 192.168.74.10 192.168.74.100;
  # l'adresse de diffusion
  option broadcast-address 192.168.74.255;
  # adresse du routeur (moi-même)
  option routers 192.168.74.1;
  # mon nom de domaine (réseau local)
  option domain-name "local";
  # les serveurs DNS à utiliser
  option domain-name-servers 8.8.8.8, 8.8.4.4;
}
```

DHCP est plus qu'un protocole qui donne des adresses aux machines d'un réseau. Il peut également leur préciser la configuration à utiliser. Notre bloc entre **{}** regroupe les informations que nous envoyons aux machines qui nous demandent une adresse. Ainsi, elles sauront également à qui passer les connexions vers des machines hors de notre réseau ou encore quels serveurs utiliser pour obtenir une adresse à partir d'un nom (DNS, ici ceux de Google).

Mais la configuration ne s'arrête pas là. Il nous reste à préciser au serveur DHCP l'interface à utiliser. En effet, sur la connexion Ethernet (**eth0**) transite également des informations DHCP : c'est

coromonadalix

justement de cette manière que la Raspberry Pi obtient une adresse et sa configuration réseau (de la part de la box généralement). Si nous nous mettons à répondre aux demandes DHCP sur **eth0** on risque de semer une belle pagaille. Par défaut donc notre serveur DHCP n'écoute sur aucune interface, à nous de lui préciser la bonne.

Éditez le fichier **/etc/default/isc-dhcp-server** et changez la ligne **INTERFACES=""** en ajoutant le nom de l'interface Wifi, exemple : **INTERFACES="wlan0"**. Mais nous ne pouvons toujours pas lancer le serveur DHCP, nous obtiendrions une erreur. L'interface Wifi, **wlan0** existe bien, mais elle n'est pas configurée. Nous n'avons précisé nulle part quelle est son adresse et comme c'est nous le serveur, elle ne risque pas d'en obtenir une automatiquement.

La configuration se passe au même endroit que celui de toutes les interfaces réseau. Éditez donc **/etc/network/interfaces** et supprimez toutes les lignes concernant **wlan0** et le Wifi. Cela se résume généralement à :

```
allow-hotplug wlan0
iface wlan0 inet manual
wpa-roam /etc/wpa_supplicant/wpa_supplicant.conf
```

Ces lignes concernent une utilisation Wifi en mode client et ne nous intéressent donc pas. Notre configuration sera statique, car nous choisissons arbitrairement l'adresse à utiliser. Ajoutez donc ceci dans le fichier :

```
auto wlan0
iface wlan0 inet static
  address 192.168.74.1
  netmask 255.255.255.0
```

L'interface sera automatiquement configurée au démarrage du système, mais il n'est pas nécessaire de redémarrer pour autant. Vous pouvez activer l'interface avec les commandes :

```
$ sudo ifdown wlan0
$ sudo ifup wlan0
```

La commande **ifconfig** nous affiche maintenant bien l'adresse de l'interface :

```
$ ifconfig wlan0
wlan0     Link encap:Ethernet  HWaddr 00:22:6b:a9:32:90
          inet adr:192.168.74.1  Bcast:192.168.74.1
Masque:255.255.255.255
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0
carrier:0
          collisions:0 lg file transmission:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

J'ai remarqué que la gestion du *hotplug* pouvait interférer grandement dans la configuration manuelle et statique du réseau. Il est de bon ton donc de supprimer le paquet **ifplugd** avec la commande **sudo apt-get remove --purge ifplugd** et vous assurer ensuite que le fichier **/etc/network/interfaces** comporte bien un **auto** pour chaque interface. Le mot d'ordre ici est que, étant donné qu'on sait parfaitement ce qu'on fait, il n'y a pas de raison de laisser en place des mécanismes de configuration automatiques. Un fichier **interfaces** propre et bien rangé ressemblera à ceci :

coromonadalix

```
auto lo eth0 wlan0

iface lo inet loopback

iface eth0 inet dhcp

iface wlan0 inet static
  address 192.168.74.1
  netmask 255.255.255.0
```

Enfin, nous pouvons démarrer le serveur DHCP avec :

```
$ sudo service isc-dhcp-server restart
[FAIL] Stopping ISC DHCP server: dhcpd failed!
[ ok ] Starting ISC DHCP server: dhcpd.
```

Le premier *FAIL* est normal puisque le service n'étant pas lancé il ne peut être arrêté avant un nouveau démarrage. L'important est la seconde ligne qui doit afficher **[ok]**.

3. LA DERNIÈRE BRIQUE

Génial ! Maintenant dès qu'une machine demandera une adresse via l'interface Wifi **wlan0** nous lui donnerons une adresse et toutes les informations utiles. Mais... heu... comment font ces machines pour se connecter et faire la demande ? Elles ne peuvent pas, du moins pour l'instant.

Nous avons installé **hostapd** mais, lui non plus n'est pas configuré par défaut. HostAPd est également un serveur et plus exactement un *IEEE 802.1X/WPA/WPA2/EAP Authenticator* doublé d'un point d'accès. C'est lui qui est chargé de la gestion des connexions Wifi et surtout d'authentifier ceux qui demandent une connexion.

Le fichier de configuration n'existe pas du tout, il faut le créer de toutes pièces ou partir de l'exemple dans **/usr/share/doc/hostapd/examples/hostapd.conf.gz** (environ 1100 lignes de configuration et de commentaires). Le nom du fichier et son emplacement importent peu, mais **/etc/hostapd/hostapd.conf** est généralement ce qui est le plus adapté.

Voici le contenu de notre fichier (encore une fois, les **#** marquent les lignes en commentaires) :

```
# interface à utiliser
interface=wlan0
# pilote
driver=nl80211
# nom de l'AP
ssid=torAP
# mode Wifi
hw_mode=g
# canal à utiliser
channel=3
# on accepte toutes les MAC
macaddr_acl=0
# on accepte que OSA
auth_algs=1
# on ne cache pas le SSID
ignore_broadcast_ssid=0
# chiffrement WPA2
wpa=2
# la phrase secrète
wpa_passphrase=rototo12
# gestion PSK
wpa_key_mgmt=WPA-PSK
# chiffrement TKIP pour WPA
wpa_pairwise=TKIP
# chiffrement CCMP pour WPA2
rsn_pairwise=CCMP
```

Les commentaires insérés décrivent l'utilité de chaque ligne, reportez-vous au fichier exemple pour les détails et les autres options possibles. Ici nous mettons en place un point d'accès WPA2 WPA-PSK/CCMP, ce qui correspond à du WPA personnel avec un chiffrement fort (AES), la combinaison la plus courante actuellement. Notez la ligne **driver** qui précise le type d'interface avec le matériel. Nous utilisons ici **nl80211** qui correspond au constat que nous avons fait en début d'article à propos de **cfg80211**. Une correspondance avec le support noyau est nécessaire, car le chiffrement et l'authentification reposent sur des fonctionnalités qui se trouvent dans le matériel. Certains adaptateurs Wifi nécessitent un **driver** différent, en cas de problème inspectez le contenu de **/var/log/syslog** et cherchez le nom de votre périphérique accompagné de « hostapd » sur le web pour trouver la bonne ligne.

La commande **`iw list`** peut également vous être utile en cas de problème. Elle listera énormément d'informations sur l'interface Wifi parmi lesquelles les modes disponibles (*Supported interface modes*) ou encore les algorithmes de chiffrement supportés (*Supported Ciphers*).

Comme pour le serveur DHCP, nous devons affiner la configuration vis-à-vis du système en faisant un tour dans **`/etc/default/`**. Nous y éditons le fichier **`hostapd`** afin de préciser la configuration à utiliser. Nous changeons donc la ligne contenant **`#DAEMON_CONF=""`** en supprimant le **`#`** et en ajoutant le fichier que nous avons créé précédemment, ce qui donne une ligne : **`DAEMON_CONF="/etc/hostapd/hostapd.conf"`**. Dès lors, le serveur sait où se trouve notre configuration et l'utilise.

Il ne reste plus qu'à relancer le serveur :

```
$ sudo service hostapd restart
[ ok ] Stopping advanced IEEE 802.11 management: hostapd.
[ ok ] Starting advanced IEEE 802.11 management: hostapd.
```

Wifi Analyzer est une sympathique application Android permettant de rapidement voir les points d'accès Wifi alentour et d'obtenir des informations précises. Voilà qui est parfait pour vérifier que notre configuration fonctionne comme prévu.

À ce stade non seulement un client Wifi peut se connecter, mais il doit aussi recevoir une adresse IP. Le surf est encore impossible par exemple, car la Raspberry Pi ne partage pas sa connexion à Internet, elle fournit juste un accès à son propre réseau 192.168.74.0. Quoi qu'il en soit, vous pouvez déjà vous connecter avec un ordinateur portable, une tablette ou un smartphone et vérifier que tout fonctionne.

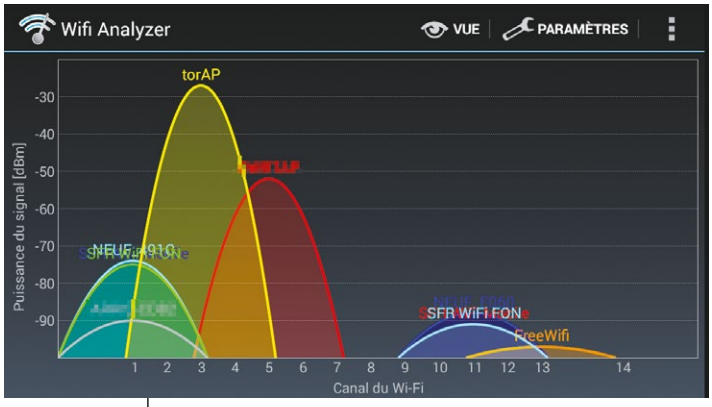
Vous pouvez également vérifier cela du côté de la Raspberry Pi en installant un outil supplémentaire avec **`sudo apt-get install iw`**. Dès lors, afficher la liste détaillée des machines connectées au point d'accès se résume à la commande :

```
$ iw dev wlan0 station dump
Station 00:04:4b:2d:6a:30 (on wlan0)
        inactive time:  2400 ms
        rx bytes:       64072
        rx packets:     776
        tx bytes:       1898
        tx packets:     13
        tx retries:     0
        tx failed:      0
        signal:         -54 dBm
        signal avg:     -53 dBm
        tx bitrate:     5.5 MBit/s
        authorized:     yes
        authenticated:  yes
        preamble:       short
        WMM/WME:        no
        MFP:            no
        TDLS peer:      no
```

coromonadalix

Si vous voulez juste une liste d'adresses, vous pouvez filtrer et obtenir une version courte avec :

```
$ iw dev wlan0 station dump |
grep "^Station"
Station 9c:35:eb:72:7b:e5
(on wlan0)
Station 00:04:4b:2d:6a:30
(on wlan0)
```



L'autre avantage de Wifi Analyzer est de pouvoir réaliser un graphique présentant la qualité du signal de chaque AP en fonction du canal Wifi utilisé. On configurera idéalement le nôtre sur un canal peu utilisé pour assurer un fonctionnement optimal.

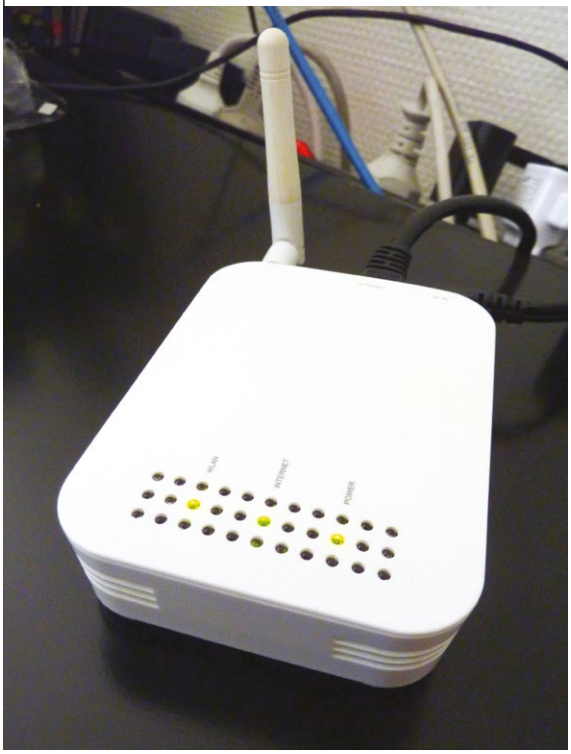
4. FONCTIONNER TEMPORAIREMENT COMME UNE PASSERELLE

« Point d'accès Wifi » ne signifie pas automatiquement « accès à Internet ». Un point d'accès ou AP n'est que ce que le terme désigne : un point d'accès à un réseau. Pour l'instant, notre AP fait très bien ce pour quoi il est conçu en permettant aux machines se connectant en Wifi d'avoir accès au réseau 192.168.74.0. De ce fait, en connectant par exemple deux ordinateurs en Wifi à l'AP, l'un pourra parfaitement accéder au second.

Votre réseau local ou LAN fonctionne exactement de la même manière. Pour sortir du bien nommé réseau local, une machine est chargée d'une tâche particulière. Celle-ci est connectée à deux réseaux, d'un côté le LAN et de l'autre Internet (c'est un peu plus complexe que cela, mais faisons simple). Les machines du LAN sont informées de cet état de fait via DHCP et c'est là que la magie du routage se met en place. Automatiquement, dès qu'un accès à une adresse « extérieure » est demandée, la machine qu'on qualifie généralement de « routeur » ou « passerelle » va faire office de « passeur ». Elle se charge également de recevoir les réponses liées aux connexions et les retransmet vers la machine sur le LAN. Votre box Internet fonctionne de cette manière et fait office à la fois de point d'accès Wifi et de routeur.

En ce qui nous concerne, nous n'avons rien configuré pour procéder à ce genre de choses et c'est donc pour cela que les machines connectées en Wifi n'ont pas accès à Internet (ni même au réseau filaire derrière la Pi). Cependant, lors de la configuration du serveur DHCP, nous avons prévu ce type de chose. La ligne `option routers 192.168.74.1` dans le fichier `/etc/dhcp/dhcpd.conf` précise aux machines qui demandent une adresse que le routeur est 192.186.74.1, qui est l'adresse de l'interface Wifi de la Raspberry Pi.

Les points d'accès Wifi ouverts comme la Fonera ou ce périphérique Meraki un peu ancien sont assez proches de l'architecture que nous utilisons ici. Ils ne sont cependant généralement équipés que de peu de mémoire et d'une faible puissance de calcul. Ceci fonctionne parfaitement pour un simple point d'accès, mais sera souvent insuffisant pour une installation de Tor.



coromonadalix

Tout ce que nous avons à faire pour changer notre point d'accès en point d'accès à Internet est de dire au système de servir de « passeur » entre le réseau Wifi et le réseau filaire. Pour cela, nous commençons par activer la fonctionnalité dans le noyau avec :

```
$ sudo sysctl -w net.ipv4.ip_forward=1
net.ipv4.ip_forward = 1
```

Ce n'est, bien entendu, pas suffisant. Il faut maintenant détailler la façon précise dont nous allons faire ce travail ou en d'autres termes, établir des règles. Pour cela, on utilise la commande **iptables** ainsi :

```
$ sudo iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
$ sudo iptables -A FORWARD -i eth0 -o wlan0 -m state \
--state RELATED,ESTABLISHED -j ACCEPT
$ sudo iptables -A FORWARD -i wlan0 -o eth0 -j ACCEPT
```

Il est possible de faire bien plus précis que cela en ajoutant des règles plus strictes, mais il ne s'agit ici que d'un test rapide. **iptables** est l'outil de contrôle d'un système appelé Netfilter qui permet, comme son nom l'indique, de filtrer le flux réseau et le manipuler. Dans les grandes lignes, nous disons ici au système qu'il doit faire croire que les connexions arrivant par **wlan0** et partant sur **eth0** viennent de lui. Inversement, des connexions déjà établies (les réponses donc) entre **eth0** et **wlan0** doivent également être prises en charge.

Une fois ces trois commandes validées, l'effet est immédiat : toutes les machines se connectant au point d'accès peuvent aller sur Internet. La configuration n'est pas figée, en cas de redémarrage de la Pi, celle-ci sera perdue. Pour atteindre notre objectif qui, je le rappelle, est de créer un accès au réseau Tor, nous n'avons pas besoin de garder cette configuration. Si en revanche vous souhaitez vous arrêter là et avoir un simple point d'accès Wifi qui fasse office de routeur, vous pouvez conserver la configuration en utilisant :

```
$ sudo sh -c "iptables-save > /etc/iptables.up.rules"
```

Le fichier **/etc/iptables.up.rules** contiendra alors les règles actuelles **iptables** qui pourront être rechargées avec **iptables-restore** (nous ne pouvons pas ici utiliser juste **sudo** en raison de la redirection dans un fichier appartenant au super-utilisateur **root**). Nous pouvons même automatiser cela lors du démarrage en modifiant le fichier **/etc/network/interfaces**. Il suffit d'ajouter en dernière ligne pour l'interface **wlan0** :

```
   up iptables-restore < /etc/iptables.up.rules
```

Il existe une méthode plus « propre » dont nous parlerons dans l'article suivant. Il ne faudra pas oublier également de soit modifier le fichier **/etc/sysctl.conf**, soit ajouter un fichier avec un nom terminant par **.conf** dans le répertoire **/etc/sysctl.d/**. Dans les deux cas, la ligne à y placer est :

```
net.ipv4.ip_forward = 1
```

Si en revanche vous souhaitez davantage qu'un simple AP Wifi Internet, inutile de redémarrer la Pi pour annuler la configuration. Vous pouvez simplement utiliser les commandes :

```
$ sudo iptables -F
$ sudo sysctl -w net.ipv4.ip_forward=0
net.ipv4.ip_forward = 0
```

5. LA SUITE DANS UN INSTANT…

Il y a déjà beaucoup de choses importantes et intéressantes dans cet article. Nous avons appris ni plus ni moins la façon de transformer une Raspberry Pi en routeur Wifi, ce qui peut déjà s'avérer utile en soi. Mais ce n'est pas tout, à la différence d'une box ou d'un routeur du commerce (si on l'utilise normalement) vous avez ici tout un florilège d'options qui s'offrent à vous.

Non seulement vous pouvez affiner la configuration avec une précision que ne vous offrira jamais aucune interface graphique (web) de routeur, mais vous pouvez, en plus, commencer à faire des choses tordues, voire franchement malsaines. En effet, le système GNU/Linux sur la Raspberry Pi se trouve dans une position privilégiée : il « voit » tout ce qui passe entre les machines connectées en Wifi et le reste du réseau. Mieux encore, non seulement il « voit », mais en plus il peut changer ou copier ces données.

Ceci demande un peu plus de technicité, mais ouvre des possibilités amusantes. Un excellent exemple est proposé par Pete Stevens, dont la page est devenue presque mythique (**http://www.ex-parrot.com/pete/upside-down-ternet.html**). Ses voisins utilisant sans autorisation son accès Wifi, plutôt que de leur couper l'accès, Pete a décidé de s'amuser : il a configuré son système de manière à inverser ou flouter toutes les images qui passaient par sa connexion.

