N°19 | JUILLET - AOÛT 2017

HACKABLE MAGAZINE

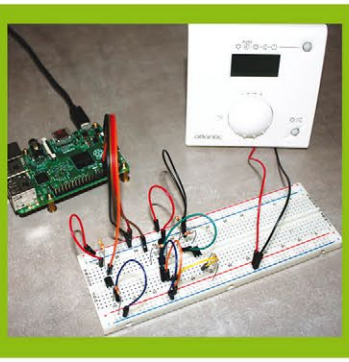
DÉMONTEZ | COMPRENEZ | ADAPTEZ | PARTAGEZ

France MÉTRO. : 7,90 € – CH : 13 CHF – BEL/LUX/PORT.CONT : 8,90 € – DOM/TOM : 8,50 € – CAN : 14 $ CAD

DOMOTIQUE / ANALYSE

Explorez et comprenez le fonctionnement d'une sonde de température Siemens Atlantic T55

p. 76



DOMOTIQUE / PI

Pilotez votre pompe à chaleur Atlantic avec une Raspberry Pi et le bus Siemens BSB

p. 84

PI / CONFIG

Gardez un œil sur la santé et l'état de votre Raspberry Pi grâce à l'outil vcgencmd

p. 70

RADIO / COMPOSANTS

Construisez un émetteur 433 Mhz pour remplacer vos télécommandes par une carte Arduino

p. 50

Arduino / Raspberry Pi / IoT

Créez des liaisons radio de plusieurs kilomètres

...pour vos projets connectés grâce à LoRa et LoRaWAN p. 18

1. Installez votre concentrateur Raspberry Pi LoRaWAN (ou pas)
2. Rejoignez le réseau communautaire, participatif et gratuit TTN
3. Créez vos sondes connectées à base d'Arduino
4. Affichez les informations collectées sur Cayenne MyDevice

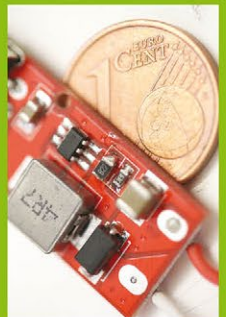
ARDUINO / ÉCRAN

Affichez simplement et sans effort des interfaces avancées grâce aux écrans intelligents Nextion

p. 04

TENSION / USB

Obtenez n'importe quelle tension à partir de l'USB en modifiant un convertisseur chinois à 1€ p. 64



L 19338 - 19 - F: 7,90 € - RD

JDEV 2017

Journées Développement Logiciel

Science des données et apprentissage automatique
Systèmes embarqués et internet des objets
Infrastructures logicielles et science ouverte
Parallélisme itinérant et virtualisation
Ingénierie et web des données
Programmation de la matière
Big data et Sécurité
Usines logicielles
Génie logiciel

Conception graphique : Benoît Colas Université Toulouse - Jean Jaurès / CPRS - DAR. Photographie : © Armand Rotereau.

Webcast

4, 5, 6, 7 juillet 2017

Aix-Marseille Université, la Canebière

JDEV2017

Information, programme, réservation et inscription :

http://devlog.cnrs.fr/jdev2017

Ce document est la propriété exclusive de Alex Arnaud(palmuxdroid@gmail.com)

ÉDITO

« C'est traduit de quelle langue ? »

Telle était la question qu'une personne m'a posée à propos du magazine lors de la MakerFaire Paris 2017 début juin à la Cité des Sciences de la Villette. La réponse est bien entendu simple et évidente : « aucune » (si on part du principe que la correction orthographique et grammaticale de ma prose n'est pas une traduction, affirmation parfois légitime de la part des personnes en charge de cette éprouvante tâche à la rédaction).

Quelque chose me dérange profondément derrière cette simple question. Je ne parle pas tant du fait que nous mettons un point d'honneur à ne détailler que des manipulations et des expérimentations effectivement réalisées par nos soins, même si cela peut être très problématique parfois, lorsque le résultat n'est pas là, il faut se résoudre à passer à la trappe des heures ou des jours de travail et écarter le sujet. Pas question donc de décrire ou d'expliquer quelque chose qui n'a pas été fait, c'est ainsi. Pas question non plus d'ailleurs de publier des articles contribués sans contre-partie financière comme c'est souvent le cas en ligne, ce qui impacte également grandement la qualité des publications. Ce sont des règles immuables, érigées en principes fondamentaux dans nos publications et il n'est pas question d'y déroger (et on peut être vraiment, mais vraiment très têtus, voire butés sur le sujet).

Non, ce qui me dérange véritablement dans cette question est qu'elle sous-entend qu'un contenu rédactionnel technique « touffu » provient forcément d'ailleurs et, je suppose, d'une source anglophone. C'est un peu comme s'il s'agissait de se résigner et d'accepter que, en France, nous ne sommes pas capables de telles réalisations. C'est pourtant oublier que depuis des dizaines d'années, sinon des centaines, c'est bel et bien l'ingéniosité, l'imagination et la créativité française qui ont conduit à d'admirables réalisations et de fantastiques prouesses technologiques.

Autant un « c'était mieux avant » est quelque chose que je peux comprendre, autant un « c'est mieux ailleurs », autrement dit « si c'est bien, ce n'est pas d'ici » est quelque chose qui me parait à la fois insensé et terriblement négatif, voire néfaste. Cela véhicule l'idée que nous aurions perdu, en France, cette capacité technologique, alors qu'il n'y a rien de plus faux. Une simple visite à la MakerFaire Paris le démontre de façon évidente.

Donc non, que cela soit dit et bien dit, nous de traduisons rien dans *Hackable*, car nous n'en avons pas besoin. Et nous n'en avons pas besoin, car la créativité, l'inventivité et la capacité de créer et bidouiller n'ont pas disparu du pays, bien au contraire. J'en suis la preuve, vous en êtes la preuve, nous en sommes la preuve... Tâchons donc de ne pas l'oublier, car le jour où nous en douterons sera précisément le triste jour où nous perdrons réellement cette qualité !

Denis Bodor

Hackable Magazine
est édité par Les Éditions Diamond

LES ÉDITIONS DIAMOND
STIMULANT DE MATIERE GRISE

10, Place de la Cathédrale - 68000 Colmar
Tél. : 03 67 10 00 20 – Fax : 03 67 10 00 21
E-mail : lecteurs@hackable.fr
Service commercial : cial@ed-diamond.com
Sites : http://www.hackable.fr/
http://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
Service des ventes : Abomarque : 09 53 15 21 77
IMPRIMÉ en Allemagne - PRINTED in Germany
Dépôt légal : À parution,
N° ISSN : 2427-4631
Commission paritaire : K92470
Périodicité : bimestriel
Prix de vente : 7,90 €

MIXTE
Papier issu de sources responsables
FSC® C015136

La rédaction n'est pas responsable des textes, illustrations et photos qui lui sont communiqués par leurs auteurs. La reproduction totale ou partielle des articles publiés dans Hackable Magazine est interdite sans accord écrit de la société Les Éditions Diamond. Sauf accord particulier, les manuscrits, photos et dessins adressés à Hackable Magazine, publiés ou non, ne sont ni rendus, ni renvoyés. Les indications de prix et d'adresses figurant dans les pages rédactionnelles sont données à titre d'information, sans aucun but publicitaire. Toutes les marques citées dans ce numéro sont déposées par leur propriétaire respectif. Tous les logos représentés dans le magazine sont la propriété de leur ayant droit respectif.

Suivez-nous sur Twitter
@hackablemag

Alsace

À PROPOS DE HACKABLE...

HACKS, HACKERS & HACKABLE

Ce magazine ne traite pas de piratage. Un **hack** est une solution rapide et bricolée pour régler un problème, tantôt élégante, tantôt brouillonne, mais systématiquement créative. Les personnes utilisant ce type de techniques sont appelées **hackers**, quel que soit le domaine technologique. C'est un abus de langage médiatisé que de confondre « pirate informatique » et « hacker ». Le nom de ce magazine a été choisi pour refléter cette notion de **bidouillage créatif** sur la base d'un terme utilisé dans sa définition légitime, véritable et historique.