Résultat : un web à l'envers pour les méchants voisins.

Une déclinaison moins sympathique et surtout absolument illégale si vous touchez aux connexions d'autrui consiste à prendre la place d'un point d'accès, espionner les communications et diriger les connexions vers des fausses pages web. Je précise ici que ce genre de choses ne demande pas un niveau de compétence très important et le simple fait de le savoir, je l'espère, devrait vous rendre plus méfiant dans vos connexions à des points d'accès Wifi que vous ne contrôlez pas (lieux publics, hôtels, restaurants, etc.). Il est toujours important de se rappeler que ce qu'on est capable de faire soi-même peut aussi être fait par d'autres, et ce souvent avec bien plus d'efficacité et de malhonnêteté. **DB**

Une autre option parfaitement viable avec le bon matériel consiste à non pas utiliser une Raspberry Pi ou un autre nano-ordinateur, mais un routeur Wifi du commerce. Ceci demande généralement davantage de travail, car il est nécessaire de prendre le contrôle du système (généralement GNU/Linux) puis de l'adapter à ses besoins. Le projet OpenWRT fournit un système utilisable pour de nombreux matériels et inclus une version adaptée de Tor. C'est le point de départ obligatoire si vous vous lancez dans une telle aventure.

coromonadalix

UNE RASPBERRY PI POUR PROTÉGER VOTRE VIE PRIVÉE : TOR

Denis Bodor

Nous disposons maintenant d'un point d'accès Wifi à base de Raspberry Pi, ce qui est déjà très intéressant. Ce qui l'est encore plus est le fait que, contrairement à une box ou un routeur Wifi du commerce, nous pouvons gérer les connexions comme il nous chante et orienter les flux comme bon nous semble. Nous allons donc tout simplement décider de faire passer tout le trafic non pas directement sur le net, mais dans le réseau d'anonymisation Tor. Plus personne ne verra donc ce qui circule (ou presque)...

coromonadalix

Sous le nom Tor se cache non seulement un logiciel, mais également une architecture, des algorithmes et une communauté de développeurs. « Tor » est l'acronyme de *The Onion Router* (le routeur en oignon) en référence à son fonctionnement interne utilisant un système de couches, comme des pelures d'oignon. Le but du projet est et a toujours été de garantir l'anonymat sur un réseau, et ce depuis 2001, année de la diffusion de la première version. Tor est, de plus, un logiciel libre sous licence très permissive BSD.

On entend souvent parler de Tor et du réseau Tor comme étant un *darknet*. Cette affirmation est fausse et, à mon sens, délibérément faite dans le but de prêter à Tor et ses utilisateurs des intentions critiquables, voire mauvaises (sinon carrément criminelles). Le mot « dark » véhicule une connotation négative, fort pratique pour les opposants à une quelconque forme d'anonymat. Utiliser ce type de qualificatif, noir, obscur, sombre, titille notre instinct de conservation et éveil inconsciemment des craintes remontant aux premières heures de l'humanité. Mais darknet est un réseau privé, ami à ami, dont les utilisateurs sont tous considérés comme de confiance au sein de la structure. Dans le cas de Tor, le réseau n'est en rien privé, mais accessible à tous, donc public, et les utilisateurs ne sont jamais considérés comme « de confiance ». Au contraire, le système est basé exactement sur le principe opposé, selon lequel même les composants relayant les communications dans Tor ne doivent en aucun être en mesure d'accéder au contenu, mais ne font que transmettre des données protégées et anonymes.

L'idée fondamentale derrière Tor est simple : il s'agit de protéger l'identité des personnes, quelles qu'elles soient. Ses détracteurs parlent généralement des utilisateurs souhaitant ainsi protéger leur vie privée comme étant systématiquement des terroristes, des délinquants, des criminels, des intégristes ou encore des pédophiles. De l'autre côté de la barrière idéologique, nous avons les activistes, les journalistes, les lanceurs d'alertes et toutes les personnes qui ont besoin d'anonymat pour mener à bien leurs activités absolument légitimement. Dans cette catégorie entrent également les forces de l'ordre. Ceci peut paraître surprenant au premier regard, mais comment voulez-vous infiltrer un réseau pédophile si les criminels savent parfaitement d'où vous venez et qui vous êtes ?

Rien d'étonnant donc qu'une majeure partie du financement de la maintenance de Tor provienne du gouvernement US, réparti entre le *Broadcasting Board of Governors*, la *National Science Foundation* et le

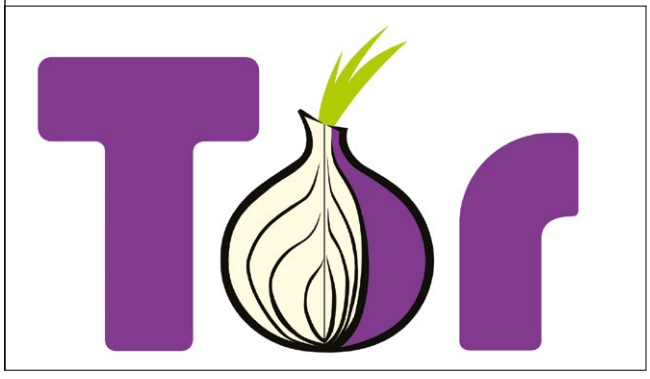
Jacob Appelbaum au 30ème Chaos Communication Congress. Chercheur en sécurité informatique et l'un des principaux développeurs du projet Tor, Jacob est également l'un des porte-paroles du projet concernant les implications techniques, politiques et sociales de cet outil destiné à protéger l'anonymat et contourner les mécanismes de censure. (Wikipedia / Tobias Klenze / CC-BY-SA 3.0 — Travail personnel)

coromonadalix

département d'État. Hé oui, lorsqu'on a des diplomates et des correspondants dans des pays avec des régimes un peu trop curieux, on apprécie la sécurité offerte par Tor. Mais ce n'est pas tout, le reste du financement est assuré par le gouvernement suédois, des organisations non gouvernementales (ONG) et une masse très importante de donateurs individuels (dont bibi).

Bien entendu, la participation de gouvernements dans le financement du projet est tantôt critiquée par certains y voyant une ingérence et un risque de sabotage. Pour ma part, je rappellerai deux choses importantes. Avant d'être l'Internet actuel (celui du « septembre éternel »), le réseau des réseaux est né comme une reprise des travaux sur ARPANET qui était une initiative de la DARPA (*Defense Advanced Research Projects Agency*) et à qui on doit des protocoles comme TCP/IP. Comme l'a rappelé magnifiquement Bruce Sterling à *Transmediale* 2014, les services secrets et l'armée ont toujours été là, bien avant Google, Facebook, Twitter, GeoCities ou même CompuServe. Les origines du réseau et une certaine partie de sa maintenance ne sont en rien le fruit d'activités civiles.

Le second point, et il est capital, est le fait que, si une architecture et les algorithmes sont robustes et fiables, peu importe qui finance et qui participe au projet, son fonctionnement s'adosse à des bases saines, car techniquement valides. Tor est un logiciel libre reposant sur une structure clairement décrite et un ensemble de protocoles publics. N'importe qui peut à loisir venir inspecter le code source et le fonctionnement de l'ensemble, exactement comme avec des logiciels libres comme GnuPG, OpenSSL, OpenSSH ou GNU/Linux. Rien n'est caché, l'ensemble peut être étudié en détail et en profondeur. Bien entendu, des bugs peuvent être présents et c'est là tout l'intérêt d'un code en logiciel libre, ils peuvent être trouvé et rapidement corrigés, contrairement à ceux des logiciels propriétaires reposant sur la seule expertise de ses créateurs.



Le logo du Projet Tor représente un oignon. C'est une image décrivant à elle seule parfaitement le fonctionnement du système qui emballe les communications dans plusieurs couches de chiffrement. Couches successivement pelées pour que le message arrive à destination de façon totalement anonyme.

On notera au passage que dans le projet de loi actuellement étudié (peut-être même adopté au moment où vous lirez ceci) c'est exactement la logique inverse qui est appliquée. En effet, les sondes qui peuvent être posées pour analyser le trafic de métadonnées utilisent des algorithmes et des codes qui ne sont pas publics, mais uniquement étudiés et approuvés par une commission (la CNCTR) qui n'est même pas majoritairement composée d'experts dans le domaine technique.

Ce que propose Tor est de protéger la communication depuis votre machine jusqu'à sa destination. Si quelqu'un ou quelque chose inspecte les données échangées par votre ordinateur, la seule chose visible est un ensemble de connexions vers le réseau Tor. On ne sait pas où vous vous connectez ni ce que vous échangez. À l'intérieur du réseau Tor, les données transitent par différents points. Ces nœuds ne voient que des données chiffrées et ne savent ni d'où elles viennent ni où elles vont. Enfin, la connexion sort du réseau Tor en un point aléatoire pour atteindre sa

coromonadalix

destination. Celle-ci ne voit qu'une connexion provenant du point de sortie et non de votre ordinateur. Le point de sortie lui-même peut voir les données, mais ne sait pas d'où elles proviennent.

Ceci assure une confidentialité relative (nous y reviendrons) et un niveau d'anonymat adapté. Vous comprendrez alors que Tor n'est pas un outil pour criminel et ne fait qu'assurer une protection pour n'importe quelle personne ne voulant pas qu'on la suive à la trace dans toutes ses actions numériques.

J'aime faire l'analogie avec les objets du quotidien. Prenez une porte équipée d'une serrure, par exemple. Elle peut servir à enfermer quelqu'un contre sa volonté ou même à cacher divers objets et méfaits. Mais généralement on l'utilise simplement pour empêcher n'importe qui de rentrer chez soi, de voler et de farfouiller dans ses culottes. Vous n'accepteriez pas qu'on vous accuse de séquestrer des gens ou de cacher des armes et de la drogue sur le simple fait que vous possédez une porte et un trousseau de clés. Utiliser Tor pour vous protéger ne fait pas de vous un criminel, un pédophile ou un terroriste. Juste quelqu'un qui ferme la porte de son domicile...

L'anonymat et la protection de ses communications ne doivent pas seulement être imaginés comme le fait de se cacher derrière un masque et de chuchoter de manière suspecte, au contraire. En mettant des rideaux à vos fenêtres, vous ne cherchez pas à vous cacher, mais souhaitez simplement bénéficier d'intimité. Ce que vous faites dans votre domicile ne regarde que vous, car vous vous sentez soustrait au jugement d'autrui et êtes donc libre de vous comporter en accord avec vos envies. Laissez-vous vous le dire autrement : Tor vous permet d'avoir une intimité dans l'utilisation d'Internet, dans le cas contraire, vous êtes tout nu.

Je terminerai cette partie introductive en précisant que Tor permet également à un grand nombre de personnes à travers le monde d'échapper à la censure. Un certain nombre de pays filtrent l'accès à Internet et tentent de forcer leurs habitants dans la vision dogmatique de leur dirigeant. Tor permet à ces citoyens de découvrir d'autres choses, d'apprendre, de s'informer et de communiquer sans entrave. Nous sommes ici loin de l'image du terroriste pédophile sataniste fondamentaliste...

1. FONCTIONNEMENT DE TOR

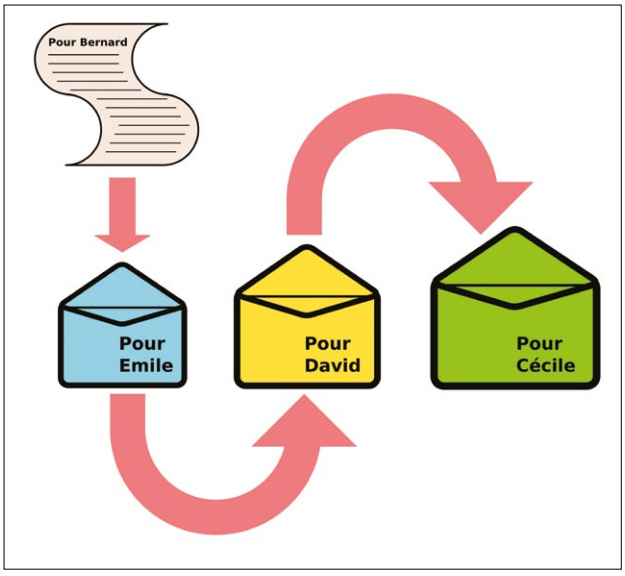
Tor est un réseau de machines participatif. Il se compose d'ordinateurs répartis dans le monde faisant fonctionner le logiciel Tor. Il existe trois types de machines qu'on appelle alors des relais Tor ou encore des routeurs ou des nœuds. Un relais peut être interne et se contenter de faire passer les communications. C'est le passage d'un relais à un autre qui forme le réseau. Un relais interne ne voit ni les données (elles sont chiffrées) ni qui parle avec qui aux deux extrémités de la communication.

Un relais peut également être celui menant au site ou au serveur de destination. On parle alors de relais de sortie (*exit node*). C'est une place particulière dans le réseau Tor, car ce point reçoit les données chiffrées et les déchiffre à destination de la cible qui n'est pas dans le réseau (il existe aussi des serveurs dans le réseau, maladroitement appelés « services cachés », mais ceci sort du cadre de cet article). Cela peut être un simple site Web parfaitement standard, un forum, une boutique, etc. Cette position permet donc au relais d'avoir connaissance des données, ce qui peut sembler être un avantage. Cependant, d'un autre côté, toutes les connexions à destination du site cible semblent provenir de cette machine. C'est là une implication importante en cas de problème. Si la communication est le fruit d'une attaque sur un site, la responsabilité sera en premier lieu celle de la source, le nœud de sortie. De façon similaire, en cas d'activité clairement criminelle comme l'envoi d'images pédophiles sur un forum, là encore, la source apparente sera cette même machine.

coromonadalix

De manière générale, il est extrêmement rare qu'un nœud de sortie « permanent » soit la machine d'un particulier ou d'une entreprise. Le plus souvent, il s'agit de machines dédiées se trouvant dans des universités ou des structures qui sont, par définition, publiques. Il est cependant possible de configurer n'importe quelle machine en point de sortie, mais le plus souvent les machines ou serveurs appartenant à des particuliers ne fonctionnent pas de manière constante ou ne sont actifs que pour une durée limitée. Beaucoup d'hébergeurs considèrent l'installation d'un relais de sortie Tor comme un *open proxy* qui est interdit dans les conditions générales d'utilisation. D'autres permettent l'installation, mais avertissent qu'en cas d'activité illégale le service sera suspendu. L'utilisation de Tor soulève ici un problème juridique puisque l'hébergeur se retrouve en ligne de mire en cas d'activité illégale et doit faire systématiquement face aux démarches légales concernant un trafic incontrôlable. Il manque clairement dans la législation un mécanisme qui protégerait et déresponsabiliserait les hébergeurs et les utilisateurs. Mais pour que cela change, il faudrait qu'une volonté politique existe, or actuellement l'orientation est diamétralement opposée.

Enfin, nous avons le relais *bridge* qui forme un point d'accès particulier au réseau Tor. Normalement, pour vous connecter au réseau, vous utilisez le logiciel Tor sur votre machine. Vous êtes donc connecté directement au réseau et votre premier interlocuteur sera un relais interne.



Ceci pose problème dans le cas d'une censure puisqu'un État en mesure de contrôler les accès à Internet peut tout simplement interdire toutes les connexions aux relais connus. Pour contourner le problème, il existe des relais qui ne sont pas listés comme tels et fonctionnent donc sur des machines de particuliers, ce sont les relais *bridge*. Ils forment la seule façon d'accéder à Tor pour les habitants sous la coupe d'un régime totalitaire. Normalement, vous n'avez pas à vous soucier de cela... pour l'instant.

Le fonctionnement interne de Tor peut être imagé simplement. Partons du principe que vous avez une lettre à envoyer à Bernard, mais vous ne voulez ni qu'on la lise en chemin, ni que ceux qui la transporte sachent qu'elle vient de vous. Vous cherchez donc, par exemple, trois intermédiaires, Cécile, David et Émile, et vous commencez à préparer votre envoi. Vous rédigez votre lettre et y notez l'adresse de Bernard avant de la glisser dans une enveloppe à la cire. Appelons-la l'enveloppe bleue. Vous y marquez l'adresse d'Émile.

Vous glissez ensuite cette enveloppe dans une nouvelle, cachetée elle aussi, mais jaune. Vous y écrivez l'adresse de David. Vous recommencez avec une dernière enveloppe, verte, que vous marquez de l'adresse de Cécile. Enfin, vous postez le tout.

Cécile va recevoir l'enveloppe verte, l'ouvrir et y trouver la jaune. Elle sait qu'elle vient de vous, mais pas à qui le message est destiné ni ce qu'il contient. Elle poste donc l'enveloppe en question.

coromonadalix

DÉCOUVREZ NOS NOUVELLES OFFRES D'ABONNEMENTS !

PRO OU PARTICULIER = CONNECTEZ-VOUS SUR :

www.ed-diamond.com

LES COUPLAGES PAR SUPPORT :

VERSION PAPIER

Retrouvez votre magazine favori en papier dans votre boîte à lettres !

VERSION PDF

Envie de lire votre magazine sur votre tablette ou votre ordinateur ?

Sélectionnez votre offre dans la grille au verso et renvoyez ce document complet à l'adresse ci-dessous !

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

HACKABLE MAGAZINE

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

Vos remarques :

coromonadalix

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

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

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

VOICI TOUTES LES OFFRES COUPLÉES AVEC HACKABLE !

POUR LE PARTICULIER ET LE PROFESSIONNEL ...

Prix TTC en Euros / France Métropolitaine

CHOISISSEZ VOTRE OFFRE !

Prix en Euros / France Métropolitaine

Offre	SUPPORT ABONNEMENT	PAPIER Réf	PAPIER Tarif TTC	PAPIER + PDF (PDF 1 lecteur) Réf	PAPIER + PDF Tarif TTC	PAPIER + BASE DOCUMENTAIRE (1 connexion BD) Réf	PAPIER + BASE DOCUMENTAIRE Tarif TTC	PAPIER + PDF + BASE DOCUMENTAIRE (PDF 1 lecteur + 1 connexion BD) Réf	PAPIER + PDF + BASE DOCUMENTAIRE Tarif TTC
HK	6n° HK*	☐ HK1	39,-	☐ HK12	58,-				
	LES COUPLAGES « EMBARQUÉ »								
D	6n° HK* + 4n° OS	☐ D1	65,-	☐ D12	98,-	☐ D13	85,-*	☐ D123	118,-*
E	6n° HK* + 4n° OS + 6n° MISC	☐ E1	105,-	☐ E12	158,-	☐ E13	179,-*	☐ E123	232,-*
E+	6n° HK* + 4n° OS + 6n° MISC + 2n° HS	☐ E+1	119,-	☐ E+12	179,-	☐ E+13	193,-*	☐ E+123	253,-*
F	6n° HK* + 4n° OS + 11n° GLMF	☐ F1	125,-	☐ F12	188,-	☐ F13	229,-*	☐ F123	292,-*
F+	6n° HK* + 4n° OS + 11n° GLMF + 6n° HS	☐ F+1	183,-	☐ F+12	275,-	☐ F+13	287,-*	☐ F+123	379,-*
G	6n° HK* + 4n° OS + 6n° LP	☐ G1	100,-	☐ G12	150,-	☐ G13	164,- *	☐ G123	214,-*
G+	6n° HK* + 4n° OS + 6n° LP + 3n° HS	☐ G+1	129,-	☐ G+12	194,-	☐ G+13	193,-*	☐ G+123	258,-*
	LES COUPLAGES « GÉNÉRAUX »								
H	6n° HK* + 4n° OS + 6n° LP + 6n° MISC + 11n° GLMF	☐ H1	200,-	☐ H12	300,-	☐ H13	402,-*	☐ H123	499,-*
H+	6n° HK* + 4n° OS + 6n° LP + 3n° HS + 6n° MISC + 2n° HS + 11n° GLMF + 6n° HS	☐ H+1	301,-	☐ H+12	452,-	☐ H+13	493,-*	☐ H+123	639,-*

Les abréviations des offres sont les suivantes : LM = GNU/Linux Magazine France | HS = Hors-Série | LP = Linux Pratique | OS = Open Silicium | HC = Hackable

* HK : Attention : La base Documentaire de Hackable n'est pas incluse dans l'offre.

N'hésitez pas à consulter les détails des offres ci-dessus sur : www.ed-diamond.com !

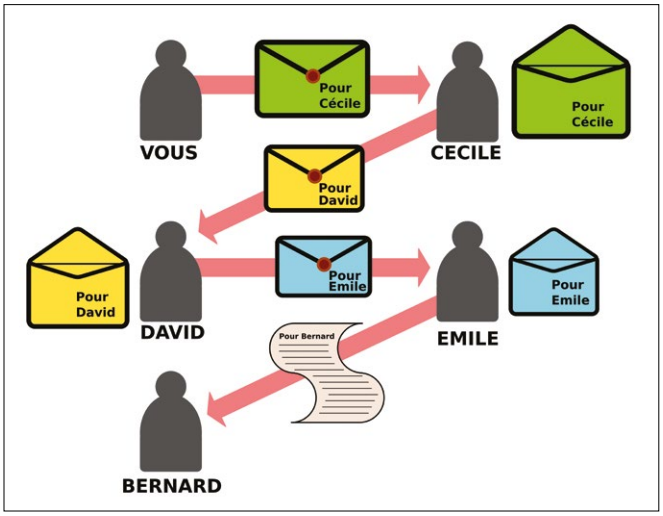
David reçoit alors l'enveloppe jaune qui semble parvenir de Cécile. Il y trouve l'enveloppe bleue à destination d'Émile. Il ne sait pas qu'elle vient de vous et ne sait pas où elle va. Pas plus qu'il ne peut savoir la teneur du message. Il poste l'enveloppe.

Émile réceptionne l'enveloppe bleue provenant de David. Lui non plus ne sait pas qui l'envoie réellement. En revanche, en l'ouvrant, il peut lire le message et constater qu'il est à destination de Bernard. Il lui apporte le message que Bernard peut lire, mais qui ne sait pas qu'il vient de vous et non d'Émile.

Je viens de décrire littéralement le fonctionnement de Tor. En lieu et place des enveloppes, nous avons le chiffrement et les messages passent non par des humains, mais par des machines relais. Le tout, avec bien plus d'intervenants, un chemin qui change en permanence et surtout une vitesse bien supérieure à celle de simples courriers.

Les caractéristiques des intermédiaires cependant sont les mêmes. Émile par exemple, comme le relais de sortie peut lire le message, s'il n'est pas chiffré d'une autre manière (HTTP et non HTTPS, par exemple). Cécile sait que les messages viennent de vous comme le premier relais utilisé, mais ne saura pas où va le message. Enfin, David ne sait rien du tout, comme un relais interne du réseau Tor, il ne fait que passer des enveloppes.

Vous comprenez sans doute mieux maintenant le pourquoi du « o » de Tor. Remplacez les enveloppes par les pelures d'oignon et tout devient limpide...



2. AJOUTONS TOR À NOTRE POINT D'ACCÈS

Ajouter Tor sur notre point d'accès Wifi ne demande pas beaucoup de travail, il suffit d'installer et configurer le logiciel. Nous commençons donc par l'installation à l'aide d'un simple `sudo apt-get install tor`. Ceci aura pour effet d'installer les éléments nécessaires, mais la configuration par défaut ne sera pas adaptée à nos besoins. Nous allons donc, comme précédemment modifier cette dernière en éditant le fichier `/etc/tor/torrc` (via `sudo nano` ou équivalent, le fichier appartient au `root`).

Le fichier exemple livré est entièrement composé de lignes en commentaire (`#`). Vous pouvez donc soit ajouter les lignes suivantes en début ou en fin de fichier ou, tout simplement effacer le contenu pour ne garder que notre configuration :

```
# fichier pour stocker les messages
Log notice file /var/log/tor/notices.log

# adresse virtuelle pour le partage
VirtualAddrNetwork 10.192.0.0/10

# on traite les .onion et .exit comme
# des machines dans le réseau Tor
AutomapHostsSuffixes .onion,.exit
AutomapHostsOnResolve 1

# on écoute sur le Wifi port 9040
TransListenAddress 192.168.74.1
TransPort 9040

# idem pour le DNS
DNSListenAddress 192.168.74.1
DNSPort 53
```

coromonadalix

Les lignes les plus importantes sont celles contenant **VirtualAddrNetwork**, **TransListenAddress** et **DNSListenAddress**. La première permet d'utiliser Tor dans un contexte « partagé ». Par défaut, le logiciel est conçu pour être utilisé localement, avec des logiciels qui fonctionnent sur le même système. Le serveur utilise donc des adresses locales par défaut (127.192.0.0/10). Comme le précise la documentation de Tor, lorsqu'on fournit le service à un réseau (ici, notre réseau 192.168.74.0 en Wifi), il faut spécifier des adresses virtuelles 10.192.0.0/10 ou 172.16.0.0/12.

Les autres lignes précisent l'adresse sur laquelle Tor attend les connexions pour les communications et les requêtes DNS. Sans entrer dans le détail, le DNS est le système qui permet à votre machine de trouver l'adresse d'une machine à partir de son nom (**hackable.fr** par exemple correspond à 213.186.33.3). Si ces demandes sont faites sans passer par Tor, mais directement sur Internet, il est facile de connaître les sites que vous visitez même si le contenu échangé est anonymisé. Il suffit d'observer les noms que vous recherchez pour vous suivre à la trace. Heureusement, il existe un système qui permet d'envoyer à Tor ces demandes, et il se chargera de trouver les réponses tout en protégeant ces informations. Pour cela, Tor fonctionne comme un serveur DNS et **DNSListenAddress**/**DNSPort** dans la configuration, précise l'adresse et le port sur lesquels attendre ces demandes.

Notre configuration est prête, il ne reste plus qu'à l'appliquer en redémarrant le service :

```
$ sudo service tor restart
[ ok ] Stopping tor daemon...done.
[ ok ] Starting tor daemon...done.
```

On pourra alors jeter un coup d'œil dans le journal d'activités **notices.log** pour s'assurer que tout fonctionne (avec **tail /var/log/tor/notices.log**) :

```
Tor 0.2.4.27 (git-412e3f7dc9c6c01a) opening log file.
Parsing GEOIP IPv4 file /usr/share/tor/geoip.
Parsing GEOIP IPv6 file /usr/share/tor/geoip6.
We now have enough directory information to build circuits.
Bootstrapped 80%: Connecting to the Tor network.
Bootstrapped 85%: Finishing handshake with first hop.
Bootstrapped 90%: Establishing a Tor circuit.
Tor has successfully opened a circuit. Looks like client
functionality is working.
Bootstrapped 100%: Done.
```

À ce stade, Tor fonctionne sur notre point d'accès et celui-ci accepte les connexions. Il ne reste plus qu'à passer les communications arrivant en Wifi dans le réseau Tor. Ceci fonctionne exactement comme la configuration de test que nous avons utilisé à la fin de l'article précédent. Nous devons simplement demandé au noyau de transférer les connexions, mais cette fois non entre le Wifi et l'Ethernet, mais entre le Wifi et Tor.

Comme cette configuration doit perdurer malgré les redémarrages de la Raspberry Pi, nous allons procéder de manière sensiblement différente. Nous ne toucherons pas à **/etc/network/interfaces** pour manuellement charger cette configuration, mais reposer plutôt sur les scripts d'un paquet existant.

coromonadalix

Commençons par définir nos règles avec **iptables**.

```
$ sudo iptables -t nat -A PREROUTING -i wlan0 \
-p tcp --dport 22 -j REDIRECT --to-ports 22

$ sudo iptables -t nat -A PREROUTING -i wlan0 \
-p udp --dport 53 -j REDIRECT --to-ports 53

$ sudo iptables -t nat -A PREROUTING -i wlan0 \
-p tcp --syn -j REDIRECT --to-ports 9040
```

L'idée ici est de tout simplement passer absolument tout le trafic du Wifi vers Tor. Ce faisant, nous risquons de créer un problème en rendant la Raspberry Pi inaccessible pour une connexion en ligne de commandes avec SSH. La première commande **iptables** indique donc que tout ce qui arrive par **wlan0** à destination du port 22 (celui de SSH) doit aller vers le système de la Pi. Si vous n'avez pas l'intention d'accéder à la Pi en SSH depuis le Wifi, il est recommandé d'ignorer cette règle (il est toujours possible d'utiliser SSH via Ethernet et donc le LAN). La seconde ligne fait de même, mais avec le trafic vers le port 53 (UDP). Ce sont les fameuses demandes DNS dont nous avons parlé précédemment. En effet, nous ne procédons pas à la résolution de nom au travers Tor, mais avec Tor lui-même.

Enfin, la dernière ligne traite tout ce qui n'a pas été filtré par les deux règles précédentes. Là, tout ce qui reste est renvoyé purement et simplement à Tor pour une communication anonyme et protégée. Il peut s'agir de web (HTTP) bien sûr, mais également de n'importe quelle autre forme de communication (protocole) comme l'accès aux mails, les sessions SSH ou encore la messagerie instantanée.

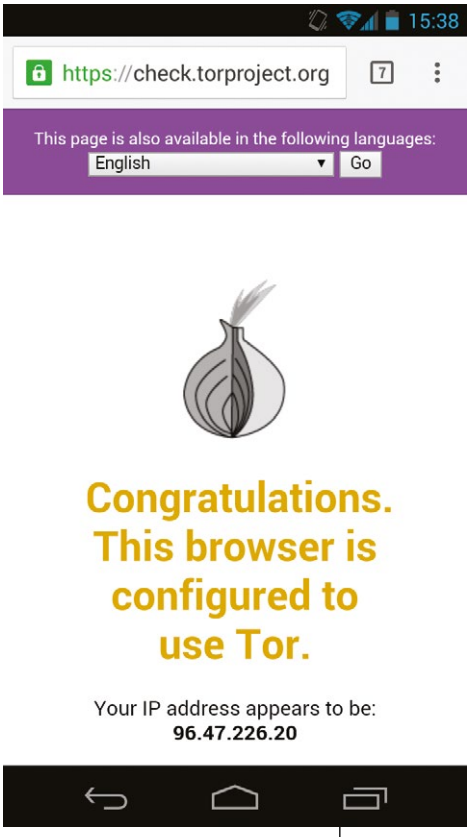
En tapant ces commandes, vous définissez des règles qui ne survivent pas à un redémarrage. Mais plutôt que de tout stocker puis recharger nous-mêmes, nous allons confier cela à des scripts. Il vous suffit d'installer un paquet avec **apt-get install iptables-persistent**. En validant cette commande, le système vous demandera s'il doit enregistrer les règles existantes, il vous suffira de répondre affirmativement et le tour est joué.

Les règles en cours seront enregistrées dans le fichier **/etc/iptables/rules.v4** et automatiquement chargées au prochain redémarrage. Vous n'avez rien d'autre à faire. Si vous souhaitez apporter des modifications, il vous faudra utiliser **iptables-save** en spécifiant le fichier en question.

3. VÉRIFICATION ET PRUDENCE

Pour vérifier que tout fonctionne et fonctionnera par la suite, il est recommandé de procéder à un redémarrage de la Raspberry Pi. C'est le seul moyen de vous assurer qu'il n'y aura pas de surprise et que tout est correctement configuré pour un démarrage automatique. Pour le test, rien de plus simple, après connexion au point d'accès, utilisez un navigateur et visitez la page dédiée par le projet Tor pour la vérification : **https://check.torproject.org**. Vous devriez voir alors apparaître un message de confirmation.

Nous avons testé cette configuration avec divers périphériques Wifi, des ordinateurs, des smartphones, des tablettes et même un iPhone. Dans le cas d'Android, nous avons pu vérifier



Pour vérifier que la communication passe bien par Tor, connectez-vous en Wifi sur votre nouvel AP et pointez simplement votre navigateur sur https://check.torproject.org. Si ce message apparaît, vous utilisez bel et bien Tor.

que tout fonctionne à merveille, aussi bien le surf sur le Web que le Play Store ou encore des applications comme Twitter.

L'intérêt de disposer d'un point d'accès à Tor est évident : il n'y a plus rien à configurer et rien à installer de la part des utilisateurs. Si, par exemple, vous proposez un point d'accès Internet dans votre association, votre restaurant ou votre entreprise, vous pouvez donc facilement fournir un service plus respectueux de la vie privée.

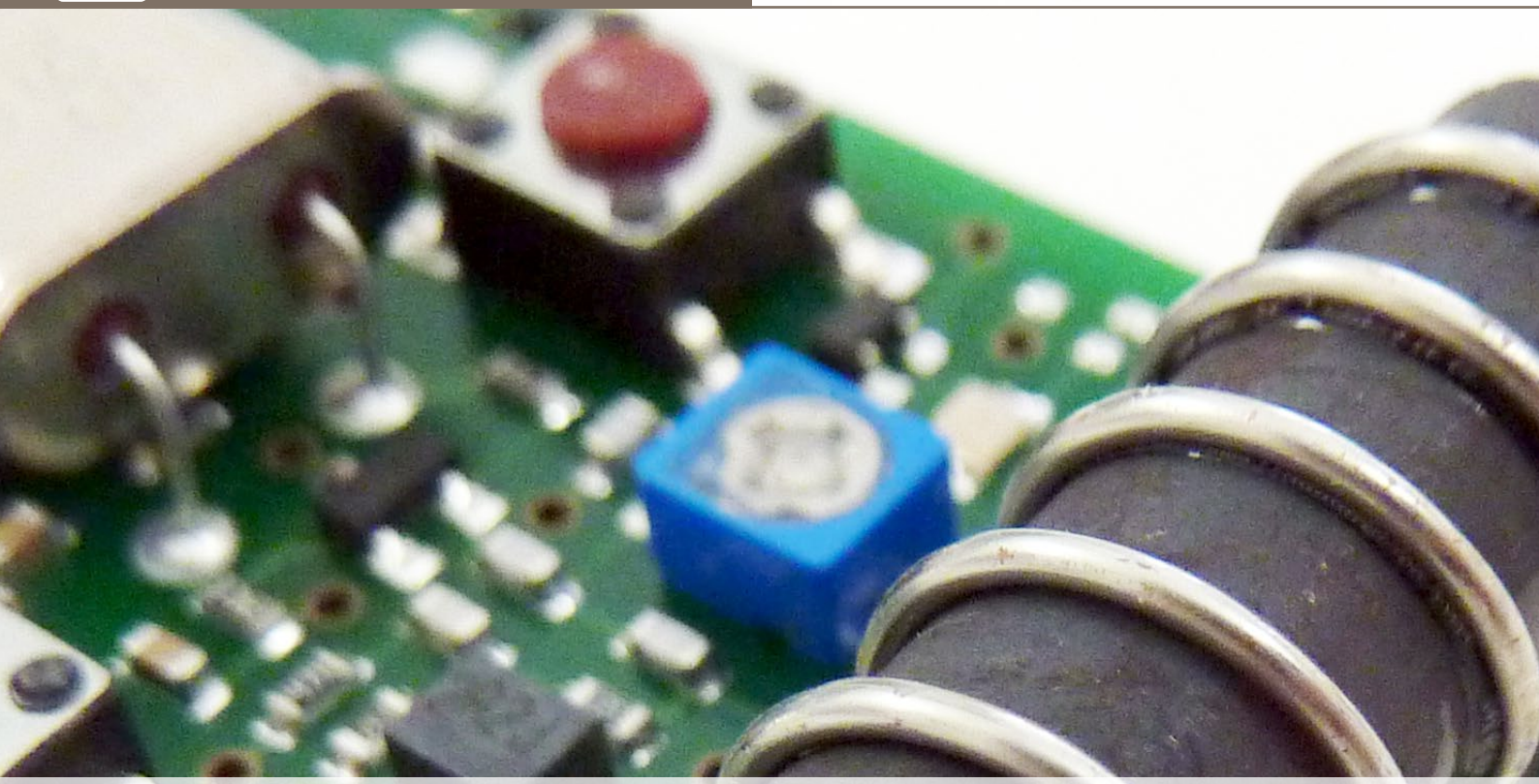
Notez que, bien entendu, cette solution n'est ni la seule ni la plus parfaite. Le projet Tor propose une solution à installer sur votre ordinateur sous la forme d'un navigateur dédié (basé sur Firefox). C'est le Tor Browser. Une solution pour smartphone est également disponible sous la forme de l'application Android Orbot Proxy. Celle-ci permet aux applications du smartphone de faire reposer leurs communications sur Tor.

L'installation d'un point d'accès Tor ne règle cependant pas tous les problèmes, car il y a bien des manières pour un navigateur de révéler votre identité. Le Tor Browser intègre des extensions Firefox bloquant par exemple Flash et Java, forçant l'utilisation de HTTPS ou encore permettant de finement régler l'exécution des JavaScript présents dans les pages web. Mais là encore ce n'est pas une solution absolue pour vous protéger. Si l'ordinateur ou le périphérique mobile utilisé est compromis par des spywares ou des malwares, Tor ne vous servira pas à grand-chose, car les données pourront être récupérées directement à la source (écran, clavier, fichiers, etc.). Remarquez également que dans le cas d'un point d'accès comme celui que nous venons de créer, la communication entre les clients Wifi et l'AP n'est pas protégée, car n'étant pas dans le réseau Tor (tout 192.168.74.0 ici).

Le principal rempart défendant votre intimité et votre vie privée est et sera toujours vous-même. Tous les dispositifs techniques du monde n'y pourront rien si votre comportement va à l'encontre de votre protection. Ce sont des règles de vie standards que beaucoup de personnes semblent oublier lorsqu'il s'agit d'Internet. Dans la vie, vous ne vous affichez pas publiquement dans la rue, vous ne devenez pas ami avec n'importe qui, vous ne confiez pas votre vie au premier venu, vous ne ramenez pas n'importe qui chez vous, vous n'exposez pas vos photos de vacances ou de votre dernière beuverie dans votre jardin... Sur le net c'est pareil.