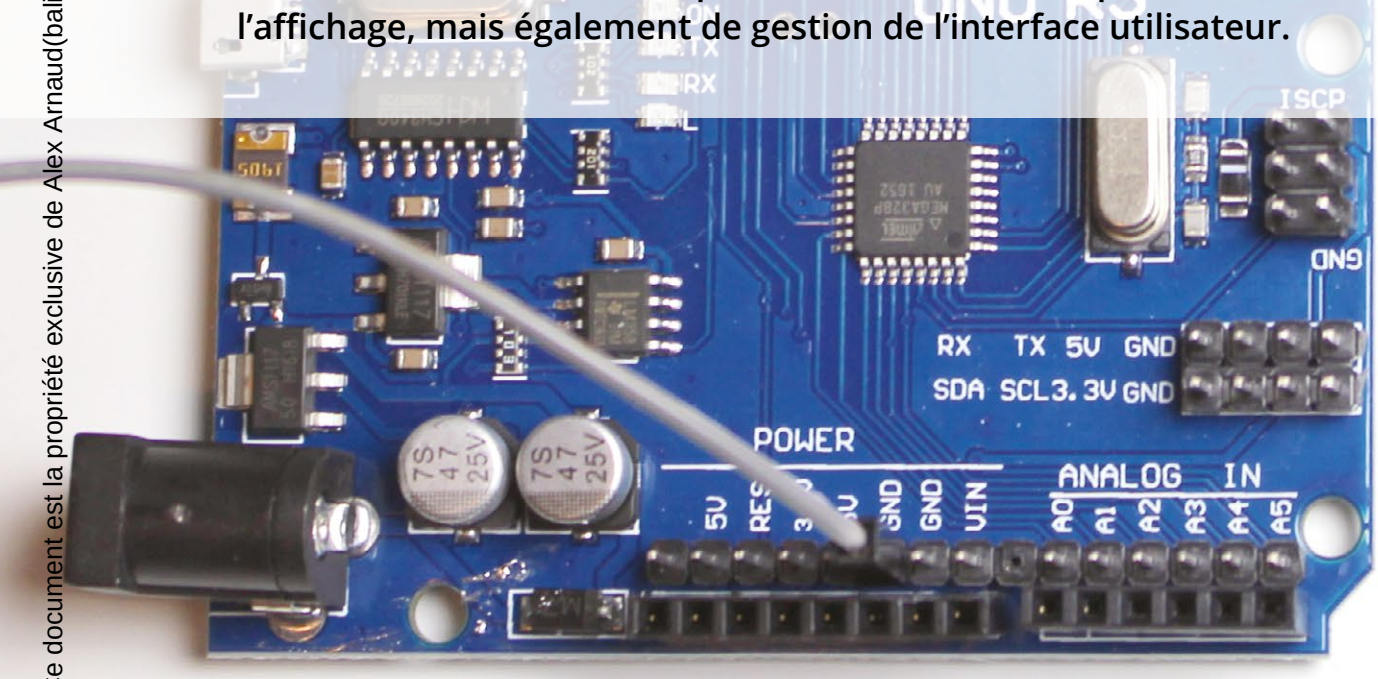
Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)

SOMMAIRE

AJOUTEZ UN ÉCRAN INTELLIGENT À VOS PROJETS ARDUINO

Denis Bodor

Une carte Arduino permet de faire beaucoup de choses, vraiment beaucoup de choses, mais elle possède également des limites dans certains domaines. Quand une telle situation se présente, la solution consiste tout simplement à déléguer une partie du travail à un périphérique ou un module avec lequel votre croquis va communiquer. C'est le principe utilisé par les écrans intelligents Nextion, déchargeant ainsi la carte Arduino de l'éprouvant travail de composition de l'affichage, mais également de gestion de l'interface utilisateur.

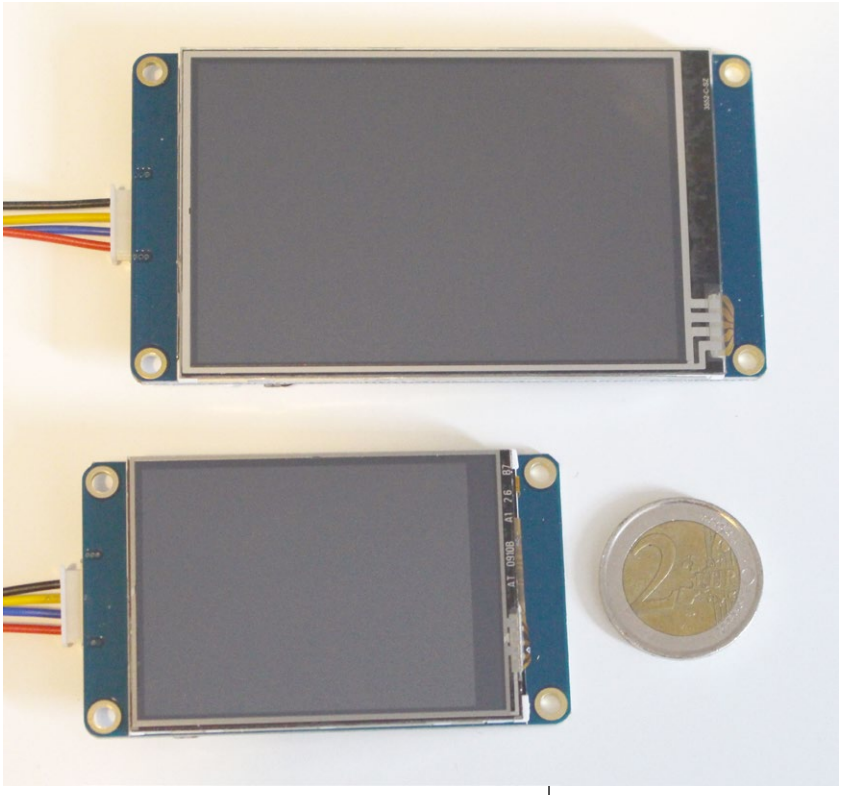


Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)

La mode est à l'intelligence, surtout côté électronique (pour le reste je vous laisse seuls juges). Après les leds intelligentes intégrant un circuit intégré permettant de définir simplement les couleurs qu'elles doivent prendre, et ce quel que soit leur nombre, voici venir une nouvelle génération d'écrans tactiles pour vos montages Arduino ou Raspberry Pi, conçus avec la même notion de facilité d'utilisation.

Un écran LCD ou OLED utilisable pour vos projets se résume généralement à un simple système d'affichage, entièrement contrôlé par vos programmes ou croquis. Vous devez donc utiliser, en plus des bibliothèques permettant le dialogue, des fonctions vous fournissant des primitives graphiques : contrôle des pixels, tracés de formes géométriques, écriture de texte, etc. Ceci représente une charge non négligeable de travail et une consommation de ressources mémoire généralement tout aussi importante.

Pour dessiner un simple écran affichant, par exemple, une jauge ou une barre de progression présentant une température accompagnée de la valeur numérique associée, vous devez donc tout faire vous-même, dessiner chaque élément avec des rectangles, des lignes, des courbes... Ne serait-il pas plus agréable de laisser faire tout cela par l'écran lui-même, vous concentrant uniquement, au niveau de votre croquis, sur l'envoi et la réception des informations et non leur présentation ? C'est là précisément ce que proposent les écrans Nextion de ITEAD.



Les deux modèles utilisés pour les expérimentations de l'article sont le NX4832T035_011R (3,5") et le NX3224T024_011R (2,4"). De taille relativement modeste, ils sont parfaitement adaptés pour des projets à base de carte Arduino, même s'ils peuvent également être utilisés avec tous types de plateformes, y compris un PC ou un Mac.

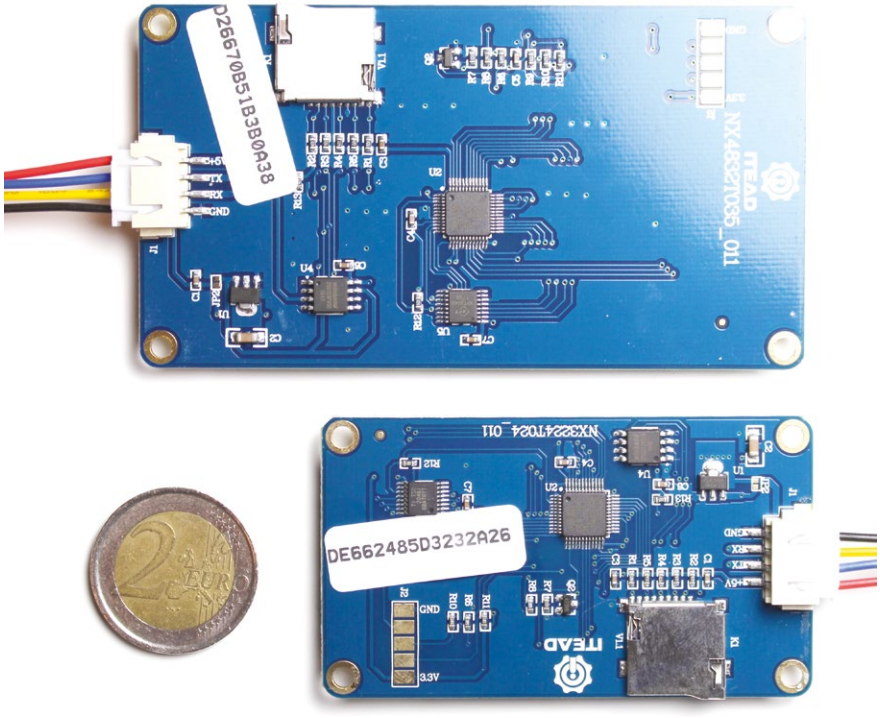
1. LES ÉCRANS NEXTION

Plus que de simples écrans, ces matériels sont décrits comme une solution d'Interface Homme-Machines (IHM, ou en anglais HMI pour *Human Machine Interface*). Il ne s'agit pas simplement de présenter des données à l'écran, mais également de gérer cette présentation et une partie des interactions (pour les modèles d'écran ayant une surface tactile). Ces écrans se chargent donc, de façon autonome, d'afficher une interface graphique stockée dans leur mémoire et de n'échanger, avec le système connecté, comme une carte Arduino, que les informations utiles. Il peut s'agir d'un clic sur un bouton, la position d'une barre de progression, le texte à afficher, etc. Tout ce qui peut être fait par l'écran lui-même sera géré de cette façon et le microcontrôleur de votre Arduino, et donc votre croquis, n'aura plus qu'à se concentrer sur les données, mais non sur la manière de les présenter.