Mark Zuckerberg et d'autres ont eu beau dire le contraire, ce n'est pas la vie privée qui disparaît, mais simplement la frontière entre la « vraie vie » et le net. Il suffit d'appliquer nos habitudes à ce nouveau paradigme. **DB**

coromonadalix



SDR : QUI PEUT ENTRER DANS MON GARAGE ?

Denis Bodor

La curiosité et la soif de connaissances peuvent mener à bien des choses. Le plus souvent, il s'agit simplement d'apprendre, mais parfois ce qui se trouve à la clé se résume successivement par les mots consternation, déception, colère et amertume. Voici l'histoire d'une petite escapade dans le monde de la radio logicielle (SDR) ou comment, grâce à une clé DVB-T, se rendre compte que n'importe qui peut ouvrir votre garage ou portail…

coromonadalix

Notre histoire commence avec le déménagement de la rédaction du magazine dans de nouveaux locaux. Cet événement déclencheur anodin en lui-même est la source même du présent article. Comme toute entreprise, les Éditions Diamond, éditeur de *Hackable*, se doivent de conserver une quantité non négligeable d'archives administratives en tous genres sur une période de plusieurs années. Ceci prend de la place et n'est que rarement consulté. Lorsqu'on déménage dans un centre-ville historique, inutile d'encombrer les bureaux de cartons et de classeurs poussiéreux. Comme beaucoup d'autres, le garage individuel en sous-sol reconverti en box de stockage est une excellente solution.

Pour entrer dans le sous-sol en question, l'agence immobilière nous a fourni une télécommande ouvrant le portail d'entrée. Là encore, c'est ici quelque chose de très courant. Une télécommande grise, moche et un peu usée nous est confiée. Elle est équipée de deux boutons et lorsqu'on appuie sur celui de gauche, le portail s'ouvre, donnant accès aux différents garages souterrains fermés individuellement à clé.

Cette aventure commence avec une pression malencontreuse sur le bouton de droite. On constate avec surprise que ceci ouvre le portail... se trouvant une dizaine de mètres plus loin. Voilà qui est intéressant ! Par simple pression d'un bouton sur **notre** télécommande, on ouvre un portail qui n'est pas le nôtre. Les personnes ayant accès à cette autre entrée, peuvent-ils aussi ouvrir notre portail en se trompant de bouton ? Quid de quelqu'un disposant d'un tel système dans une tout autre ville ? Peut-il aussi venir ouvrir des portails qui ne lui appartiennent pas ? Le nôtre peut-être ?

Notre « victime », une télécommande Cardin S466-TX2 comme il doit en exister des milliers en Europe. Celle-ci est un modèle spécialement acheté pour ne pas devoir « jouer » avec la télécommande utilisée par mes collègues.

Toutes ces questions méritent des réponses. Non que des archives, dont la lecture est somme toute soporifique, puissent être considérées comme un bien d'une extrême valeur, l'objet premier d'un garage est de mettre un véhicule à l'abri des intempéries... et des voleurs. Tous les parkings souterrains n'offrent pas de garages individuels fermés à clé et la raison première de mettre son véhicule dans un tel emplacement est bien de le protéger ou d'en protéger le contenu (GPS, radio, etc.).

1. UN PEU DE RECHERCHE

Après quelques minutes de recherche sur le Web, le type et le modèle de télécommande en ma possession est relativement clair : c'est un produit italien fabriqué par *Cardin Elettronica Spa* sous le nom S466-TX2.

Sur le site du fabricant, à la section concernant le modèle S46 **[1]** on peut lire « *La vaste gamme de modèles permet de répondre à tous les cas de figure en garantissant à l'utilisateur une sécurité absolue. […]*

coromonadalix

Gros plan sur l'antenne de l'émetteur. Dans 80% des cas, lorsque la télécommande ne fonctionne toujours pas après changement de la pile, c'est là qu'il faut regarder. Une simple soudure peut vous faire économiser 25 euros...

Elle trouve sa meilleure application dans la commande de systèmes de fermetures automatisées et dans toutes les installations où est requis un actionnement à distance (sans fil) protégé par un code secret haute fiabilité. […] Un système complet et sûr réalisé avec la rigueur et la qualité qui distinguent tous les produits CARDIN ». Selon le constructeur donc, nous avons affaire à un système « sûr », car utilisant un « code secret haute fiabilité » et offrant une « sécurité absolue ».

Le modèle exact est référencé TRQ46620C.EUR. Exactement celui décrit sur le site. Il existe une déclinaison 4 fonctions, référence TRQ466400, mais que je n'ai pas eu le loisir de tester.

Ce modèle semble extrêmement courant et le Web est truffé de messages d'utilisateurs cherchant à remplacer leur télécommande endommagée par un modèle neuf. Qui dit demande importante, dit forcément offres nombreuses. On retrouve ainsi des dizaines de sites proposant des télécommandes de remplacement aussi bien sous la forme de petites boutiques en ligne que d'offres sur les sites d'enchères.

Comment donc un équipement offrant une « sécurité absolue » via un « code secret haute fiabilité » peut-il être si facilement remplaçable ? C'est relativement simple, sous la petite trappe permettant d'accéder à la pile se trouve également une série de petits interrupteurs numérotés de 1 à 9. Chacun d'eux peut prendre une position « - », « o » ou « + ». Une paire supplémentaire d'interrupteurs permet de définir un « canal » supplémentaire affecté au second bouton de la télécommande. Enfin, un cavalier permet d'assurer une compatibilité avec deux séries de récepteurs (« C » et non « C »).

Après quelques minutes de recherche supplémentaires, on finit par trouver un PDF de deux pages (fascicule ZVL111.06 série S46-S46C du 30/01/2008). Celui-ci décrit l'utilisation et la configuration des émetteurs 2 et 4 fonctions ainsi que des récepteurs. Pour remplacer une télécommande défectueuse ou cassée, il suffit d'en acquérir une nouvelle du même modèle, de placer les interrupteurs exactement dans la même position et le tour est joué. En d'autres termes, si quelqu'un connaît la position de vos interrupteurs, votre code, il peut configurer sa télécommande pour ouvrir votre garage/portail.

Instinctivement, la première question qui nous vient à l'esprit est donc : quelqu'un peut-il trouver notre code ? Une réponse rapide est « oui, il suffit qu'il ouvre la trappe des piles de votre télécommande et mémorise la position des interrupteurs », mais ceci suppose qu'il ait accès à votre télécommande. Cependant, c'est à mon sens déjà une faille, en particulier pour un produit qui revendique offrir une « sécurité absolue ».

La notice technique, elle, est bien plus optimiste et précise : « *Considéré que la programmation peut être effectuée plusieurs*

coromonadalix

fois après avoir effectué l'installation, l'inviolabilité du code usager est assurée ». Que doit-on comprendre ? Comme on peut changer le code alors c'est forcément inviolable ? La description sur le site paraphrase : « *La programmation peut être faite maintes fois, même après la pose, ce qui garantit l'inviolabilité du code usager* ». En gros, comme on peut changer le code toutes les 5 minutes, c'est sécurisé ? Quelqu'un qui a trouvé votre code vient de vandaliser votre voiture, il vous suffit de le changer ? Une telle affirmation, en dehors du fait d'aller à l'encontre du bon sens, se doit d'être vérifiée. Comment la sécurité du code est-elle assurée ?

Techniquement, la notice ainsi que le site web du fabricant nous apportent un grand nombre d'indications nous permettant de vérifier ce point :

- fréquence utilisée (pour l'Europe) : 27,195 Mhz ;
- modulation AM/ASK, ce qui correspond à une modulation d'amplitude (la fréquence ne change pas) et plus précisément à une modulation par déplacement d'amplitude ou ASK (de l'anglais « *Amplitude-Shift Keying* »). C'est tout bonnement l'une des modulations les plus simples ;
- 19383 codes programmables par dip-switch (micro-interrupteurs) ternaires à 9 voies. En clair, 9 interrupteurs pouvant prendre une position sur trois, nous donne 3 puissance 9 combinaisons possibles, soit 19683 combinaisons.

La question est maintenant simple : sans regarder la position des interrupteurs dans le boîtier, peut-on tout simplement « lire » les données transitant dans les ondes et en déduire le code à utiliser ? Si tel est le cas, quiconque pouvant démoduler le signal sera en mesure de programmer le code correspondant dans sa télécommande et déclencher l'ouverture comme vous le feriez vous-même.

2. RTL-SDR ET DONGLE DVB-T À LA RESCOUSSE

Dans le numéro 2 de *Hackable*, nous avons appris comment un simple récepteur TNT DVB-T USB était en mesure de recevoir bien plus que des émissions TV lobotomisantes. Pour rappel, ce type de périphérique se contente de traduire un signal radio en une suite d'informations numériques. À la charge des logiciels sur PC ou Mac de démoduler et décoder ces informations, initialement pour afficher à l'utilisateur un flux audio/vidéo.

Ce traitement minimal par le matériel et surtout le fait que le logiciel sur l'ordinateur se charge de tout le travail de traitement du signal porte un nom, c'est la radio logicielle ou SDR pour l'anglais *Software Defined Radio*.

Une petite trappe permet à la fois de changer la pile 12V et d'accéder aux micro-interrupteurs configurant le code de la télécommande. Notez que quelqu'un avec une bonne mémoire se contentera de quelques secondes d'inattention de votre part pour lire et mémoriser le code... Mais ajouter une vis ça coûte cher et c'est compliqué pour le client...

coromonadalix

Le principal intérêt de cette méthode tient dans le fait que pour changer le type de démodulation et de décodage qu'on souhaite utiliser, il suffit d'utiliser ou d'écrire un logiciel. C'est ainsi qu'un simple périphérique USB se transforme en récepteur universel capable de prendre en charge la radio FM, la TNT, mais aussi la radiomessagerie, les protocoles de surveillance coopératifs pour le contrôle du trafic aérien, les données météorologiques envoyées par les satellites ou même les communications de la station spatiale internationale (ISS).

Précisons ici que tous les récepteurs DVB-T USB ne fonctionnent pas de la sorte, mais, fort heureusement, c'est un choix technique économique pour les fabricants (moins d'électronique) et donc courant pour les récepteurs à quelques 20 ou 25 euros. Ces périphériques se composent de deux éléments, une puce de conversion analogique/numérique RTL2832U (d'où le nom du support, RTL-SDR) et d'un tuner permettant la réception. Les tuners actuellement supportés pour les applications SDR sont :

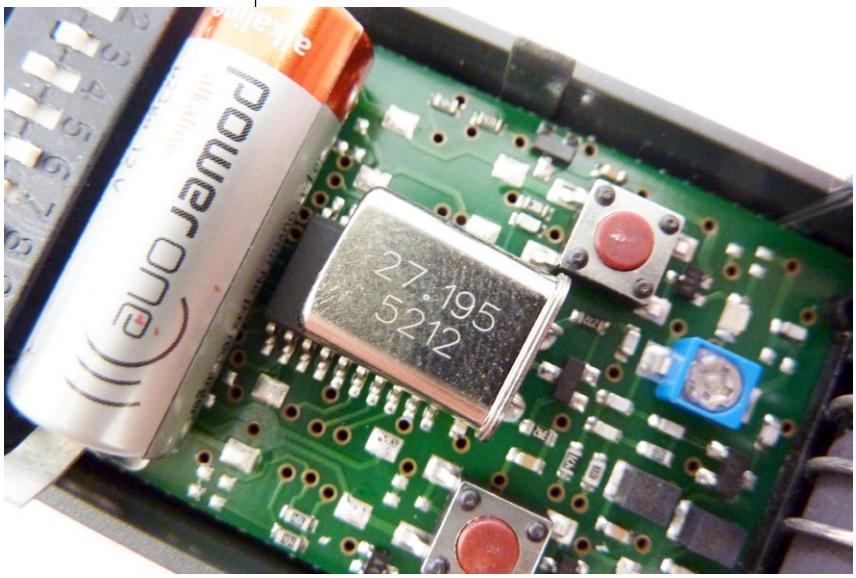
- Elonics E4000 : réception de 52 à 2200 MHz (avec un trou de 1100 MHz à 1250 MHz) ;
- Rafael Micro R820T : réception de 24 à 1766 MHz ;
- Rafael Micro R828D : réception de 24 à 1766 MHz ;
- Fitipower FC0013 : réception de 22 à 1100 MHz ;
- Fitipower FC0012 : réception de 22 à 948.6 MHz ;
- FCI FC2580 146 à 308 MHz et 438 à 924 MHz.

Les périphériques actuellement les plus courants sont équipés de R820T de Rafael Micro. Si vous cherchez un tel récepteur USB, et grâce à la popularité de leur utilisation en SDR, optez pour un produit dont la description parle tout simplement de SDR ou RTL-SDR. Une source sûre est **nooelec.com** (boutique en ligne, Amazon ou eBay) ou encore **passion-radio.com** en France (je n'ai pas d'actions, j'ai simplement acheté un autre équipement chez eux, un HackRF One, et ai été très satisfait de la communication et du service). Il en existe sûrement d'autres et vous pouvez également tenter votre chance sur eBay auprès de vendeurs chinois.

Notez que le périphérique qui historiquement a déclenché cette nouvelle fièvre de la SDR (voir *Hackable n°2*) était équipé d'un tuner Elonics E4000 qui n'est **PAS** adapté dans le cas présent. La fréquence utilisée par la télécommande, 27,195 Mhz, est en dessous de la limite de 52 Mhz de ce tuner.

Dans cet article, nous allons partir du principe que vous ayez une petite connaissance concernant ces périphériques USB ou, du moins, que vous en avez déjà un d'installé. Il est donc préférable que vous ayez lu ou survolé le dossier présenté dans le numéro 2 du magazine.

En l'absence de documentation détaillée ou de marquage sur le boîtier, il est toujours possible, avant de se lancer dans une recherche par tâtonnage, de simplement lire la fréquence du quartz. Il est fort probable ici que la version « hors-CE » du matériel se résume, pour le constructeur, à l'utilisation d'un quartz 29,875 MHz en lieu et place du 27,195 Mhz.



Nous allons cependant utiliser un logiciel différent de ceux que nous avons déjà étudiés. Ceci est un avantage pour les lecteurs découvrant le domaine de la SDR avec un périphérique RTL-SDR, car ce logiciel est le très connu et puissant GNU Radio et son interface graphique GNU Radio Companion.

3. GNU RADIO ET GNU RADIO COMPANION

GNU Radio est une boite à outils permettant le traitement de signal et la création de radio logicielle. À la base, il s'agit d'un kit de développement pour programmeur Python et C++. Il propose des blocs de traitement pour la plupart des opérations liées à la SDR (mathématiques, démodulation, modulation, échantillonnage, analyse, etc.). On peut donc l'utiliser pour ses programmes, mais il offre également une solution pour les non-programmeurs. En effet, grâce à GNU Radio Companion, il est possible d'utiliser un environnement graphique, disposer les blocs de traitement et les lier entre eux visuellement, pour composer un graphe représentant votre création.

L'interface se présente comme un tableau blanc sur lequel on dispose des blocs possédant des entrée(s), des sortie(s) ou les deux. Il suffit alors de connecter entre eux ces blocs pour définir l'enchaînement des opérations. Sur la droite de la fenêtre, une liste des blocs disponibles est présentée, classée par type et fonction. Pour utiliser un bloc, il suffit de le prendre dans la liste et le glisser dans la zone de travail. Au bas de la fenêtre se

GNU RADIO LIVE

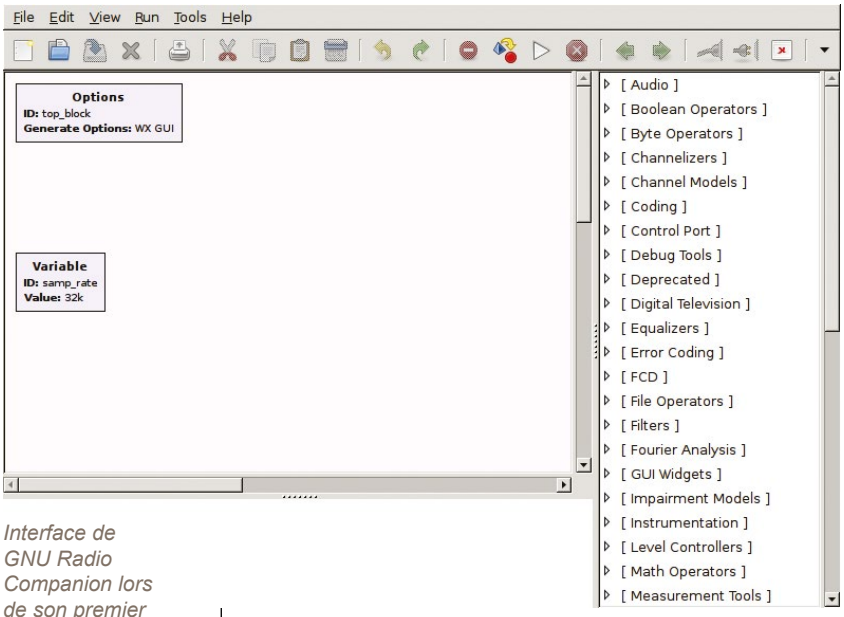
Si vous venez juste de vous procurer une clé USB DVB-T compatible RTL-SDR ou que vous rencontrez des difficultés pour configurer votre système (GNU/Linux, Windows ou Mac OS X) afin qu'il prenne en charge le périphérique, il existe une solution clé en main : la distribution Live d'un système GNU/Linux. Un système dit « live » permet le démarrage un ordinateur sur un DVD ou une clé USB et l'utilisation du système sans rien installer.

Il existe quantité de distributions GNU/Linux (et d'autres systèmes) en version Live CD ou Live USB. Il y a des systèmes permettant de découvrir GNU/Linux et d'autres spécialisés dans certains domaines (sécurité, musique, mathématique, etc.). Bien entendu, ce qui nous intéresse ici c'est la SDR et la distribution à utiliser est *GNU Radio Live SDR Environment*.

Pour la télécharger, il vous suffira de pointer votre navigateur sur https://gnuradio.org/redmine/projects/gnuradio/wiki/GNURadioLiveDVD et de télécharger l'image du DVD (attention 2 Go !) soit directement, soit avec un client BitTorrent (plus efficace).

Ensuite, deux solutions s'offrent à vous, soit vous gravez l'image du DVD sur un DVD-R avec le logiciel de votre choix pour ensuite démarrer dessus, soit vous utilisez une application appelée UNetbootin (http://unetbootin.sourceforge.net/). Celle-ci, disponible pour Windows, GNU/Linux ou Mac OS X permet d'utiliser une image de CD ou de DVD pour créer une clé USB de démarrage. Lancez simplement UNetbootin, sélectionnez le fichier à utiliser (image ISO précédemment téléchargée) et la clé USB qui servira de support (attention, celle-ci sera totalement effacée), puis lancez la procédure. Une fois l'opération terminée vous n'aurez plus qu'à démarrer votre PC ou votre Mac sur la clé USB et vous aurez accès à un système Ubuntu GNU/Linux avec GNU Radio Companion et les pilotes pour RTL-SDR directement utilisables.