Pour arriver à un tel mode de fonctionnement, la logique consiste à concevoir entièrement l'interface, qu'elle soit simple ou complexe, à l'aide

Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)

ÉCRAN LCD



Les écrans Nextion d'ITEAD ne sont pas de simples systèmes d'affichage, mais sont programmés, via une carte microSD, pour accueillir des interfaces complètes (bouton, barre de progression, jauges, images, etc.) communiquant avec vos projets par une liaison série.

d'un logiciel dédié. Cette interface sera compilée par le logiciel et chargée dans la mémoire de l'écran via une carte microSD (il est également possible d'utiliser une connexion série, mais ceci implique l'utilisation d'un convertisseur USB/série 5V). Une fois la copie réalisée, l'écran affichera cette interface, même sans être connecté au système qui le contrôle. Il peut s'agir d'une carte Arduino, une Raspberry Pi, mais aussi de tout autre type de systèmes capables de communiquer via une liaison série.

Ces écrans sont donc des systèmes complets, possédant leur propre mémoire et leur propre microcontrôleur (STM32), programmés via un transfert par carte microSD. Il ne s'agit plus de simples écrans comme ceux construits autour des contrôleurs ILI9340, PCF8833, SSD1351 ou tout autre modèle dont nous avons précédemment parlé dans le magazine (voir *Hackable n°3 et 4*) et pouvant être utilisés, par exemple, avec les bibliothèques *Adafruit_GFX* ou *U8glib* (voir *Hackable n°9*).

Les différents modèles d'écrans Nextion sont officiellement au nombre de 16 (d'après la version actuelle du logiciel) et se répartissent en deux catégories : « Basic » (basique) et « Enhanced » (étendu). Les modèles étendus disposent de plus de mémoire et de ressources processeur, intègrent une horloge temps réel (RTC), des entrées/sorties (GPIO) et une fonction de sauvegarde en mémoire flash (support de stockage). Ils sont aussi plus coûteux et présentent surtout l'avantage de pouvoir être utilisés de façon totalement autonome. Nous nous concentrerons cependant ici sur les modèles basiques, plus économiques qui se distinguent entre eux par leur taille et leur résolution :

- NX3224T024_011R : 2,4", 320×240 pixels, 4 Mo de flash ;
- NX3224T028_011R : 2,8", 320×240 pixels, 4 Mo de flash ;
- NX4024T032_011R : 3,2", 400×240 pixels, 4 Mo de flash ;
- NX4832T035_011R : 3,5", 480×320 pixels, 16 Mo de flash ;
- NX4827T043_011R : 4,3", 480×272 pixels, 16 Mo de flash ;
- NX8048T050_011R : 5", 800×480 pixels, 16 Mo de flash ;
- NX8048T070_011R : 7", 800×480 pixels, 16 Mo de flash ;
- NX8048T090_011R : 9", 800×480 pixels, 16 Mo de flash.

Tous les modèles affichent en 65536 couleurs, disposent de 3584 octets de mémoire vive et s'utilisent de la même façon, avec la même interface série, le même logiciel de création d'interfaces (Windows uniquement) et les mêmes bibliothèques Arduino pour

Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)

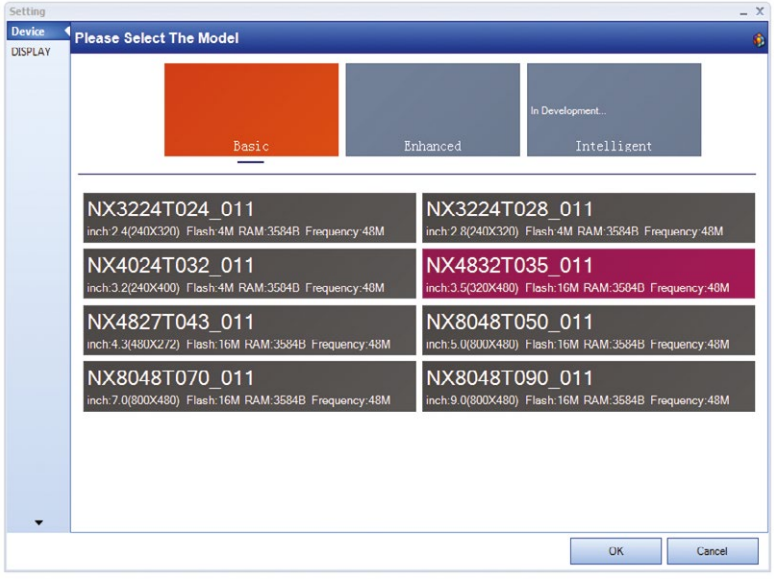
le pilotage du périphérique. Les deux modèles sélectionnés pour cet article sont NX4832T035_011R (3,5") et NX3224T024_011R (2,4"). Ils ont été acquis sur eBay, respectivement auprès des vendeurs de Shenzhen, « elec_mall2012 » et « h-quality_electronic », au prix de 27€ et 14€ (port gratuit dans les deux cas). Je n'ai personnellement pas trop d'intérêt pour les modèles d'une taille supérieure qui affichent généralement des prix assez importants de l'ordre de 60€ pour le 5" ou 75€ pour le 7" (le modèle NX8048T090_011R semble introuvable pour l'instant, sans doute un futur modèle présent uniquement dans le logiciel). À ce prix-là, autant utiliser une Raspberry Pi et un écran HDMI, DSI ou DPI, sachant, de plus, que les projets Arduino sont souvent assez compacts par eux-mêmes.

2. CRÉER VOTRE INTERFACE

Vous l'avez compris avant même d'envisager la connexion d'un écran Nextion à une carte Arduino, il est nécessaire de construire son interface graphique. Pour cela, la seule solution actuellement disponible (à ma connaissance) consiste à télécharger le *Nextion Editor* sur le site d'ITEAD : **https://nextion.itead.cc/download.html**. La dernière version en date de cette application est la 0.47, mais il est fort probable qu'au moment où vous lirez ceci une nouvelle version soit disponible, avec davantage de fonctionnalités. À cette date, seule une version en langue anglaise est disponible.

Il s'agit d'une application propriétaire (les sources ne sont pas diffusées), Windows s'installant tout à fait classiquement et aucun pilote n'est nécessaire. En effet, le transfert de l'interface dans l'écran se fait sous la forme d'un fichier, produit par le logiciel, copié sur une microSD formatée en FAT32 (important). Celle-ci sera alors tout simplement placée dans l'emplacement réservé à cet effet à l'arrière de l'écran et lors de la prochaine mise sous tension, son contenu est copié dans la mémoire flash. La microSD peut alors être retirée et l'écran réinitialisé par un nouveau cycle d'arrêt/démarrage.

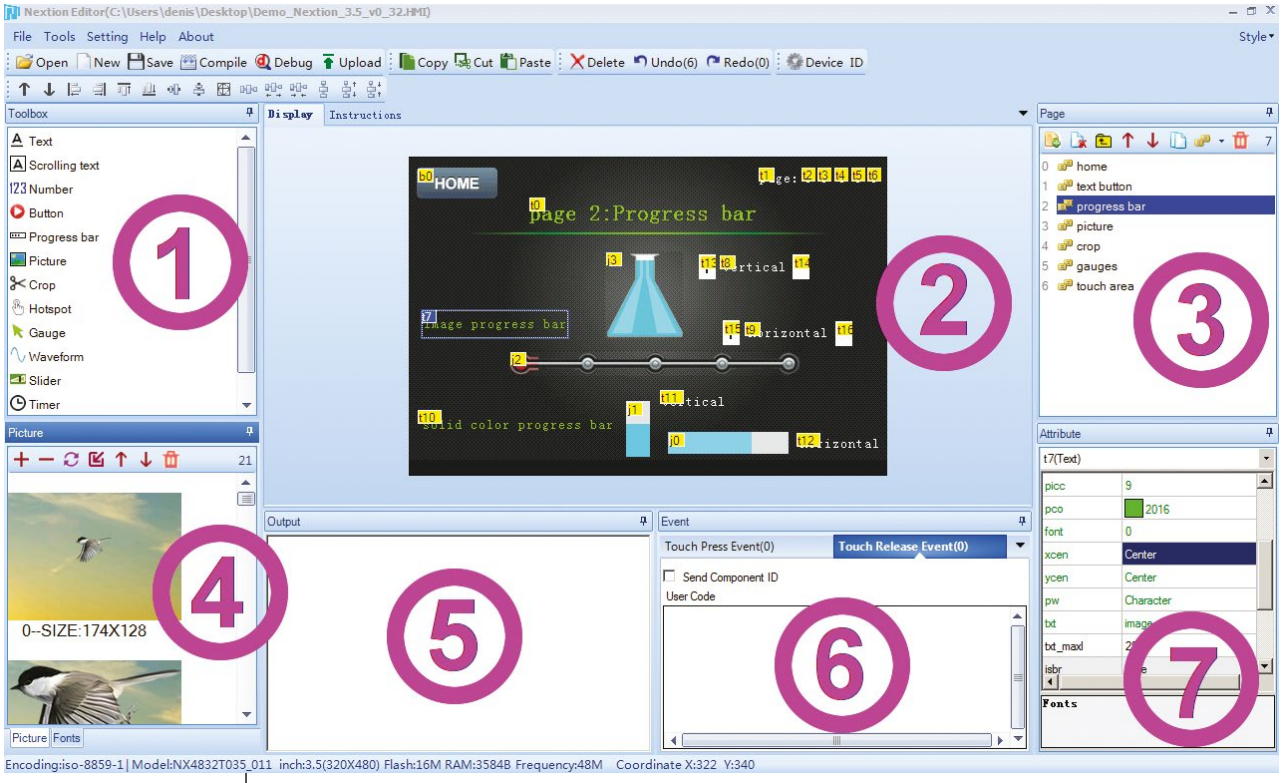
Au lancement de l'application fraîchement installée, l'interface qui se présente est assez simple, mais en anglais. Pour démarrer un nouveau projet, il suffit de cliquer sur ***New*** dans la barre supérieure et de spécifier un nom de fichier qui prendra une extension **`.HMI`**. On vous demandera ensuite de préciser le modèle d'écran pour lequel vous souhaitez composer une interface. Il aurait été plus logique de proposer ce choix avant le nommage du projet, mais en cas d'annulation, rien n'est sauvegardé. Vous ne risquez donc pas de créer des fichiers HMI inutiles de cette façon. Sachez qu'il vous est également possible de changer ce choix de modèle par la suite en utilisant le bouton ***Device ID***.



La sélection du modèle d'écran utilise les désignations qui se trouvent à l'arrière des modules et est répartie dans une famille de modèles. Sur la gauche, « Device » permet de choisir le type et « DISPLAY » l'orientation à utiliser par incrément de 90°. Ceci n'est pas très logique et si on ne le sait pas à l'avance, c'est là quelque chose qui peut prendre vraiment très longtemps à retrouver...

Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)

ÉCRAN LCD



Nextion Editor est l'application Windows vous permettant de composer une interface pour les écrans Nextion. Ce n'est pas un modèle d'ergonomie et certaines fenêtres ont encore des sinogrammes par endroit. Malheureusement, c'est le seul logiciel permettant de créer des interfaces pour l'instant, et son fonctionnement interne, tout comme le format des fichiers utilisés, n'est pas documenté.

L'interface par défaut se divise en plusieurs parties comme le montre l'illustration présentée ici. La zone centrale (2) est une représentation de votre future interface à l'échelle et à la taille de l'écran que vous avez sélectionné (il n'y a pas de fonction « zoom »). Sur la gauche (1) se trouve votre boite à outils regroupant tous les objets (ou composants) que vous pouvez utiliser. Ceux-ci sont ajoutés automatiquement dès que vous cliquez dessus et peuvent être déplacés et modifiés ensuite. Il n'y a pas de cliquer-déposer, c'est un peu déroutant au début, mais on s'y fait...

Toujours à gauche de l'interface, une section nommée ***Picture*** (4) présente les images ainsi que les polices utilisables (onglet en bas). La logique ici consiste à assembler une collection d'images ou polices pour les utiliser ensuite, par référence, dans la configuration des objets utilisés. En haut de cette section, les boutons **+** et **-** permettent d'ajouter les éléments sous la forme de fichiers et les supprimer. Une fois une importation réalisée, le nom de l'image n'est plus important, mais sa position dans la liste sert alors d'index. Ce numéro est présenté sous chaque image, accompagné de sa taille, et c'est celui que vous devrez utiliser dans l'attribut de remplissage des objets (*picc*, *bpic*, *ppic*, etc.).

L'onglet ***Fonts*** fonctionne de la même manière, mais avec des fichiers de polices générés par l'application. Il vous est possible d'utiliser n'importe quelle police présente dans votre système. En utilisant le menu ***Tools*** et ***Font Generator***, vous choisirez une police et une taille puis générez un fichier après avoir spécifié un nom. Vous pourrez alors enregistrer cette police sous la forme d'un fichier `.zi` et aurez l'opportunité de l'intégrer directement à votre composition. Vous pouvez aussi sauter cette étape pour ensuite utiliser le bouton **+** et sélectionner le fichier `zi` à charger. Notez qu'un fichier `.zi` ne contient qu'une police à une taille donnée. Si vous utilisez plusieurs tailles, il vous faudra générer, intégrer et utiliser plusieurs fichiers.

Comprenez bien ici qu'il faut voir les images et les polices

Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)

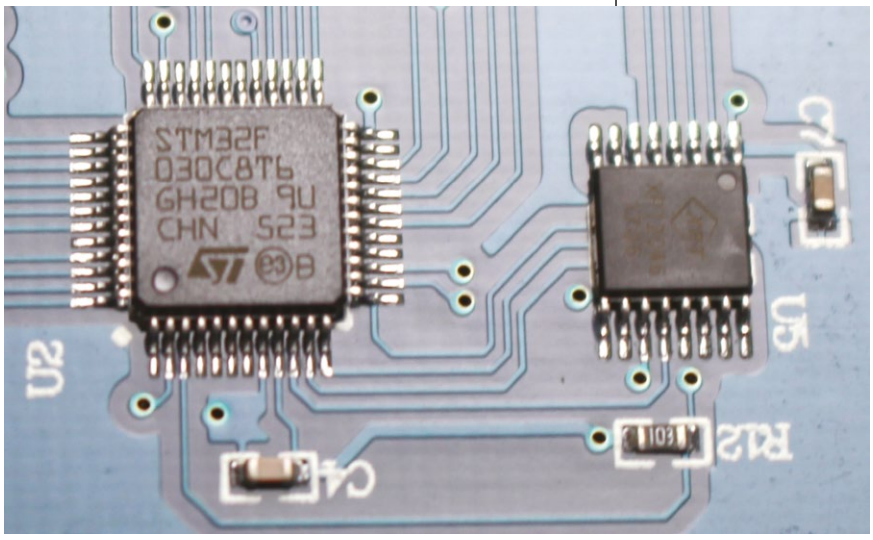
comme des ressources qui seront embarquées dans votre interface, qu'elles soient utilisées ou non. Chacune de ces entrées occupe donc un certain espace dans la mémoire flash de l'écran qui, selon le modèle, est limité à 4 ou 16 Mo. Après importation, il ne s'agit plus de fichiers, mais de blocs de données sans nom, seuls les numéros sont importants.

Sur la droite vous trouverez une section ***Page*** (3) regroupant les pages ou écrans utilisables. Une interface est toujours composée au minimum d'une page sur laquelle trouvent place les objets (texte, image, barre de progression, jauges, etc.). Vous pouvez avoir autant de pages que vous le souhaitez de façon à présenter, depuis votre croquis différents écrans. Il est également possible de gérer cela de façon indépendante du croquis Arduino, mais nous verrons cela une autre fois. Chaque page peut avoir le nom qui vous plaira, il vous suffit de double-cliquer et d'en spécifier un autre. Une barre de boutons vous permet d'ajouter, supprimer, insérer, changer la position, dupliquer, importer et exporter des pages. On notera ici que les boutons en question sont totalement différents de la section images/polices précédente... Dans le doute, survolez simplement un bouton avec le pointeur de la souris pour avoir une bulle précisant de quoi il s'agit (en anglais bien sûr).