Graver un DVD est sans doute plus facile, mais cela ne vous permet pas d'enregistrer ce que vous faites avec la distribution Live. Le fait d'utiliser une clé USB vous permet de faire usage du système, sans rien installer, tout en disposant du confort d'utilisation normal (enregistrement, configuration qui perdure entre les redémarrages, personnalisation, etc.).



Interface de GNU Radio Companion lors de son premier lancement.

trouve la console où apparaissent les messages liés à la génération du « programme » et à son exécution. La logique est la suivante : on compose son programme visuellement, celui-ci est traduit en code puis exécuté s'il ne contient pas d'erreur.

Chaque bloc possède un certain nombre d'éléments de configuration ou propriétés qui permettent d'ajuster son comportement. Ces propriétés sont accessibles à l'aide d'un simple double-clic.

On distingue grossièrement quatre types de blocs :

- les blocs *Source* qui fournissent un signal. Il peut s'agir d'un générateur purement logiciel, d'un bloc de lecture de fichier ou, plus souvent, d'un bloc permettant la réception d'un signal à partir d'un matériel spécifique comme une carte son ou un récepteur comme une clé DVB-T. Le bloc source `osmocom` permet l'utilisation de nombreux récepteurs SDR comme ceux supportés par RTL-SDR, mais également le FUNcube, AirSpy, Ettus USRP ou encore le HackRF. Ce bloc permet également de prendre en charge un récepteur RTL-SDR au travers d'un réseau.
- les blocs *Sink* qui dispensent une information. Ce sont des « sorties » qui peuvent être, là encore, des accès à du matériel (carte son, émetteur, etc.), des fichiers ou encore des éléments d'interface graphique permettant de visualiser des informations (oscilloscope, analyseur de spectre, graphique *waterfall*, etc.).
- les blocs de traitement qui sont destinés à manipuler ou traiter le signal ou encore à en extraire des éléments spécifiques. Ces blocs permettent également de moduler et démoduler des signaux.
- les blocs « statiques » permettant de stocker des paramètres et des variables. Ceux-ci permettant d'associer un nom à une valeur et d'utiliser ce nom en lieu et place de la valeur dans d'autres blocs.

Lorsque vous lancez GNU Radio Companion pour la première fois ou créez un nouveau graphe, le logiciel ajoute pour vos deux blocs automatiquement. Le premier, `top_block` représente la fenêtre de l'interface graphique de votre programme (son ID est aussi le nom du programme Python qui sera généré). Tous les éléments (*widget*) que vous utiliserez dans votre graphe/programme trouveront place dans cette fenêtre. Le second bloc automatiquement créé est une variable `sample_rate` dont la valeur est fixée à 32k. Cette variable vous permet d'utiliser `sample_rate` comme référence à la valeur définie. En double-cliquant sur ce bloc, vous en affichez les propriétés. On trouve alors son ID qui est le nom de la variable (`sample_rate`) et sa valeur (*Value*) par défaut à 32000.

Il est important de remarquer ici que la valeur d'une variable peut être directement indiquée, mais qu'on peut également spécifier un calcul, éventuellement basé sur d'autres variables. Ainsi, si vous glissez depuis la liste des blocs une autre variable et éditez ses

coromonadalix

propriétés, vous remarquerez qu'elle a été automatiquement appelée `variable_0` et que sa valeur est 0. Vous pouvez changer son nom et spécifier comme valeur `samp_rate/2`. Dès validation, le bloc affichera « 16k » en guise de valeur, Cette variable sera systématiquement la moitié de celle de `samp_rate`.

Un autre point important concerne la façon de spécifier une valeur. Vous l'avez sans doute remarqué, notre bloc `samp_rate` affiche 32k, mais sa valeur dans la propriété est 32000. La syntaxe utilisée pour spécifier une valeur est celle du langage Python. Il ne vous est pas possible d'indiquer « 32k ». Si vous tentez de le faire, le titre du bloc deviendra rouge, ainsi que la valeur affichée, vous signalant une erreur. En accédant aux propriétés, là encore, « Value » sera rouge, indiquant un problème et au bas de la fenêtre une erreur sera signalée : *Value "32k" cannot be evaluated*. Vous pouvez cependant spécifier des valeurs importantes en utilisant une notation scientifique (x fois 10 puissance n). Ainsi plutôt que d'utiliser `32k`, vous pouvez spécifier `32e3` comme 32 fois 10 puissance 3. Pour les milliers, les avantages ne sont pas forcément évidents, mais si l'on parle par exemple d'une fréquence, il est bien plus intelligible et facile d'écrire `27,195e6` que `27195000`. Dans le bloc, GNU Radio Companion se chargera, comme un grand, d'afficher « 27,195M ».

Les blocs placés peuvent être supprimés facilement, il suffit de sélectionner un ou plusieurs blocs en cliquant dessus une fois, en utilisant la touche Ctrl pour en cliquer plusieurs ou en maintenant le bouton gauche de la souris pour faire une sélection. Une fois le ou les blocs sélectionnés, on utilisera simplement la touche Suppr pour les effacer. Il est également possible de copier/coller ou couper/coller des blocs. Les ID des copies seront automatiquement renommés en ajoutant un chiffre à la fin de l'ID original. Enfin, il est possible de désactiver des blocs afin qu'ils ne soient plus pris en compte dans le graphe tout en restant disponibles pour plus tard. Il suffit de les sélectionner et d'utiliser la touche « D » pour désactiver (*disable*) et « E » pour activer (*enable*). Le ou les blocs en question deviennent gris une fois désactivés.

4. UN PREMIER PAS AVEC GNU RADIO COMPANION

Nous n'allons pas ici entrer dans des explications et des concepts trop avancés, mais nous devons toutefois disposer de quelques bases solides afin de donner un sens à nos futures manipulations. Le domaine de la SDR et du traitement de signal est vaste et tantôt complexe. C'est le royaume où les mathématiques et la trigonométrie sont omniprésentes. On remarquera avec amusement que selon la voie professionnelle qu'on aura choisie, il est courant de se dire que certains concepts mathématiques ne nous serviront jamais... jusqu'au jour où on aborde un domaine particulier qui nous fait regretter de ne pas avoir été bien plus attentif dans certains cours. On peut ainsi passer toute une vie sans avoir à réutiliser des nombres complexes ou imaginaires pour subitement se dire que la racine carrée de -1 n'est finalement pas qu'un outil de torture pour écolier.

Nous n'irons pas si loin ici, mais sachez que si vous voulez pousser plus loin votre découverte de la SDR, viendra forcément le moment où vous devrez revoir des concepts et des méthodes apprises il y a fort longtemps pour certains. Que ce soit clair, vous n'y couperez pas, à moins bien entendu de simplement survoler le sujet et vous en tenir au rôle d'acteur en chargeant des graphes tout faits sans les comprendre et sans pouvoir les modifier.

Nous l'avons vu précédemment, GNU Radio Companion crée pour vous une variable `samp_rate`. Il s'agit de la fréquence d'échantillonnage ou en d'autres termes le nombre de fois par seconde où une valeur est mesurée.

coromonadalix

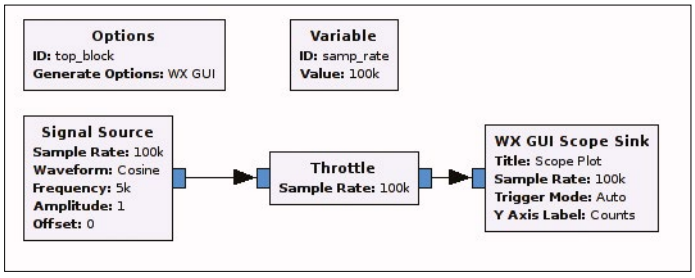
Votre ordinateur n'est pas analogique, pour mesurer un tel phénomène il doit, à répétition, obtenir une valeur numérique discrète. C'est précisément le travail du convertisseur analogique/numérique présent dans le récepteur DVB-T, mais pour commencer nous allons utiliser une source différente, plus simple.

Dans GNU Radio Companion, recherchez et placez un bloc « *Signal Source* » (il se trouve dans « *Waveform generator* »), puis double-cliquez pour changer ses propriétés. Par défaut, la fréquence d'échantillonnage est réglée sur `samp_rate`, le type sur *Cosine* (cosinus) et avec une amplitude de 1. Spécifiez une fréquence (*Frequency*) à `5e3`, soit 5000 Hz (ou 5 Khz).

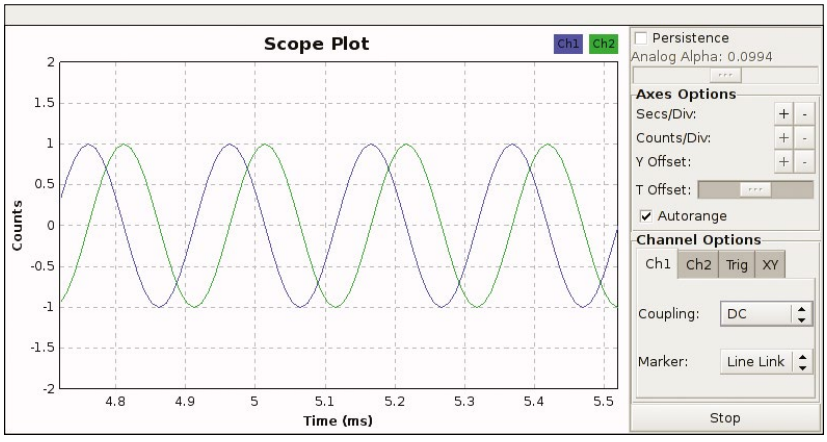
Ajoutez ensuite un bloc « *Throttle* » (dans « *Misc* »). Ce bloc est indispensable lorsqu'on n'utilise pas une source physique comme un récepteur. Il permet de limiter le flux de données dans le graphe. S'il est absent, GNU Radio va essayer de traiter les données le plus rapidement possible, mais comme rien ne limite cet effort, il va se mettre à utiliser votre ou vos processeurs intégralement (100 %), ce qui peut conduire à de gros problèmes, voire un plantage du système. Le bloc « *Throttle* » utilise, lui aussi, automatiquement la variable `samp_rate`, mais cette fois avec une signification particulière : limiter le traitement des données à cette fréquence d'échantillonnage pour ne pas saturer la machine.

Enfin, ajoutez un bloc « *WX GUI Scope Sink* » (dans « *Instrumentation* » et « *WX* »). Il s'agit d'un *widget*, un élément de l'interface graphique présentant l'équivalent d'un oscilloscope rudimentaire. Son travail est de présenter un signal dans le temps avec en abscisse le temps et en ordonnée l'amplitude du signal, c'est-à-dire la valeur mesurée à un instant donné.

Connectez à présent les blocs entre eux. Cliquez sur l'excroissance bleue sur la droite du bloc « *Signal Source* », puis sur celle à gauche du « *Throttle* ». Procédez de même entre le « *Throttle* » et le troisième bloc pour obtenir quelque chose comme ceci :



Changez ensuite les propriétés de la variable `samp_rate` pour spécifier une valeur plus importante : 100000. Enfin, générez le programme et exécutez-le en cliquant sur le bouton *Lecture* dans la barre d'outils ou en appuyant simplement sur la touche F6. Ce que vous devez voir apparaître sur l'écran ressemble à ceci :

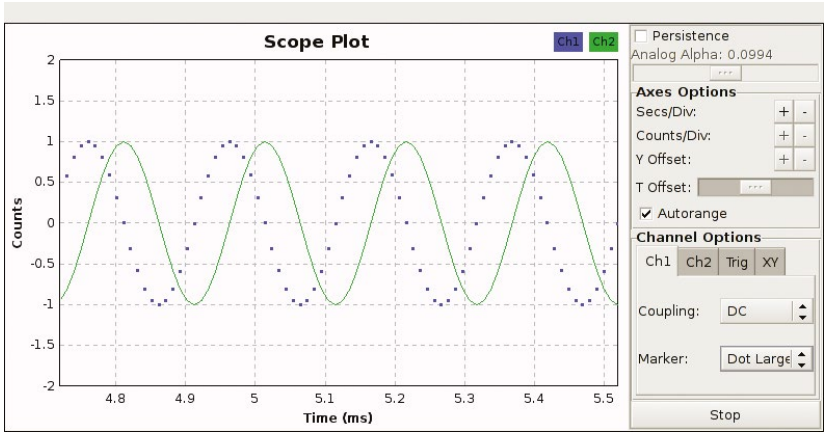


Nous voulions un cosinus et le voici ! À ce stade, normalement, vous devez vous poser un tas de questions...

Si mon ordinateur est incapable de gérer des données analogiques, pourquoi ai-je un beau cosinus ? En réalité, cette fonction ou ce signal n'est pas aussi parfait que vous

coromonadalix

pouvez le penser. En bas à droite de la fenêtre se trouvent quelques onglets. Dans le premier, vous disposez d'un menu déroulant « *Marker* » permettant de spécifier la façon d'afficher les données. Changez-le de « *Line Link* » en « *Dot Large* » :



Voici les vraies données. Nous avons un signal périodique à 5000 hz avec une fréquence d'échantillonnage de 100000 mesures par seconde. Si vous comptez les points d'une période (crête à crête par exemple), vous vous rendrez compte qu'ils sont au nombre de 20 (comme 100000/5000) et qu'ils s'étalent sur une période de 0,2 millisecondes (1/5000 de seconde).

5. QUELQUES NOTIONS DE BASE, JUSTE UN PEU…

L'autre question qui doit vous venir à l'esprit après cet essai est « pourquoi nous avons deux courbes sur l'écran ? ». La réponse est littéralement complexe, mais pas vraiment compliquée. Ceci est en lien direct avec la façon dont sont représentées les données du signal. Une façon « naïve » de manipuler un signal numérisé consiste à mesurer l'amplitude à un instant *t* et de la stocker comme une valeur à virgule flottante. Ceci cependant, même si cela semble intuitivement une bonne solution n'est pas adapté à la SDR, ni même au domaine du traitement numérique de signaux (ou DSP en anglais pour *Digital Signal Processing*).

Nous allons essayer de poser clairement les choses, et ce de la manière la plus simple et concise possible (au risque de faire hurler le lecteur versé en mathématiques ou en traitement de signaux, désolé). Avant tout, lorsqu'on parle de signaux, de DSP ou de SDR, on parle de quelque chose de bien précis qui tient en quelques affirmations :

- ce qui nous intéresse est un signal sinusoïdal périodique ;
- en radio logicielle, nous modulons ce signal pour transporter des données ;
- une modulation consiste à modifier un signal sinusoïdal (en amplitude, en fréquence ou en phase).

Oubliez pour l'instant l'échantillonnage, la mesure et la numérisation. On parle de sinus et de cosinus, et l'on s'acharne à torturer ces fonctions pour leur faire transporter des informations (son, données, bits, images, etc.). Pour rappel, une onde sinusoïdale se résume à :

```
V(t) = A * sin(2*pi*f*t + p)
```

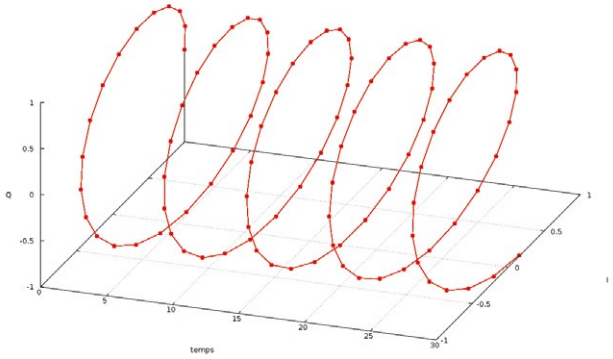
V est la tension à un instant *t* tel que l'amplitude maximum *A* multiplié par la fonction sinus. Pour « torturer » le signal on peut jouer sur :

- *A* pour une modulation d'amplitude ;
- *f* pour une modulation de fréquence ;
- *p* pour une modulation de phase.

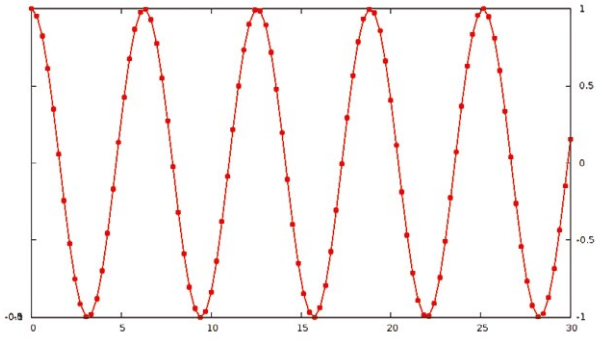
Vous avez appris en classe qu'il est possible de représenter un tel signal en deux dimensions avec à l'horizontal le temps (x) et à la verticale le résultat d'une fonction

sin(x). La SDR ne fonctionne pas comme cela, car ce n'est ni pratique en termes de modulation/démodulation, ni efficace pour nos ordinateurs.

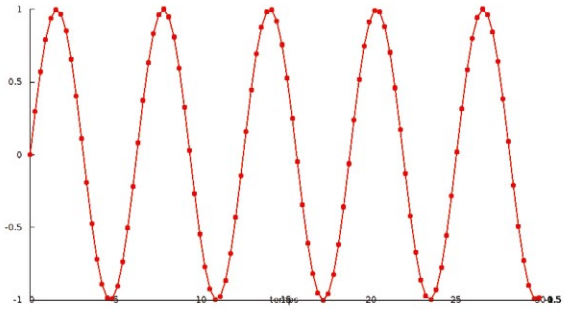
Pour comprendre la représentation d'un signal en SDR (entre autres), nous devons nous rappeler nos cours de trigonométrie. Mais avant cela, le plus simple est de tout simplement l'illustrer. Voici donc un signal tel que le conçoit un récepteur SDR et GNU Radio :



Oui, il s'agit d'un modèle 3D et notre signal apparaît comme une spirale. Chaque point sur cette représentation est un échantillon, les lignes reliant les points ne sont là que pour aider à la visualisation. Si l'on regarde le signal de côté, avec le temps devenant l'axe horizontal, nous voyons ceci :

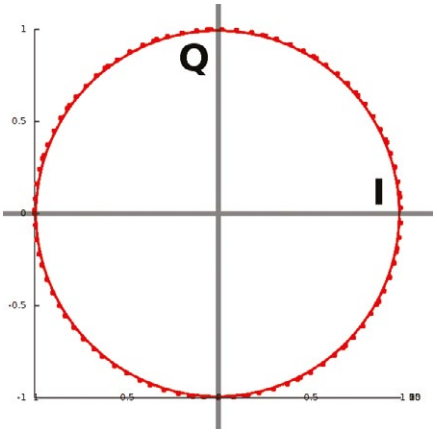


C'est une sinusoïdale et la représentation est maintenant en 2D. Ceci semble plus naturel puisque c'est ainsi que nous avons l'habitude de voir un tel signal (sur un oscilloscope par exemple). Cependant, nous n'avons plus aucune information « en profondeur ». Si à présent nous regardons le modèle 3D par le haut, nous voyons ceci :



Le résultat est sensiblement différent même s'il reste similaire. Notre signal semble décalé, pourtant il s'agit des mêmes données. En y réfléchissant un court instant, c'est tout à fait normal puisque nous avons « tourné » autour de la spirale et un même point apparaît à un autre endroit relativement aux repères. Nous avons donc tourné notre spirale de 90° (un quart de tour) afin d'en observer d'autres caractéristiques.