À droite en bas, se trouvent les attributs de l'objet actuellement sélectionné. Ceux-ci changent en fonction du type d'objet, mais de façon générale, on retrouve :

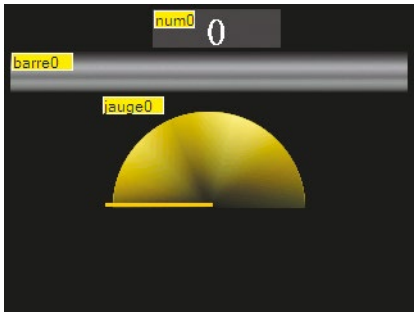
- « id » : le numéro unique de l'objet permettant de l'identifier ;
- « objname » : le nom de l'objet que vous pouvez changer ;
- « type » : le type numérique de l'objet ;
- « vscope » : la portée de la variable. Ceci permet de faire en sorte que les variables des objets puissent être accessibles ou non depuis d'autres pages ;
- « sta » : le type de remplissage souhaité, au choix, entre une couleur et une image ;
- « x » : la position horizontale du coin supérieur gauche de l'objet ;
- « y » : la position verticale même coin ;
- « w » : la largeur de l'objet en pixels ;
- « h » : et sa hauteur.

Les sections en bas de l'écran (5 et 6) présentent respectivement les messages générés lors de la compilation de l'interface et les événements qui doivent être générés par l'objet sélectionné à différents moments dans la gestion de l'interface et ses interactions avec un utilisateur (avant initialisation, après initialisation, lors d'une pression, lors d'un relâchement, etc.). Un tel événement peut déclencher une action sur un paramètre d'un autre objet, comme le pourcentage de progression d'une barre, ou tout simplement provoquer



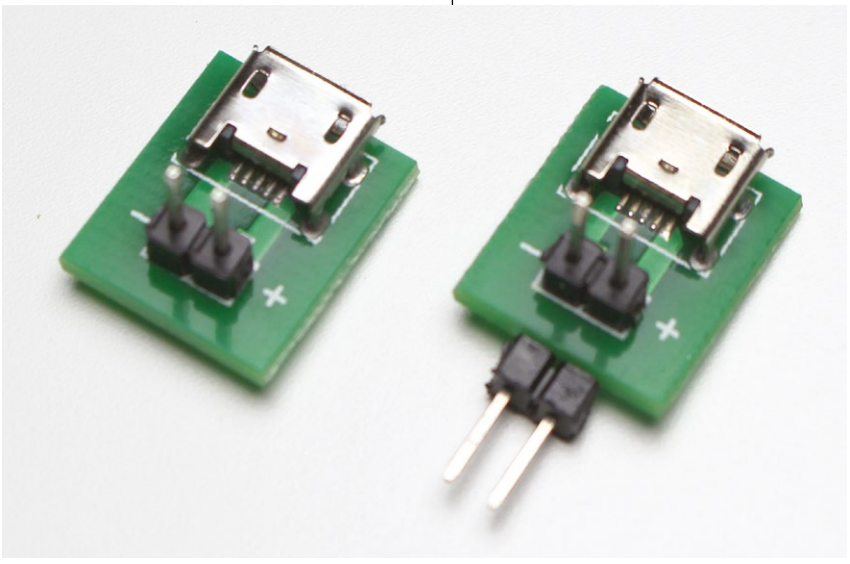
Au cœur d'un écran Nextion se trouve un microcontrôleur STM32 (à gauche) chargé de présenter l'interface utilisateur conçue avec une application spécifique. En plus de l'écran LCD, ces modules disposent d'une mémoire flash pour le stockage de l'interface et d'un contrôleur de surface tactile résistive (à droite).

Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)



L'interface conçue dans le logiciel d'ITEAD consiste en un agencement et une configuration de composants graphiques interactifs. Chaque composant est nommé et se voit paramétré par un ensemble d'attributs (taille, couleur, fond, valeur par défaut, etc.). Votre croquis Arduino ne fera qu'interagir avec ces composants et sera donc totalement déchargé de leur gestion ou de leur affichage.

Les écrans Nextion sont livrés avec la connectique nécessaire et un adaptateur permettant de les alimenter avec un bloc secteur identique à celui utilisé pour une Raspberry Pi. Petit problème cependant, les masses de l'écran et de la carte Arduino doivent être reliées sous peine de voir apparaître de gros problème de communication. L'une des solutions possibles consiste à tout simplement souder un connecteur supplémentaire pour avoir deux masses et deux broches +5V.



la communication du numéro de l'objet via la liaison série. Nous verrons tout cela via un exemple un peu plus loin...

Notez qu'il est possible de personnaliser l'interface du logiciel, de déplacer des sections et même de les combiner sous forme d'onglets afin de gagner un peu de place pour la zone principale. Le système est un peu « tordu », mais il suffit de prendre la barre de titre d'une des zones et la glisser quelque part sur l'écran. Des icônes en croix apparaissent alors au survol des zones, permettant d'ajouter en haut, en bas, à gauche ou à droite de la zone en question. Le centre de la croix permet un ajout sous forme d'onglet à la zone survolée.

La prise en main de cette interface vous demandera certainement un peu de temps, car c'est assez peu intuitif, mais une fois la logique assimilée vous pourrez travailler assez rapidement. C'est davantage une question d'habitude qu'un réel problème de conception de l'application, même si elle n'en est pas exempte, loin de là.

3. INSTALLEZ LES BIBLIOTHÈQUES

Deux bibliothèques Arduino permettent l'utilisation d'une interface chargée dans la mémoire d'un écran Nextion. Nous avons en premier lieu celle directement installable depuis le gestionnaire de bibliothèques : *NeoNextion*. Celle-ci n'est pas la bibliothèque officielle d'ITEAD, mais une version modifiée et étendue par Dan Nixon. Bien que toujours maintenue, elle ne prendra pas en charge de futurs modèles d'écrans car, comme le dit Dan lui-même, celui-ci n'utilise plus d'écran de ce type pour ses projets. Les corrections et ajouts de fonctionnalités sont cependant toujours d'actualité sur les modèles qu'il possède. La dernière version stable est la 2.2.0 d'août 2016.

L'autre bibliothèque est celle proposée directement par le fabricant des écrans, mais celle-ci n'est pas présente dans le gestionnaire. Il vous faudra donc la télécharger directement depuis GitHub à l'adresse **https://github.com/itead/ITEADLIB_Arduino_Nextion/releases**, puis l'installer dans le sous-répertoire `libraries` de votre carnet de croquis, après désarchivage/décompression. La dernière version stable est la 0.7.0, celle-ci est relativement ancienne (août 2015) et fait l'impasse sur bon nombre de fonctionnalités maintenant présentes dans l'éditeur. Mieux vaudra donc utiliser la

Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)

version de développement directement téléchargeable à la racine du dépôt GitHub sous forme de fichier ZIP et l'installer de la même manière.

Les deux bibliothèques ont une approche sensiblement différente, mais il est relativement aisé de porter un croquis de l'une vers l'autre. Le principal avantage de la bibliothèque de Dan est sa capacité à utiliser un port série de son choix. En effet, la bibliothèque d'ITEAD est initialement prévue pour travailler facilement sur Arduino Mega2560 et non sur UNO. Deux ports série sont utilisés par défaut, un pour dialoguer avec l'écran (`Serial2`) et un pour les informations de mise au point (`Serial`). Le problème avec l'Arduino UNO est, bien entendu, le fait que cette carte ne possède **que** `Serial`.

Il faut donc en principe, suivant les recommandations de la documentation ITEAD, éditer le fichier `NexConfig.h` pour changer la définition des macros `dbSerial` (le moniteur) et `nexSerial` (pour l'écran). Une telle modification est relativement pénible, car normalement on n'édite pas directement une bibliothèque de cette façon en fonction de la carte utilisée.

J'ai donc décidé de pousser la modification plus loin pour directement me libérer de cette contrainte. Le fichier `NexConfig.h` est effectivement modifié, mais afin de commenter toute la ligne « `#define nexSerial Serial2` » en la faisant précéder de `//`. Ainsi, `nexSerial` n'est plus défini du tout. On se tourne ensuite vers le fichier `NexHardware.h`, qui est le fichier d'entête du code qui gère la communication avec le matériel (`NexHardware.cpp`), et on y ajoute :

```
#include <SoftwareSerial.h>
extern SoftwareSerial nexSerial;
```

Cette modification a pour effet de dire au compilateur qu'un objet `nexSerial` de type `SoftwareSerial` existe quelque part (`extern`) dans le code, mais ce n'est pas une déclaration à proprement parler. Tout ce que nous aurons à faire dans notre croquis sera de déclarer `nexSerial` avant toute utilisation de la bibliothèque (et `nexInit()` en particulier) :

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

SoftwareSerial nexSerial(4, 5); // RX, TX
```

Cette modification nous autorise alors à utiliser la bibliothèque *SoftwareSerial* permettant de spécifier un port série géré de façon logicielle en utilisant deux broches arbitrairement choisies sur la carte Arduino (ici 4 et 5) pour communiquer avec l'écran. Nous ne sommes plus coincés et pouvons utiliser `Serial`, le port physique relié au moniteur série, de façon tout à fait classique.

Note : Si vous souhaitez ne pas voir de messages de la bibliothèque s'afficher sur le moniteur série, vous pouvez également commenter la ligne « `#define DEBUG_SERIAL_ENABLE` » dans `NexConfig.h`, et donc avoir la totale maîtrise de `Serial` à la vitesse de votre choix (9600 bps dans le cas contraire). C'est bien entendu ce que j'ai fait.

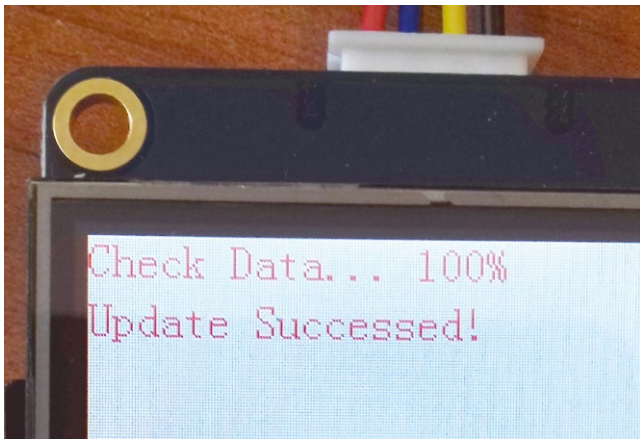
Notez au passage que le fait d'installer les deux bibliothèques est une mauvaise idée. Les deux partageant un fichier nommé `Nextion.h`, ceci peut créer une confusion au niveau de la compilation (le `Nextion.h` de *NeoNextion* étant utilisé avec la bibliothèque d'ITEAD). Je ne comprends pas pourquoi ce choix a été fait par Dan plutôt que de simplement utiliser le nom de sa bibliothèque.

4. PREMIER EXEMPLE : L'ÉCRAN COMME AFFICHEUR

Appréhender toutes les fonctions disponibles grâce à ces écrans n'est pas vraiment possible en un unique article, en particulier si je devais passer tous les types de composants en revue. Le plus simple est donc de créer un projet simple sur lequel vous pourrez construire vos expérimentations à venir.

Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)

ÉCRAN LCD



Pour transférer une interface dans un écran Nextion, il suffit de copier un fichier sur une carte microSD formatée en FAT32, de la glisser dans l'emplacement à l'arrière de l'écran et de mettre ce dernier sous tension. Automatiquement l'interface sera copiée dans la mémoire de l'écran et la carte ne sera plus nécessaire.

Notre première réalisation consistera donc à afficher des données. La source sera un simple potentiomètre relié entre la masse et la tension d'alimentation de la carte Arduino ainsi que, pour la partie « mobile », à l'entrée analogique A0. Ceci nous fournira une lecture sous la forme d'une valeur entre 0 et 1023 via **`analogRead()`**.

Notez qu'il est impératif d'alimenter correctement. L'alimentation type micro USB doit être capable de fournir un courant de l'ordre de 1,5A en 5V. Un connecteur est livré avec l'écran, mais il sera également important de mettre les masses en commun (Arduino/écran), ce qui nécessite généralement une petite modification du connecteur ou l'utilisation d'une platine à essais.

Pour afficher cette valeur sur l'écran Nextion, nous opterons pour une seule page (**`0`**) sur laquelle nous plaçons (de haut en bas) :

- Un composant *Number* avec l'identifiant **`1`** et le nom **`num0`**. On définira de simples couleurs de premier et arrière-plan, ainsi qu'une police importée pour l'occasion.
- Un composant *Progress bar*, identifiant **`3`** et le nom **`barre0`**. Ici, nous pouvons choisir un remplissage par image (*sta* = *image*) plutôt qu'avec une couleur. La technique consiste à importer deux images de la taille de la barre, une représentant la barre totalement inactive et l'autre totalement active. La valeur de la barre sera utilisée, verticalement ou horizontalement, pour définir la proportion de l'image d'avant plan (*ppic*) recouvrant celle d'arrière-plan (*bpic*). Ceci permet de créer des barres de progression très travaillées graphiquement.
- Un composant *Gauge*, identifié par **`2`** et nommé **`jauge0`**. Là encore, une image est utilisée comme arrière-plan (*sta* = *crop image*) et on choisira simplement la couleur et l'épaisseur de l'aiguille. L'image d'arrière-plan doit être de la taille de l'écran et celle-ci est rendue visible qu'à l'arrière du composant, comme un masque ou un pochoir. Il faut donc créer une image ayant le fond du cadran à la position exacte où sera placée la jauge dans l'interface.

Une fois tous ces composants placés et configurés dans l'éditeur, il est possible de tester l'interface en utilisant le bouton ***Debug***. Ceci aura pour effet d'ouvrir une nouvelle fenêtre présentant l'interface telle qu'elle se présentera sur le matériel. Cette simulation provoque la compilation de l'interface, mais vous pouvez aussi utiliser le bouton ***Compile*** pour faire de même. On passera ensuite par le menu ***File*** et ***Open build folder*** pour ouvrir le répertoire contenant le fichier compilé, avec l'extension **`.tft`**, dans le gestionnaire de fichiers et on le copiera sur une carte microSD.

Celle-ci doit ensuite être glissée dans l'emplacement à l'arrière de l'écran et celui-ci est mis sous tension. Un message précisant que la carte a été détectée est alors affiché et un délai de quelques secondes est décompté avant la copie automatique dans la mémoire flash. Une fois l'opération terminée, on peut mettre l'écran hors tension et retirer la carte microSD. L'interface est maintenant chargée et prête à être utilisée.

On se tourne alors vers le croquis Arduino :

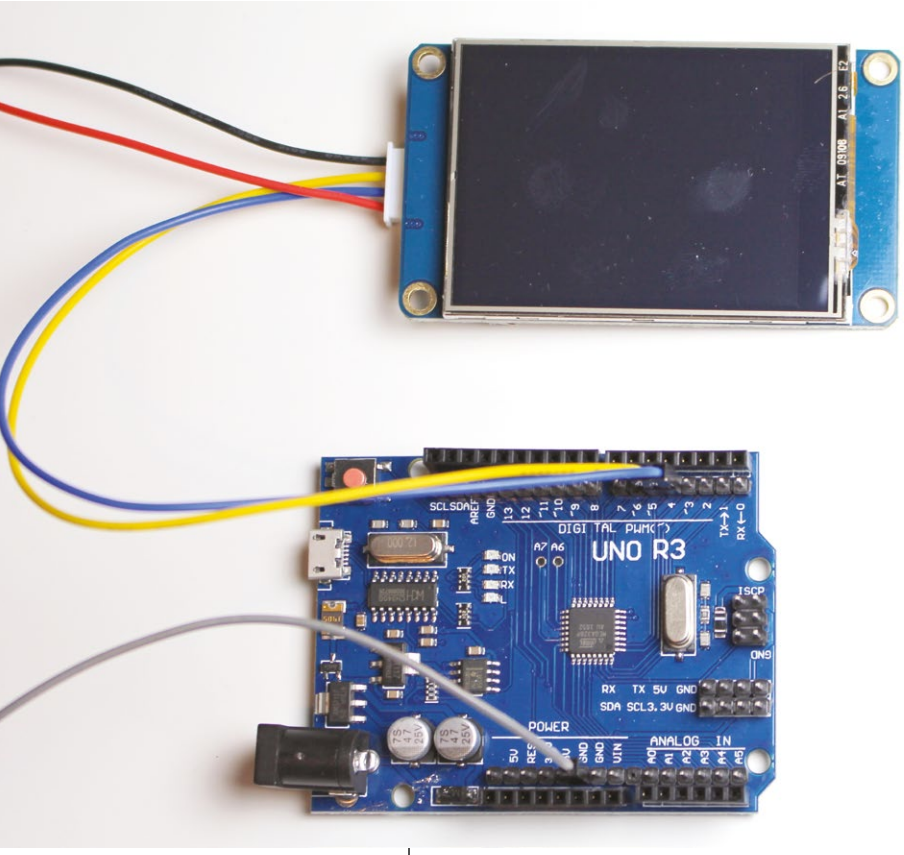
Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)

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

// port série sur broches 4 et 5
SoftwareSerial nexSerial(4, 5); // RX, TX

// composants de l'interface
NexNumber num0         = NexNumber(0, 1, "num0");
NexGauge jauge0        = NexGauge(0, 2, "jauge0");
NexProgressBar barre0 = NexProgressBar(0, 3, "barre0");

// configuration
void setup(void) {
  Serial.begin(115200);
  // cette fonction retourne vrai si l'écran répond
  if(nexInit()) {
    Serial.println("setup ok");
  } else {
    Serial.println("setup error !!!");
    while(1){;}
  }
}

// valeur précédente
int pval = 0;

// boucle principale
void loop(void) {
  // lecture analogique et transposition de valeurs
  int val = map(analogRead(A0),0,1023,0,180);
  // on ne change l'interface que si la valeur change
  if(val != pval) {
    // changement de la barre de progression
    barre0.setValue(map(val,0,180,0,100));
    // changement de la jauge
    jauge0.setValue(val);
    // changement du nombre affiché
    num0.setValue(val);
    // mise à jour
    pval=val;
  }
  // pause
  delay(20);
}
```

J'utilise ici la bibliothèque originale modifiée par mes soins, c'est pourquoi il est nécessaire de déclarer `nexSerial` en tout début de croquis afin d'utiliser les broches 4 et 5 en guise de port série logiciel.