Si maintenant nous nous plaçons de manière à ce que l'axe du temps avance vers nous, nous verrons quelque chose comme ceci :



Nous voyons notre spirale sous la forme d'un cercle, avec chaque échantillon trouvant une position bien particulière dans ce nouveau graphique très particulier. Ici, notre signal est constant, il ne change ni en amplitude, ni en fréquence et tous les points apparaissent sur un même cercle. Chaque échantillon est, en SDR, un point sur ce graphique. Notre signal est découpé en autant de tranches que de points.

Mais cette représentation est très particulière, car il ne s'agit pas d'un simple graphique, mais d'un plan

coromonadalix

complexe ou plan d'Argand où chaque point est un nombre complexe. On trouve en abscisse la partie réelle du nombre complexe et en ordonnée la partie imaginaire (i = unité imaginaire = racine carrée de -1). N'importe quel point sur ce plan peut donc être défini à l'aide d'un simple nombre complexe. Mais ce n'est pas tout, car en travaillant de cette manière, il est possible de profiter de toutes les propriétés étranges et déroutantes des nombres complexes, ainsi que des relations trigonométriques qu'il est relativement simple de visualiser.

En SDR, les axes de ce plan complexe ont un nom particulier. L'axe des réels est dit « en phase » ou « *In-phase* » en anglais et est généralement résumé en « I » (à ne pas confondre avec le « i » minuscule qui est l'unité imaginaire telle que $i^2=-1$). L'axe des nombres imaginaires quant à lui est dit « en quadrature » et est noté « Q ». Pour décrire ce type de représentation d'échantillons sous forme de nombres complexes, on parle donc de données IQ, de signaux IQ ou de signal en quadrature.

Ainsi, si nous revenons à la question de départ portant sur le fait d'avoir deux signaux dans notre widget scope, la réponse est finalement simple : nous avons en bleu la partie réelle des échantillons et en vert la partie imaginaire. Celle-ci est décalée de 90° car, comme nous l'avons vu, il s'agit du même signal, mais observé avec une rotation de 90° ou un quart de tour (d'où le terme « en quadrature »).

D'où proviennent ces données me direz-vous ? Tout simplement du récepteur lui-même et ce en relation directe avec la façon donc fonctionne également un émetteur en SDR. Pour écouter une station radio sur, disons, 95 Mhz, votre récepteur DVB-T va faire fonctionner un oscillateur embarqué dans le matériel à la fréquence souhaitée. On parle d'oscillateur local ou LO et parfois de VLO (« V » pour variable).

Le signal produit par LO est ensuite dupliqué : une copie est utilisée telle quelle et une autre est décalée en phase de 90°. Ces deux signaux sont mixés (multipliés par un mélangeur) au signal reçu puis numérisés par deux convertisseurs analogiques/numériques. Les données obtenues sont alors envoyées à votre ordinateur. Le signal mixé avec LO est I et celui mixé avec LO décalé en phase est Q. Voici vos données IQ.

Mais ceci n'est pas très différent d'un récepteur radio classique puisque le fait d'utiliser une fréquence générée localement et mélangée au signal reçu est le principe même du fonctionnement d'un récepteur superhétérodyne dont l'invention remonte à une centaine d'années. La plupart des récepteurs radio sont des récepteurs superhétérodynes.

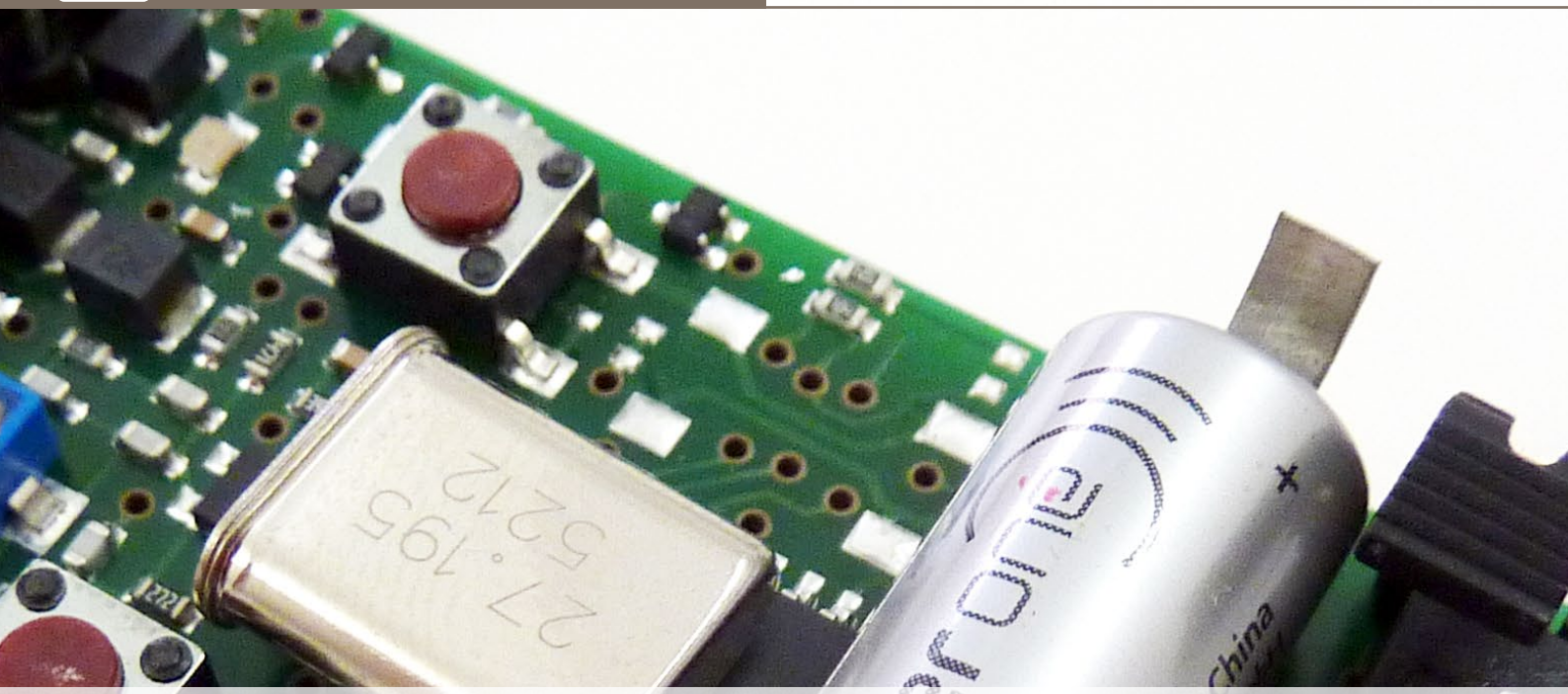
6. ALLER PLUS LOIN

Il est à présent temps de nous tourner vers la pratique et nous pencher sur notre fameuse télécommande de garage. Le sujet du traitement numérique de signaux, vous vous en doutez, va bien au-delà des quelques considérations très sommaires que nous venons d'évoquer. Il existe énormément de littérature sur le sujet aussi bien en français qu'en anglais. Si vous êtes anglophone, je vous recommande de pointer votre navigateur sur **https://greatscottgadgets.com/sdr/**. Il s'agit du site de Michael Ossmann, créateur du HackRF One, un récepteur SDR open source et open hardware assez similaire à un récepteur DVB-T, mais conçu spécialement pour cette tâche. Cette page du site référence une série de vidéos didactiques sur les notions de base de la SDR (DSP, notion de décibels, nombres complexes, etc.). **DB**

[1] http://www.cardin.it/fr-fr/produits/electronique/telecommandes-radio/telecommandes-radio

```
set zlabel "Q"
set ylabel "I"
set xlabel "temps"
set ticslevel 0
set size 1,1;set view 90,270;set yrange[-1:1];set
zrange[-1:1];set xrange[0:30];splot [t=0:30]
t,cos(t),sin(t) with linespoints ls 1 pt 7

plot sin(x), sin(2*pi*30000*x)*sin(x), (sin(2*pi*
30000*x)*sin(x))*sin(2*pi*30000*x)
```

coromonadalix



SDR ET TÉLÉCOMMANDE : QUI PEUT ENTRER DANS MON GARAGE EN PRATIQUE ?

Denis Bodor

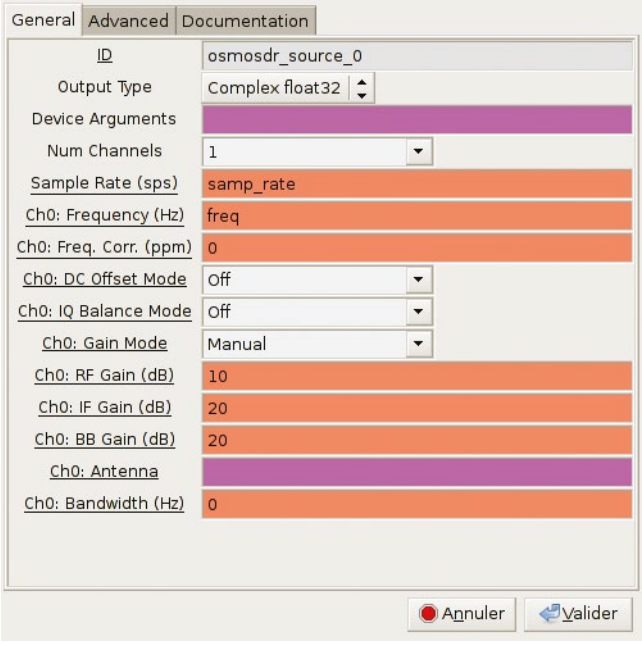
Après l'incontournable, nécessaire et peut-être pénible, introduction à GNU Radio, il est maintenant temps de passer aux choses sérieuses. Nous avons des questions, des interrogations, des inquiétudes et des doutes qui demandent réponses, éclaircissements et dissipations. Armé de nos connaissances basiques en SDR, mais d'une forte détermination, il est temps de découvrir ce que notre télécommande hurle à tout va lorsqu'on appuie sur ses boutons !

coromonadalix

Nous allons en grande partie uniquement utiliser GNU Radio Companion et un récepteur USB DVB-T utilisant un tuner RT820. GNU Radio nous fournit tous les outils dont nous allons avoir besoin pour obtenir, analyser et comprendre le signal émis par la télécommande. Nous allons procéder par étape suivant une méthode relativement courante. Précisons enfin que les données que le constructeur fournit nous permettent déjà d'avoir de bonnes pistes pour cette exploration. Nous avons en effet, la fréquence utilisée ainsi que le type de modulation (ASK).

1. FFT : TROUVER LE SIGNAL

Avant toutes choses, préparons le strict minimum pour nos opérations. Cette fois, il ne s'agira plus d'utiliser une source synthétique comme nous l'avons fait précédemment, mais nous utiliserons le bloc **`osmocom`**. Créez donc un nouveau graphe dans GNU Radion Companion et glissez le bloc en question dans l'espace de travail. Double-cliquez ensuite pour obtenir ses propriétés :



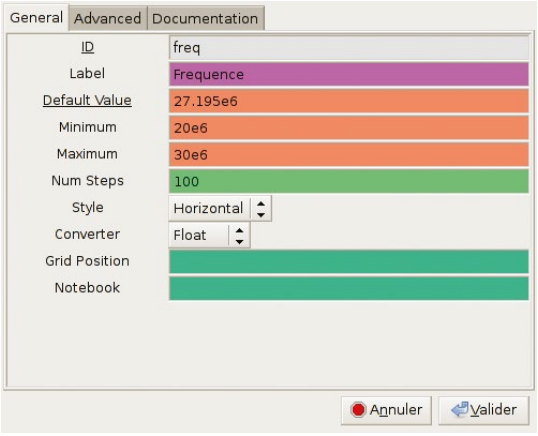
La plupart des valeurs par défaut ne demandent aucun changement. Deux cependant sont capitales :

- *Sample rate (sps)* : sa valeur est actuellement **`samp_rate`** qui est le nom d'une variable existant sous la forme d'un bloc dans notre graphe. Pour changer ce nombre d'échantillons par seconde (à 32000 par défaut), nous n'allons pas toucher à sa valeur ici. En revanche, nous passerons la valeur de **`samp_rate`** à **`2e6`**, soit 2 millions d'échantillons par seconde qui est, pour un récepteur RTL-SDR, plus ou moins la valeur maximum.
- *Ch0: Frequency (Hz)* : c'est la fréquence sur laquelle doit se caler le récepteur en hertz. Par défaut, nous avons **`100e6`** qui correspond donc à 100 Mhz. Ici, plutôt que de renseigner une nouvelle valeur, nous allons spécifier **`freq`** qui sera une variable qui reste à définir.

Une autre propriété qui est souvent ajustée est « *Ch0: Gain Mode* » qui détermine la façon de régler le gain ou en d'autres termes l'amplification du signal reçu. Vous avez le choix entre un réglage manuel ou automatique. Par défaut, le choix est manuel et il est possible, selon le matériel utilisé, de spécifier l'amplification pour différentes étapes (*stage*) du traitement du signal dans le matériel (RF pour le signal reçu, IF pour le signal après mixage/mélange, et BB pour la fréquence de base qui « porte » le signal). En débutant, il faut garder principalement une chose à l'esprit : en augmentant le gain, vous augmentez l'amplification non seulement du signal, mais également du bruit. Le réglage automatique du gain correspond aux fonctions AGC (*Auto Gain Control*) des récepteurs, ajustant le gain en fonction du bruit pour améliorer la réception. Dans notre cas, un gain manuel avec les valeurs par défaut fera parfaitement l'affaire.

coromonadalix

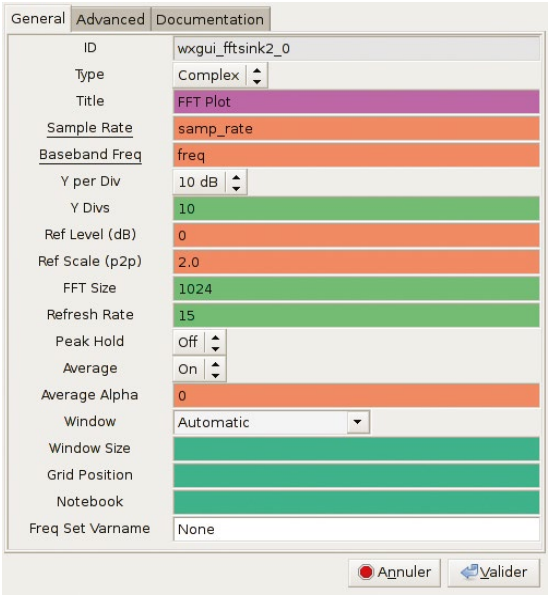
Nous venons d'utiliser une variable, **freq** qui n'est définie nulle part. Nous allons donc corriger ce point, non pas en ajoutant un bloc *variable*, mais en utilisant un composant de l'interface graphique : un *slider* WX GUI. Trouvez le bloc « *WX GUI Slider* » dans la liste et glissez-le dans la zone de travail puis éditez ses propriétés :



Nous spécifions l'*ID* correspondant au nom de la variable et donc **freq**. Il s'agit d'un élément de l'interface, un *slider*, qui est une sorte de curseur réglable dont la position détermine une valeur. Nous pouvons spécifier un label (**Frequence**), une valeur par défaut qui correspond à la fréquence supposée de la télécommande en hertz (**27.195e6** pour 27195000), une valeur minimum (**20e6**) et maximum (**30e6**). Ceci nous permettra éventuellement d'ajuster la fréquence de réception entre 20 Mhz et 30 Mhz en cours de fonctionnement.

Nous avons donc une source et un ensemble de paramètres, il ne nous reste plus qu'à définir une « sortie » ou *sink* (l'inverse d'une source, quelque chose qui absorbe un signal). Il existe un élément graphique qui se prête idéalement à notre première approche : *WX GUI FFT Sink*. C'est une représentation graphique des fréquences reçues avec en abscisse les fréquences et en ordonnée l'amplitude du signal à une fréquence donnée. Une telle représentation est produite par un algorithme appelé transformation de Fourier rapide ou FFT (pour *Fast Fourier Transform* en anglais), qui permet de transformer des données discrètes du domaine temporel dans le domaine fréquentiel et donc de produire un analyseur de spectre. Oubliez temporairement le jargon technique et mathématique, vous pourrez vous y attacher plus tard, retenez ici qu'une FFT vous permet de visualiser les fréquences sur lesquelles un signal est présent et à quelle amplitude.

Cherchez, glissez et affichez les propriétés de ce bloc :



Nous pouvons régler énormément de choses pour ce bloc mais, là encore, la plupart des valeurs par défaut font l'affaire. Il faut savoir également qu'il s'agit d'un élément de l'interface graphique et que celui-ci dispose de différents éléments de contrôle. La majorité des propriétés que vous définissez ici peuvent être changées dans l'exécution du graphe/programme GNU Radio. Nous allons ici ajuster deux choses :

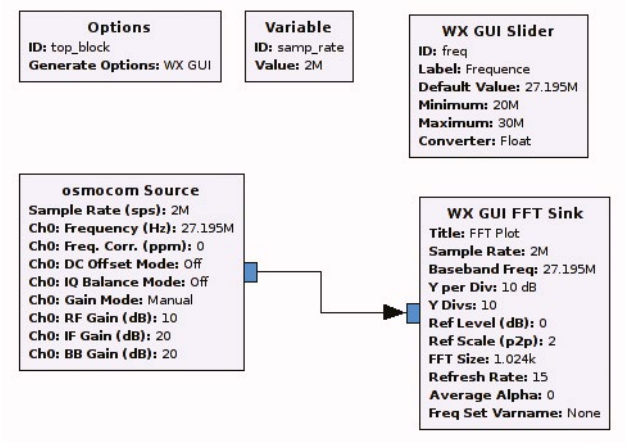
- *Average* qu'on passera à *On* afin d'adoucir l'animation en n'affichant pas directement une représentation des valeurs, mais en calculant une moyenne flottante. Le résultat est moins « vivace » et donc plus facile à lire pour notre expérimentation ;
- *Baseband Freq* qui est la fréquence de base sur laquelle nous nous sommes calés. Nous précisons ici simplement notre variable **freq**.

Ce second point est important. Rappelez-vous du fonctionnement du récepteur, nous réglons un

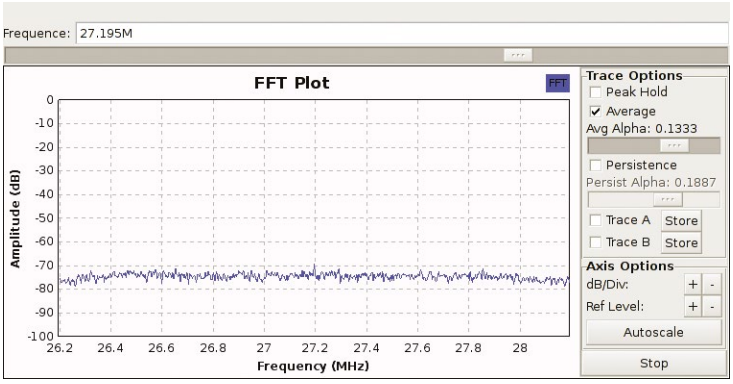
oscillateur local (dans le récepteur) sur une certaine fréquence, celle-là même qui est utilisée par le bloc *osmocom*. Cette fréquence est mélangée en phase et en quadrature avec le signal reçu et les deux signaux résultants sont numérisés et passés au PC sous la forme d'échantillons IQ représentés par des nombres complexes.