Pour utiliser les composants définis dans l'interface, on commence par déclarer des variables les représentant dans notre code. Les constructeurs prennent en argument l'identifiant de la page sur laquelle ils se trouvent, leur propre identifiant ainsi que leur nom. Le croquis, assisté de la bibliothèque d'ITEAD, ne crée pas les composants, mais utilise ces informations pour la communication série avec l'écran.

Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)



Ces écrans se pilotent à l'aide d'une simple liaison série à 9600 bps. Une petite modification de la bibliothèque Arduino officielle est nécessaire pour faciliter les choses avec une carte Arduino UNO ne disposant que d'un seul port de ce type, déjà relié au moniteur série via USB. Cette modification consiste à utiliser SotfSerial et donc permet de choisir arbitrairement les broches pour connecter l'écran.

La fonction `setup()` initialise la communication avec le moniteur série ainsi que celle avec l'écran (`nexInit()`) et teste un éventuel silence de sa part. Tout se passe dans `loop()` où nous mesurons la tension sur la broche A0 et réutilisons cette valeur, mise à l'échelle (`map()`), pour déterminer la valeur de chaque composant, via leur méthode `setValue()`. Afin de minimiser la quantité de données circulant, nous ne mettons à jour ces valeurs qu'en cas de changement. Ceci n'est pas important pour la valeur numérique affichée ou la barre de progression, mais évite un scintillement des plus désagréables sur la jauge.

La valeur des composants utilisés n'est pas le seul attribut qu'il vous est possible de changer depuis votre croquis. Vous pourrez trouver une documentation complète de chaque méthode pour chaque type de composant directement dans le répertoire de la bibliothèque. Il vous suffit d'ouvrir, avec votre navigateur, le fichier `index.html` se trouvant dans le sous-répertoire `doc/Documentation`.

Là, parcourez *Class* et *Class List* pour obtenir une liste de types correspondant, en principe, aux composants utilisables dans la dernière version de l'éditeur d'interfaces.

5. SECOND EXEMPLE : UN ÉCRAN POUR CONTRÔLER UN MONTAGE

Même si la présentation de données peut être très intéressante en comparaison à ce qu'il est possible de faire avec un simple afficheur TFT, la véritable richesse des écrans Nextion s'exprime surtout dans l'utilisation inverse : réaliser une interface pour piloter un montage. Il est possible de composer une interface avec toutes sortes d'éléments : réglettes, boutons, cases à cocher, etc., permettant de remplacer les boutons physiques, les claviers, les potentiomètres et les interrupteurs généralement utilisés.

Pour cette réalisation, nous allons conserver deux éléments de l'interface précédente, la barre de progression et la valeur numérique affichée, mais allons supprimer la jauge et la remplacer par :

- deux boutons « b0 » et « b1 » possédant respectivement les identifiants 3 et 4 et marqués des textes « - » et « + » ;
- un *slider* (réglette/glissière) permettant de régler une valeur entre 0 et 255 (« maxval »).

Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)

Il est important de cocher, pour ces trois éléments, la case ***Send Component ID*** sur l'onglet ***Touch Release Event*** dans la partie ***Event*** de l'éditeur. Ceci aura pour effet d'envoyer un message contenant l'identifiant du composant lors du « relâché », via la liaison série entre l'écran et l'Arduino. Ceci constituera un événement qui sera pris en compte dans notre croquis.

Côté Arduino, nous nous contenterons de connecter une led et sa résistance de 470 ohms, entre la broche 6 et la masse. L'intensité lumineuse de la led sera l'élément contrôlé en PWM par l'interface.

Commençons par la fin cette fois puisque la fonction **loop()** ne saurait être plus simple :

```
void loop(void) {
  nexLoop(nex_listen_list);
}
```

Celle-ci ne fait qu'appeler à répétition **nexLoop()** chargée de gérer les événements et prenant en argument une liste de composants « à écouter ». Une liste qui est définie en début de croquis :

```
NexTouch *nex_listen_list[] =
{
    &b0,
    &b1,
    &h0,
    NULL
};
```

Nous trouvons ici trois pointeurs (caractère **&**) vers les objets, dans le croquis, qui correspondent aux composants pouvant générer des événements :

```
NexNumber num0        = NexNumber(0, 1, "num0");
NexProgressBar barre0 = NexProgressBar(0, 3, "barre0");
NexButton b0          = NexButton(0, 3, "b0");
NexButton b1          = NexButton(0, 4, "b1");
NexSlider h0          = NexSlider(0, 5, "h0");
```

nexLoop() va être à l'écoute du port série et capter tout message signalant une action sur ces composants. Il dispatchera ensuite les événements sous la forme d'appels à ces fonctions *callback*, attachées dans la fonction **setup()** :

```
// configuration
void setup(void) {
  pinMode(6, OUTPUT);
  Serial.begin(115200);
  // cette fonction retourne vrai si l'écran répond
  if(nexInit()) {
    Serial.println("setup ok");
  } else {
    Serial.println("setup error !!!");
    while(1){;}
  }
  // configuration des fonctions à appeler
  b0.attachPop(b0Callback);
  b1.attachPop(b1Callback);
  h0.attachPop(h0Callback);
}
```

Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)

Ceci étant fait, nous n'avons plus qu'à écrire les fonctions en question, toutes trois assez similaires. Voici par exemple la première, appelée automatiquement lorsque le premier bouton sur l'interface est relâché :

```
// fonction lancée lors du relâchement de b0
void b0Callback(void *ptr) {
  uint32_t val;
  num0.getValue(&val);
  if (val >= 5)
    val -= 5;
  else
    val = 0;
  barre0.setValue(map(val,0,255,0,100));
  num0.setValue(val);
  h0.setValue(val);
  analogWrite(6, val);
}
```

Celle-ci va lire la valeur actuelle du composant **num0** (la valeur numérique affichée à l'écran) et, sous condition, la décrémenter avant de mettre à jour cette même valeur, en retour, sur l'écran, mais également en impactant l'intensité de la led.

Comme vous pouvez le voir, le croquis se limite à l'écriture des fonctions *callback* et il devient possible de disposer d'une interface travaillée tout en se concentrant presque uniquement sur l'aspect fonctionnel côté Arduino. Il vous sera possible d'utiliser des images pour vos interfaces, exactement comme dans l'exemple précédent et même de permettre la saisie de valeurs et de texte. En jouant avec les composants et leur configuration, il deviendra un jeu d'enfant, par exemple, de définir une couleur pour un ruban de leds WS2812b ou encore d'ajuster la position d'un servomoteur.

6. TOUT N'EST PAS PARFAIT, LOIN DE LÀ

Avant toutes choses, précisons que malgré les critiques que je vais faire, les écrans Nextion permettant de finaliser rapidement des projets qu'il aurait été autrement plus difficile à réaliser à l'aide d'un simple afficheur ou écran LCD TFT standard. Le simple cas de la barre de progression texturée ou de la jauge représenterait un travail considérable du point de vue graphique. Les écrans Nextion apportent donc effectivement une solution très intéressante pour créer des interfaces élaborées et de qualité. Mais...

Il serait presque malhonnête d'affirmer que l'éditeur d'interfaces est une application finie. Le numéro de version lui-même, 0.47, est relativement explicite et le fait que son développement soit entièrement du fait d'ITEAD ne permet pas une évolution rapide. L'ergonomie générale est assez perturbante sur bien des aspects. À titre d'exemple et en plus des points déjà abordés dans cet article, ce sont surtout des petites choses qu'on prend généralement pour acquises qui manquent à l'appel : une carence en homogénéité dans la présentation des options, le fait de ne pouvoir ouvrir qu'un projet à la fois, l'absence de support de différentes langues, l'absence de guides, de grilles ou de tout autre système d'alignement digne de ce nom, l'impossibilité de cliquer-glisser, l'absence de zoom pour un placement des composants « au pixel près », etc.

En termes de fonctionnalités des écrans eux-mêmes, qui sont liées par définition avec l'éditeur et les bibliothèques, on regrettera également quelques points. C'est le cas, par exemple, de l'absence de gestion de la transparence et, quitte à ajouter la fonctionnalité, une transparence ajustable au même titre que d'autres attributs des composants. Il pourrait être très intéressant de pouvoir faire apparaître un composant en fonction d'une valeur *alpha* comme avec un logiciel de retouche d'images ou des bibliothèques graphiques comme il en existe par ailleurs (Evas, EFL, Cairo, SDL, etc.).