Le PC lui, ne voit pas de fréquence de base, il reçoit un certain nombre d'échantillons par seconde (2000000 en l'occurrence) qui représentent des points dans un signal. Notre graphique va donc par défaut afficher des fréquences qu'il aura calculées, mais uniquement sur la base des échantillons. Un signal de forte amplitude capté, par exemple, 300 kilohertz de plus que la fréquence de base sera donc affiché comme étant un signal de 300 kilohertz et une fréquence en dessous de celle de base comme une fréquence négative. Nous ne pousserons pas ici davantage cette notion, mais souvenez-vous qu'un signal représenté par une liste d'échantillons complexes n'est rien d'autre qu'une rotation sur le plan complexe, rotation qui peut se faire dans un sens, mais aussi dans l'autre. Ceci n'est pas discernable en observant uniquement la composante réelle, car cos(x) = cos(-x). Le fait de régler *Baseband Freq* sur `freq` nous permet d'aligner le graphique FFT avec comme point central la fréquence sur laquelle nous réglons notre réception.

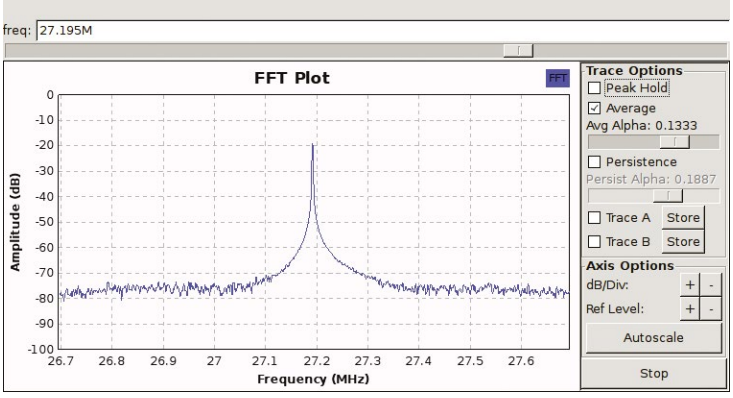
Il ne nous reste plus qu'à relier la source *osmocom* avec le *WX GUI FFT Sink* pour obtenir le graphe complet suivant :



Enregistrez le fichier avec un nom qui vous convient puis exécutez le graphe en appuyant sur F6. Presque instantanément, vous devez voir apparaître une nouvelle fenêtre présentant notre *slider* ainsi que le graphique FFT.



Nous pouvons à présent approcher la télécommande de l'antenne du récepteur et appuyer sur l'un des boutons. Rapidement, nous verrons le graphique devenir ceci :



Nous voyons un pic très clair avec une amplitude maximum se trouvant précisément à la fréquence de 27,195 Mhz. Remarquez que des signaux des fréquences sensiblement supérieures et inférieures sont également tracés, mais l'amplitude se réduit à mesure qu'on s'éloigne de la fréquence centrale.

DÉCIBEL

Le terme décibel est quelque chose que vous risquez de voir souvent en explorant le domaine de la radio (mais pas seulement). Il est important de bien comprendre ce qu'est un *déci-bel*, autrement dit un dixième de *bel* (1 dB = 1/10 B et 10 dB = 1 B). Cette unité de mesure apparaît sur le graphique FFT, en ordonnée, par exemple.

Cette unité définit un ordre de grandeur, sous la forme d'une puissance de 10. Un ratio de 100 pour 1 (100/1) est un ratio de 2 bels (10^2), 10 pour 1 est 1 bel, 10000 pour 1 est 4 bels, etc. On peut généraliser en disant que si le ratio de A/B=10^n, alors n est le nombre de bel.

Le bel ce n'est pas une unité de mesure de *quelque chose*. Rien n'a 1 bel, 15 dB ou 2 dB. C'est une unité de grandeur ! Exemple : si un ami fait 1 km à vélo et que vous en parcourez 10, vous faites 1 bel ou 10 dB de sa distance. Si vous en faites 100, vous parcourez 2 bels, ou 20 dB de sa distance.

Vous comprendrez alors qu'il ne peut y avoir que des décibels par rapport à quelque chose. Dans notre exemple, la distance que vous parcourez est 20 dB plus importante. On peut aussi dire que votre distance est de 20 dB distance-ami.

Ceci marche également dans le sens opposé. Vous parcourez 100 fois plus de kilomètres que votre ami et donc 20 dB de sa distance. Lui en revanche, parcours -20 dB de la vôtre, soit 1/100 de votre distance. En revanche, vous ne pouvez pas dire qu'il a parcouru -20 dB de moins que vous, -20 dB exprime déjà qu'il a fait moins de kilomètres que vous. Vous pouvez dire qu'il a fait 20 dB de moins, ou -20 dB de votre distance.

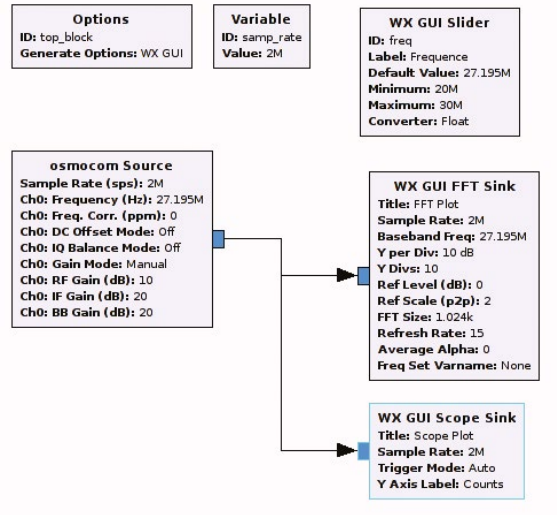
Sur notre graphique FFT, on peut constater qu'il s'agit de décibels relatifs à l'amplitude maximum du signal. Mais plus important encore, c'est l'échelle qui est importante, car notre signal sur 27,195 Mhz a une amplitude de -20 dB du maximum (100 fois moins) et le bruit de fond sur les fréquences adjacentes est à -80 dB du maximum (soit -60 dB de notre signal, 1000000 de fois moindre). Remarquez comme les décibels permettent de facilement exprimer des ratios très importants et comme il est bien plus simple de calculer des différences entre ces ratios.

Une bonne astuce pour s'en sortir avec les décibels est de retenir que 3 dB est un rapport de 2/1 et 10 dB est 10/1. Quelque chose de deux fois plus lourd qu'une brique est 3 dB-brique lourd. Quelque chose de deux fois plus léger est -3 dB-brique lourd.

Si vous ne savez pas dans quelle fréquence émet votre télécommande malgré une recherche sur le Web, vous pouvez essayer de simplement tâtonner dans les bandes généralement utilisées (bande ISM) comme 26,957 à 27,283 MHz, 40,660 à 40,700 MHz ou encore 433,05 à 434,79 MHz. Cette dernière est très largement utilisée par énormément d'appareils de toutes sortes (capteurs, télécommandes, alarmes, etc.). Pour être sûr que la fréquence est la bonne, le plus simple est de tout simplement approcher votre télécommande de l'antenne. Si le signal est plus fort à mesure que vous vous approchez, c'est très certainement le bon signal.

2. OSCILLOSCOPE : VOIR LE SIGNAL

Nous le savons maintenant, ou du moins en avons une confirmation, notre télécommande S466-TX2 émet bien à 27,195 Mhz. La prochaine étape consiste à avoir un aperçu du signal pour savoir précisément ce qu'il transporte comme information. Nous sortons donc de notre manche un nouvel outil sous la forme d'un bloc *WX GUI Scope Sink* que nous glissons puis connectons à la source *osmocom*, en parallèle du *WX GUI FFT* :



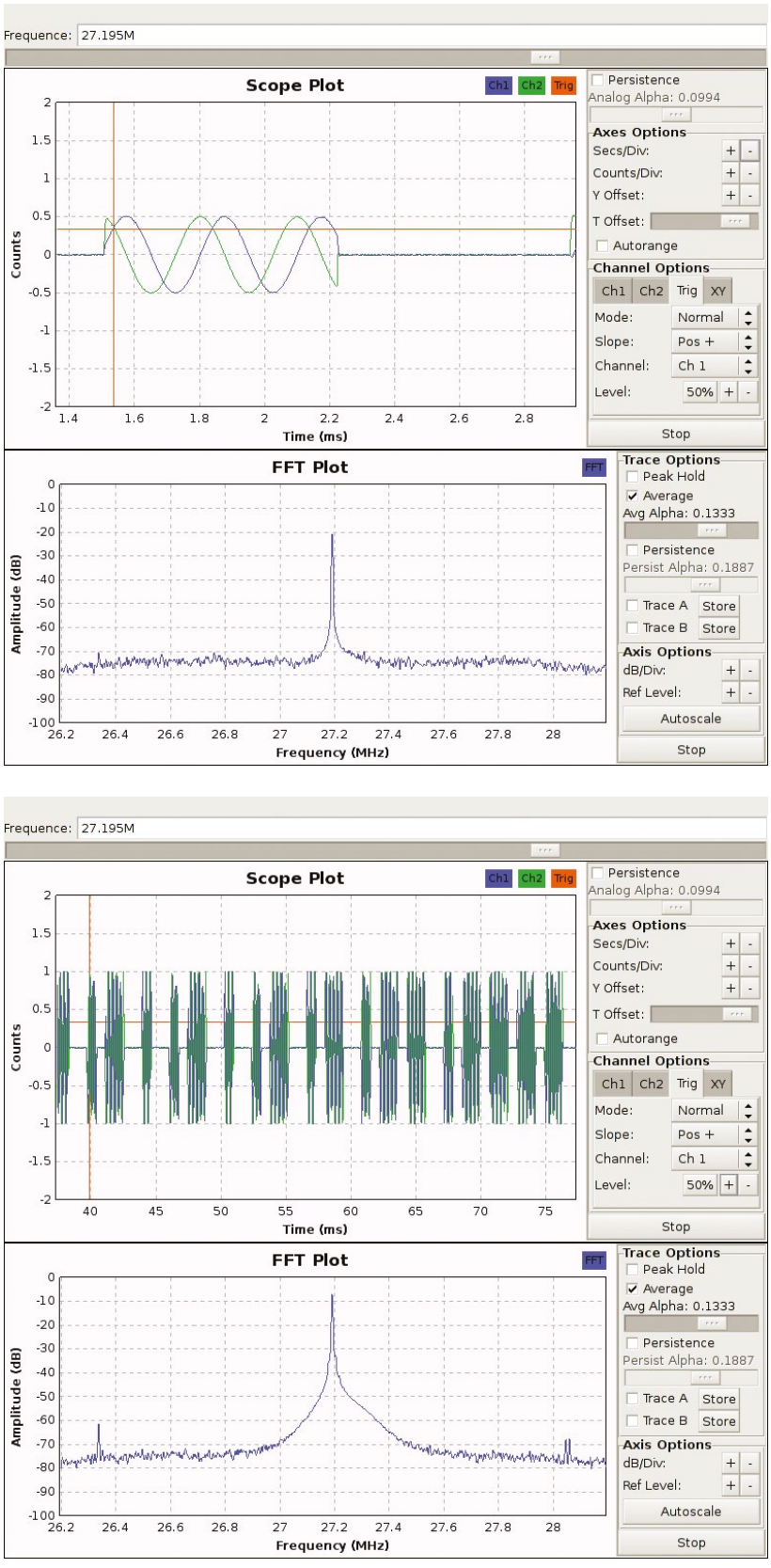
coromonadalix

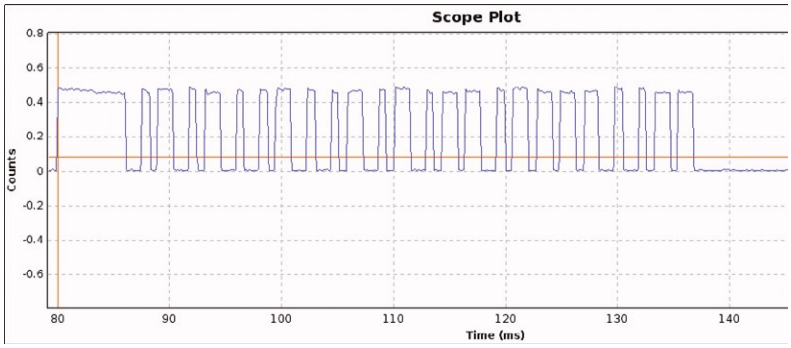
Les propriétés par défaut nous vont parfaitement, nous allons procéder aux réglages en direct en exécutant le graphe. Dans la partie supérieure se trouve maintenant le *scope*. Commencez par désactiver l'*Autorange* afin de pouvoir régler manuellement les échelles en temps (horizontal) et en *counts* (vertical). Sur la droite, vous pouvez jouer sur les boutons +/- de *Sec/Div* pour le temps et *Count/Div* pour l'axe vertical.

En appuyant à nouveau sur le bouton de la télécommande, vous devez voir le signal prendre forme. Le problème est que cela ne dure que le temps de l'émission, ce n'est pas très pratique. Il serait plus intéressant de déclencher la capture uniquement lorsque le signal dépasse un certain seuil. Voilà précisément l'intérêt d'un *trigger* (comme sur un oscilloscope). Toujours sur la droite, en bas, se trouvent une série d'onglets dont un marqué *Trig*. Réglez le mode en *Normal* et montez le *Level* avec le bouton +.

Des tracés rouges vous indiquent la position du déclencheur (*trigger*) et le tracé est en pause. Si vous appuyez sur le bouton de la télécommande, il s'animera. Vous pouvez maintenant régler plus aisément les échelles pour obtenir quelque chose comme sur la figure en haut à droite.

C'est certes fascinant, mais pas vraiment parlant. On ne distingue pas vraiment de données modulées de cette façon. On notera cependant que le signal n'est pas émis de façon continue, mais sous la forme de brèves émissions qui débutent et s'arrêtent brutalement. Si nous changeons l'échelle de temps, en revanche, les choses changent (voir figure ci-contre).

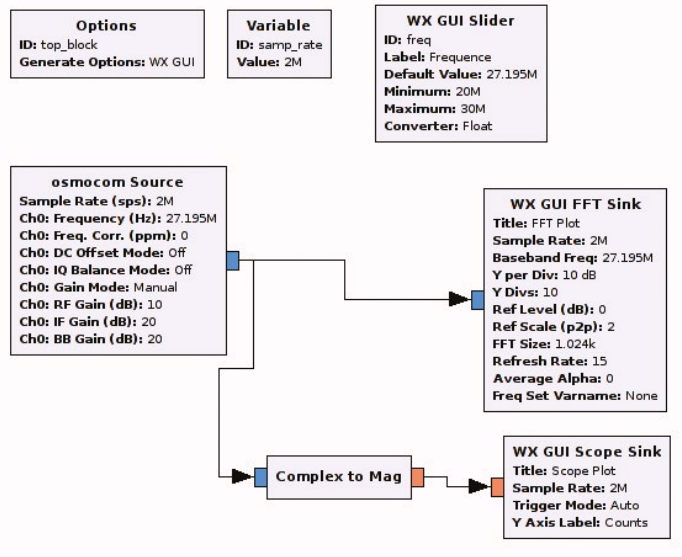




On distingue maintenant clairement qu'il y a des petits paquets de signaux, certains plus grands que d'autres, qui se suivent à la queue leu leu. C'est un progrès, car ceci doit avoir une signification.

Le problème ici est la forme du signal. Nous voyons clairement que tantôt nous avons un signal avec une certaine amplitude et tantôt aucune amplitude. Mais l'amplitude peut être positive (haut) ou négative (bas). Il serait donc intéressant de visualiser cet écart par rapport à l'axe horizontal de façon absolue. En d'autres termes, ce n'est pas l'amplitude du signal qui nous intéresse, mais sa magnitude. GNU Radio Companion a ce qu'il nous faut, un bloc *Complex to Mag* qui prend en entrée un nombre complexe et en calcul la magnitude.

En glissant le bloc dans votre espace de travail, vous constaterez que sa sortie est d'une couleur différente. L'entrée est bleue, car c'est un nombre complexe qui est attendu, mais en sortie c'est une simple valeur en virgule flottante (*float*). Il est donc possible de connecter la source *osmocom* au bloc *Complex to Mag*, mais pas à notre *scope*. Nous devons changer les propriétés du *scope* et changer le *Type* de *Complex* à *Float*. Ceci fait, il ne reste plus qu'à tout connecter ainsi :



À présent, lorsque nous exécutons notre graphe, nous voyons dans la partie *scope* (après ajustement des échelles) que nous avons là clairement quelque chose :

Chaque pression sur le bouton génère une succession de « longs » et de « courts » après une sorte de préambule très long. Ce à quoi nous avons affaire ici, comme il est spécifié dans la documentation du constructeur, est une modulation de l'amplitude du signal ou *Amplitude-Shift Keying* (ASK). Des données numériques sont représentées par une variation de l'amplitude d'un signal à une certaine fréquence, sur une certaine durée.

3. ESSAIS ET TÂTONNEMENTS

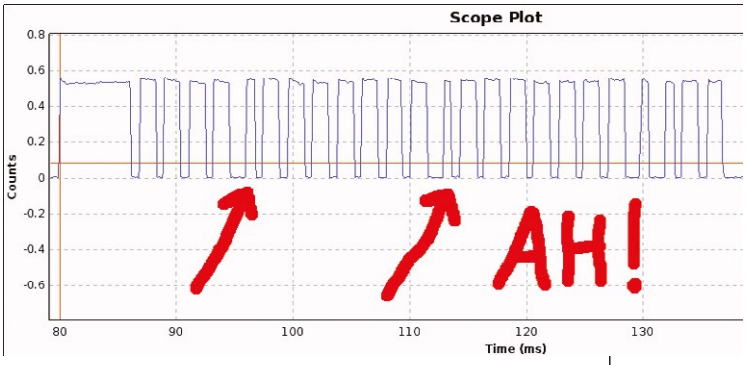
Nous voici maintenant de retour dans un domaine plus « informatique » puisque la tâche qui se présente à nous consiste à trouver comment sont encodées les données. Quelles données me direz-vous ? On peut supposer que si la télécommande utilise des micro-interrupteurs pour configurer un « code », il y a de fortes chances que ce code transparaisse d'une certaine façon dans le signal que nous pouvons maintenant voir.

Si nous faisons abstraction du préambule, nous voyons clairement 24 morceaux de forte magnitude. Nous voyons également que nous avons deux « types » de morceaux, des longs et des courts.

coromonadalix

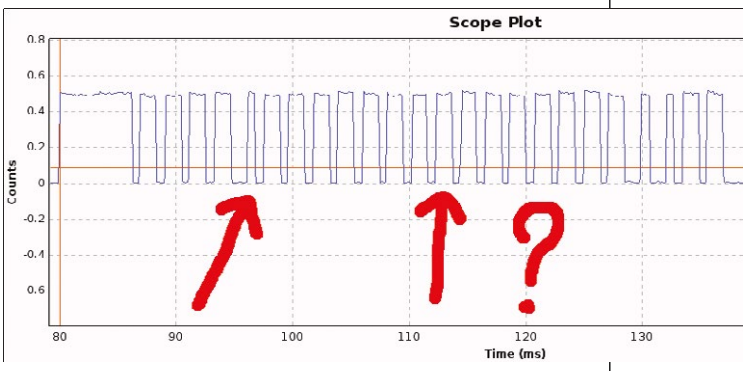
La télécommande utilise 9 interrupteurs sur trois positions, soit 19683 codes possibles. On peut donc éliminer d'office une éventuelle correspondance du type long = une position et court = une autre.

Chaque interrupteur peut être mis en position « + », « o » ou « - ». À première vue, il nous faut donc obligatoirement au minimum deux morceaux longs ou courts pour chaque interrupteur. 2 fois 9 interrupteurs nous donnent 18, pas 24. Ce n'est pas grave, nous allons essayer en partant du principe qu'une partie de ce que nous nommerons maintenant « bits du signal » n'est tout simplement pas utilisée pour ces 9 interrupteurs.

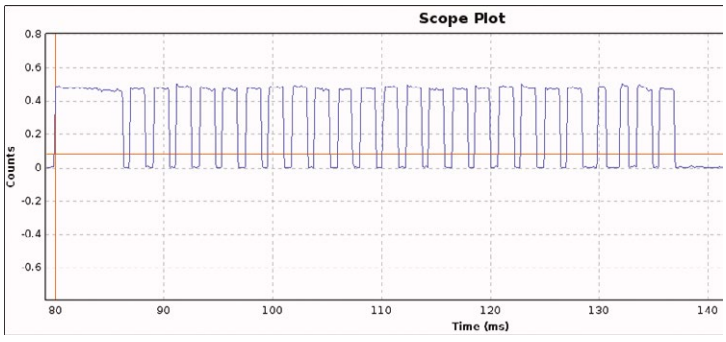


Tiens donc ! Nous avons maintenant `000010000000100000001100`. Voyons ce que cela donne avec le même interrupteur sur « o » :

La suite est simple, mais pénible. Il nous faut essayer différentes combinaisons et observer les changements. Ici, je n'ai pas voulu jouer avec la télécommande réellement utilisée, craignant d'user prématurément les interrupteurs, oublier la combinaison initiale ou encore de simplement vider la pile. J'ai donc commandé une télécommande identique de remplacement et me suis assuré qu'elle fonctionnait avec une configuration identique.



J'ai ensuite placé tous les interrupteurs sur « - » et obtenu ceci :



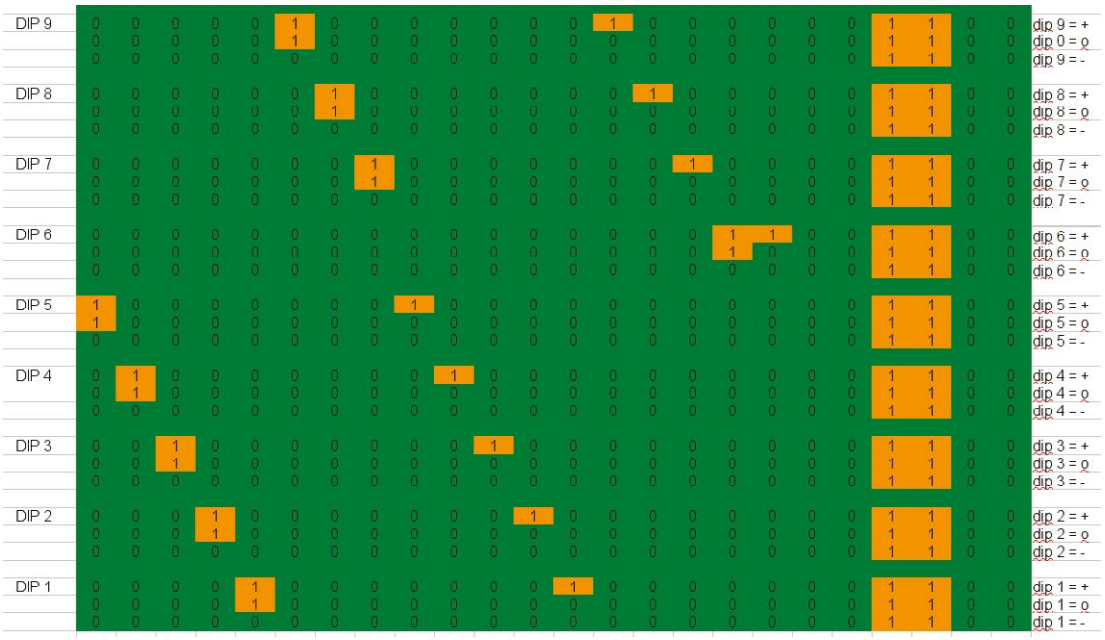
Intéressant, le signal est devenu `00001000000000000001100`. Nous avons bien une paire de bits qui change en influant sur un seul interrupteur. Nous pouvons provisoirement en conclure que :

- `00` = "-"
- `10` = "o"
- `11` = "+"

On constate clairement que la majorité des morceaux sont maintenant les « longs » et qu'une partie, à la fin, ne semble pas changer. On décide donc joyeusement d'arbitrairement de définir un long comme un bit de signal à « 0 ». Ce qu'on voit sur le *scope* est donc `00000000000000000001100`.

À présent, il n'y a plus qu'à mettre le premier interrupteur sur « + » et de voir ce qui se passe :

Sur cette base, et à la fois pour confirmer cette théorie et trouver la position de chaque paire, il nous suffit de procéder de même avec tous les interrupteurs, un par un. Mais un bon programmeur/bidouilleur étant par définition paresseux, mieux vaut ne pas faire des dizaines de captures d'écran, mais plutôt faire appel à un outil plus adapté : un simple tableur pour noter nos résultats.



Après quelques dizaines de cycles configuration/capture/saisie, et grâce aux styles conditionnels de LibreOffice, on finit par obtenir le tableau ci-dessus.

Nous avons fait le tour des interrupteurs et nos soupçons se sont confirmés concernant les combinaisons de bits et leur signification au regard de la position des interrupteurs. Mieux encore, nous avons à présent un tableau de correspondances et pouvons simplement vérifier notre théorie. Si nous plaçons les interrupteurs sur une position aléatoire et observons le signal, nous retrouvons les correspondances avec une parfaite exactitude.

Et les 6 derniers bits me direz-vous ? Ils correspondent simplement au « canal » de la paire d'interrupteurs supplémentaires configurant l'utilisation du second bouton de la télécommande (celui de droite), avec une logique assez similaire.

Vous rendez-vous compte de ce que cela signifie ? La télécommande transmet directement, sans aucune protection, la position des interrupteurs censée définir un « code secret haute fiabilité ». Nous n'avons pas ici à faire à une quelconque protection, mais à un simple encodage d'informations. D'un point de vue « informatique », on dit que l'information est tout simplement transmise en clair. Autrement dit, la télécommande « hurle » son code à qui veut bien l'entendre sans la moindre pudeur !

Une solution plus poussée, mais demandant bien plus d'efforts et de temps consiste à utiliser GNU Radio et un ensemble de blocs de traitement pour ensuite réutiliser le code Python produit pour le compléter en programmant. Il est aussi possible d'écrire un code Python s'interfaçant avec votre graphe GNU Radio Companion ou encore de créer un programme dans le langage qui vous plaît recevant des informations de GNU Radio.

Tout ceci sort du cadre de cet article qui se veut une introduction à l'utilisation de GNU Radio et des outils SDR. Cependant, mon langage préféré étant le C, je me suis personnellement orienté vers un projet existant appelé *rtl_433*. Le but initial de ce programme est de décoder et d'afficher des informations transmises par

4. ALLER PLUS LOIN

Le décodage du signal à la main n'est pas très amusant (enfin si, une fois, au départ). Il est possible d'automatiser la chose en utilisant le tableur et une série de formules pour obtenir quelque chose de relativement joli comme ceci :

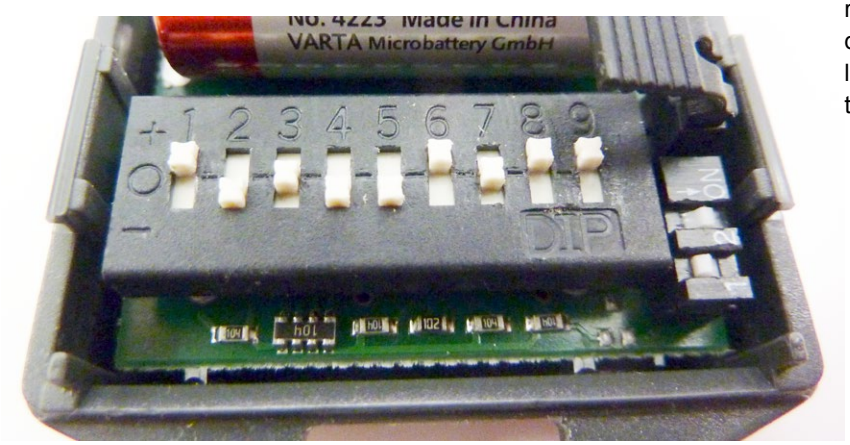
DIP	1	2	3	4	5	6	7	8	9
POS	2	0	2	0	2	0	2	0	1
+	X		X		X		X		
o									X
-		X		X		X		X	

différents équipements dans la bande 433,05 à 434,79 MHz (433.92 MHz par défaut). Les matériels en question sont par exemple des capteurs *Oregon Scientific*, des télécommandes de prises murales et différents capteurs de températures.

Cependant, comme le code de *rtl_433* est architecturé de manière à faciliter l'ajout de matériel, il m'a suffi de créer un seul fichier source complétant ainsi le support déjà existant. Voyez-vous, c'est là le principe même du logiciel libre : avoir un besoin, trouver un projet, comprendre et améliorer le code. Naturellement, une fois le support fonctionnel, je me suis empressé de proposer à l'auteur d'ajouter mon code dans le sien. À présent, ce support pour les télécommandes Cardin S466-TX2 est officiellement intégré dans *rtl_433*.

Résultat :

```
% rtl_433 -f 27195000
[...]
Using device 0: Generic RTL2832U OEM
Found Rafael Micro R820T tuner
Exact sample rate is: 250000.000414 Hz
Sample rate set to 250000.
Sample rate decimation set to 0. 250000->250000
Bit detection level set to 10000.
Tuner gain set to Auto.
Reading samples in async mode...
Tuned to 27195000 Hz.
----------------------------
protocol       = Cardin S466
message        = 10001010 10000000 11001001

                 123456789
dipswitch      = o---++-o-

                 -->ON
right button   = 2 --o (this is right button)
                 1 o--
```



Gros plan sur les fameux micro-interrupteurs permettant de configurer le code « secret » avec, à droite, deux autres interrupteurs définissant la configuration du second bouton de la télécommande.

rtl_433 n'a pas besoin de GNU Radio et repose directement sur les bibliothèques RTL-SDR donnant accès aux dongles DVB-T compatibles. Ceci signifie que ce programme ne fonctionnera qu'avec les récepteurs RTL-SDR et non avec d'autres périphériques comme le HackRF One. En contrepartie, il offre une solution clé en main pour obtenir des données de nombreux objets utilisant une communication radio. Si vous programmez en C, même un peu, je vous recommande très fortement de contribuer à ce projet. Les explications données ici devraient vous permettre de tester vos télécommandes RF et sans doute d'autres équipements. Si vous obtenez un résultat dans vos expérimentations, ne vous arrêtez pas en si bon chemin. Étudiez le code de *rtl_433*, programmez un support pour vos trouvailles et contribuez, c'est très facile avec GitHub.

Enfin, si d'aventure vous n'avez pas encore réalisé la gravité de la situation : si vous possédez un tel équipement, et je ne parle pas seulement du S466-TX2, empressez-vous de le changer ou de le faire changer, car votre sécurité repose sur un pseudo-secret de polichinelle. Un peu comme lorsqu'on glisse sa clé sous le tapis de porte...

coromonadalix

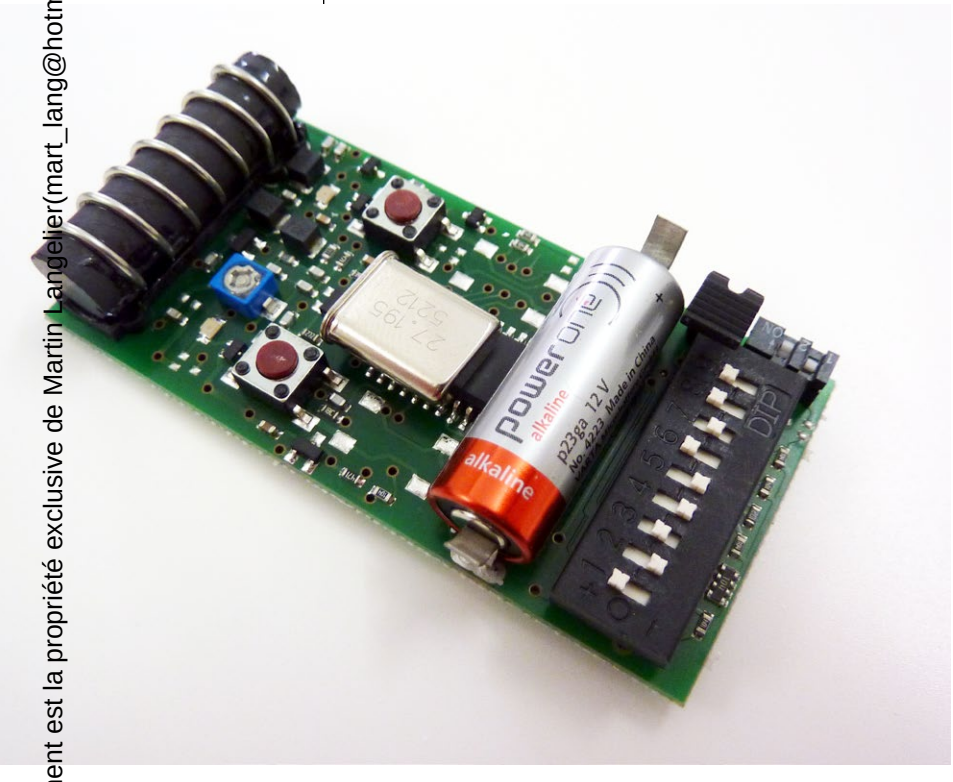
POUR CONCLURE

Précisons ici que le problème n'est absolument pas spécifique à cette marque et encore moins à cette fréquence. La plupart des équipements de télécommandes actuellement utilisés reposent sur ce type de technique et de modulation, même en 433 Mhz. Ceci va de la télécommande de portail aux jouets télécommandés en passant par différents capteurs ou encore les prises électriques télécommandées. Les équipements modernes de qualité utilisent maintenant une technique appelée *rolling codes* qui permet de ne jamais envoyer deux fois les mêmes données et donc d'éviter le rejeu. C'est ce type de technique qui est utilisé pour le verrouillage des voitures à distance par le biais d'une télécommande. Ce système n'est cependant pas parfait, mais il présente au moins l'avantage de rendre la tâche plus difficile aux personnes mal intentionnées.

Le nombre d'installations utilisant un système de télécommande du même type que celui que nous avons étudié ici est très important et ceci ne risque malheureusement pas de changer de si tôt. Chaque fabricant maintient toujours les gammes de produits non sûrs pour des raisons de compatibilité (qui souhaite vraiment changer toute son installation lorsqu'il suffit d'acheter une nouvelle télécommande ?). Le problème avec la télécommande Cardin S466-TX2 n'est pas tant la technologie elle-même, mais le fait qu'elle soit présentée comme offrant une « sécurité absolue » encore aujourd'hui, et ce sur la même page web qui propose également des solutions plus sûres à base de *rolling codes*.

La documentation constructeur que nous avons trouvée date de 2008, avant l'arrivée de la SDR pour tous à 20 euros. Mais même en ce temps-là, il était parfaitement possible avec un simple récepteur HF (ou ondes courtes) et une carte son de récupérer le code utilisé. La solution n'était, en ce temps-là déjà, pas sûre et certainement pas basée sur un « code secret haute fiabilité ».

Vous devez vous douter qui, selon moi, est à blâmer pour une telle promotion trompeuse dans le descriptif du produit. Ce ne sont bien entendu pas les personnes chargées du développement de ces produits et les ingénieurs, mais encore et toujours ceux prêts à faire n'importe quels raccourcis médiocres et autres affirmations insensées pour déclencher une vente. Donc, Cardin, un simple conseil : tenez donc vos marketeux en laisse, car ce sont eux qui détruisent petit à petit la réputation que vos ingénieurs essaient de construire ! **DB**



Difficile de résister à la tentation d'ouvrir la bête, mais il n'y a finalement pas grand-chose à inspecter. Une puce apparemment dédiée, quelques composants passifs, un circuit d'amplification et c'est tout. Remarquez les deux emplacements vides pour des boutons poussoirs supplémentaires. Ce circuit est sans le moindre doute le même pour la version 4 voies TX4.

NE MANQUEZ PAS OPEN SILICIUM N°14 !

FOCUS :

ÉTENDEZ LE SUPPORT MATÉRIEL D'ANDROID !

...pour prendre en charge vos périphériques

DISPONIBLE CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :

www.ed-diamond.com

coromonadalix

innorobo

1-3 Juillet
Lyon - France

Les technologies et systèmes robotiques apportent des solutions à nos enjeux sociétaux planétaires et contribuent à l'amélioration de notre qualité de vie. Sur cette conviction partagée, nous avons construit et développé un sommet robotique de 3 jours qui réunit un vaste écosystème d'affaires et qui a pour objectif la mise sur le marché des innovations robotiques le plus rapidement possible.

Innorobo™ 2015 développera 6 thématiques majeures d'applications robotiques

Artificial Intelligence
Materials, Mechatronics
Nano Technologies & Electronic
Software and Components
Big data, HRI
Cloud robotics

Nouveautés 2015

Village des Développeurs et Espace Makers

Avez-vous déjà pensé à **programmer des robots ?** Vous maîtrisez le C, Java, ou bien des environnements types comme **ROS, URBI, Matlab ou bien Chorégraphe ?** Venez montrer vos talents et **échanger avec des experts !**

Passionnés d'électronique et de mécanique ? Venez exposer à Innorobo pour montrer votre savoir faire, dans l'unique lieu en Europe réunissant imprimantes 3D et robots, sociétés robotiques, labos de recherche, startups, développeurs et passionnés.

On vous attend !

www.innorobo.com

YouTube theinnorobo

coromonadalix

SYROBO · ONLYLYON GREATER LYON · RhôneAlpes Région · GDR robotique · INNOECHO