De nouveaux types de composants sont ajoutés tantôt au fil de l'évolution des versions de l'éditeur. Il serait ainsi très appréciable de voir ajouter quelque chose tenant du croisement de la barre de progression et de la jauge, comme une sorte de barre de progression circulaire ou un camembert. Et ceci nous amène à

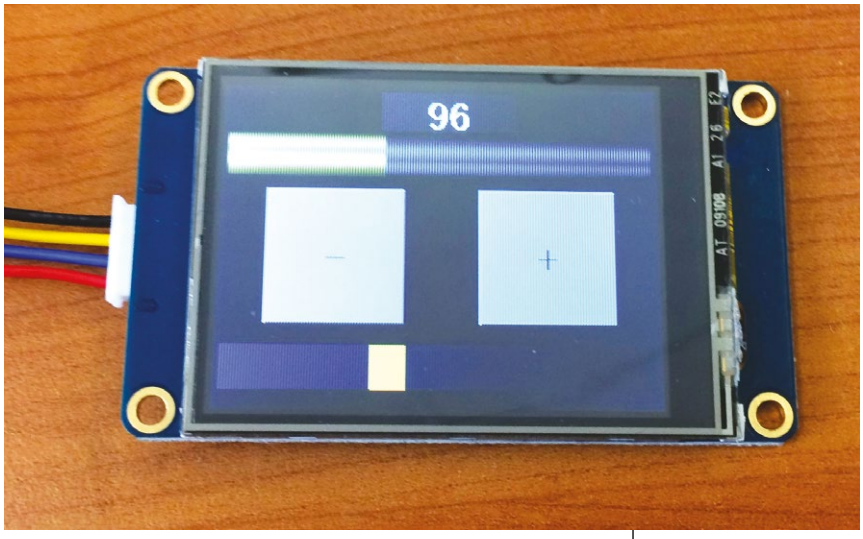
Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)

un autre point gênant : l'absence d'ouverture du logiciel et des données qu'il produit.

En effet, si une solution de développement était disponible, ou que les sources du logiciel étaient accessibles (ou le format des données documenté), un développeur pourrait créer et proposer des composants supplémentaires. On pourrait même imaginer des greffons s'ajoutant au logiciel permettant de compléter ainsi la boite à outils disponible avec des composants non officiels. Le protocole de communication série est documenté, mais ceci n'est malheureusement pas suffisant pour permettre ce type de participation communautaire.

Enfin, nous avons un problème que bon nombre d'utilisateurs rencontrent tôt out tard et qui revient régulièrement sur les forums : la difficulté de faire cohabiter la gestion d'interface avec le pilotage de l'affichage. Nos deux exemples abordent deux aspects pratiques des écrans Nextion : l'Arduino qui utilise les composants pour afficher des informations, et inversement, l'interface graphique de l'écran qui permet de piloter des fonctionnalités gérées par la carte Arduino.

Si nous prenons un cas simple mélangeant ces deux concepts, les problèmes apparaissent : une interface présentant une barre de progression, deux boutons pour augmenter/diminuer la valeur et un troisième, à deux états, pour débrailler ce contrôle et ajuster la barre de progression via un potentiomètre relié à l'Arduino. On se retrouve ici dans une situation où l'écran doit pouvoir envoyer des événements au croquis (événements) mais, dans le même temps, où le croquis doit pouvoir envoyer des ordres à l'écran. Le tout de façon asynchrone.



Une fois l'interface chargée dans la mémoire de l'écran, l'affichage et les interactions sont gérés de façon totalement indépendante du croquis Arduino. Celui-ci n'a alors pas à s'occuper de la gestion graphique, mais uniquement de la transmission des données à afficher et du traitement des événements liés à l'utilisation de l'interface.

Le problème qui se pose est qu'en utilisant n'importe quelle méthode de la bibliothèque dans `loop()`, en plus de l'appel à `nexLoop()`, nous introduisons une désynchronisation. En donnant un ordre à l'écran, nous vidons le tampon où se trouvent les données de communication. Si à cet instant celui-ci contient un message provenant de l'écran (comme une utilisation du bouton pour débrailler le contrôle), le contenu est perdu. Dans la pratique, on peut effectivement passer du contrôle par boutons au potentiomètre, mais on ne peut pas faire l'inverse. Le croquis passe son temps à donner une nouvelle valeur à la barre de progression, effaçant tout message en attente dans le sens opposé. Il est possible de contourner ce comportement avec le composant « Timer », mais le résultat n'est ni idéal ni totalement efficace.

Notez que ceci n'est pas réellement un bogue, mais plutôt une conséquence inévitable d'un choix technique avec lequel il faut composer en s'interdisant simplement d'appeler n'importe quelle fonction qui agit sur l'écran en plus de `nexLoop()`, en dehors des fonctions *callback* bien sûr. Pour certains projets ceci peut rapidement devenir un casse-tête.

Malgré tous ces points, je considère cependant que les écrans Nextion restent très pratiques et nous en reparlerons sans doute pour aborder des points et des fonctionnalités spécifiques les concernant, dans le cadre de projets divers. Qui sait, avec le temps, les désagréments seront peut-être corrigés, voire l'éditeur d'interface passé en open source.... **DB**

Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)

DÉCOUVREZ LORAWAN ET CRÉEZ VOTRE PASSERELLE / CONCENTRATEUR

Denis Bodor

LoRa et LoRAWAN sont des termes qui apparaissent de plus en plus fréquemment dès lors qu'on parle de l'Internet des objets (IoT). C'est l'un des deux standards, avec Sigfox, qui semble définitivement s'installer dans ce domaine. Nous allons explorer ici cette technologie et la mettre en pratique, brique par brique, car l'étendue de choses à comprendre, faire et maîtriser à propos de LoRaWAN est à la mesure des avantages qu'on peut en tirer : impressionnants.

Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)

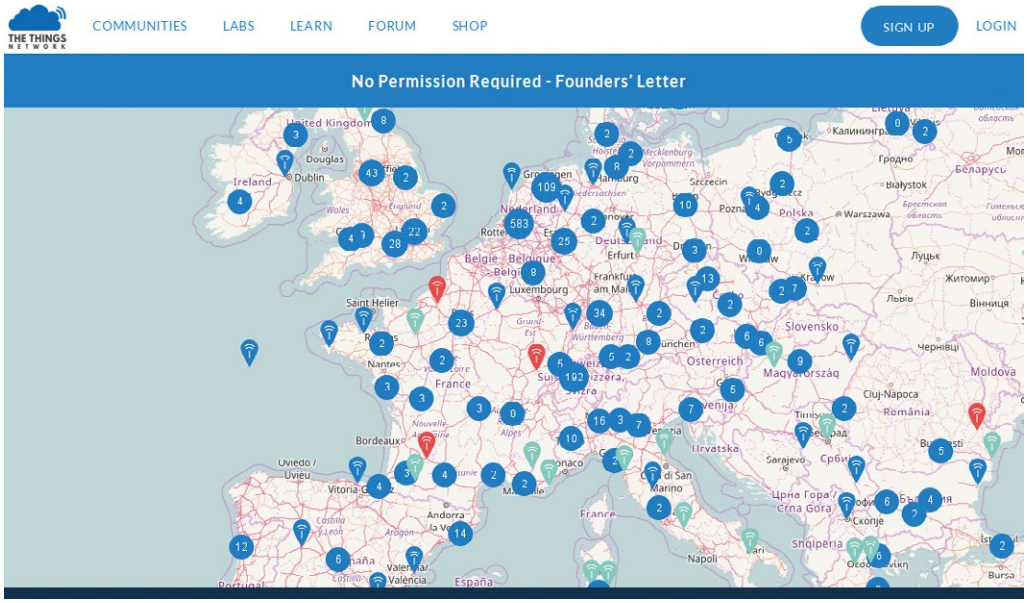
Pour comprendre l'intérêt de LoRaWAN, il faut avant tout constater les limitations des technologies existantes dans le domaine des objets connectés. Je pense qu'il n'est pas nécessaire de détailler outre mesure le premier point important : la nécessité d'une communication sans fil. Il est désormais clairement entendu qu'il n'est pas souhaitable, acceptable ou possible, dans de nombreux cas, de « tirer » des câbles pour connecter des objets communicants (même si dans certains cas, en ce qui me concerne, cela reste l'approche par défaut).

On parle donc ici de technologies sans fil et la gamme de standards, protocoles et normes est relativement conséquente, mais aucun n'est réellement adapté à la construction d'un WAN (*Wide Area Network*) ou réseau étendu. À l'inverse d'un LAN (*Local Area Network*), comme celui formé par un réseau Ethernet par exemple, un WAN couvre une vaste zone géographique. On parle ici d'une échelle allant de la ville à la planète entière en passant par les régions et les pays. Un exemple de WAN est le réseau téléphonique ou 3G/4G permettant à n'importe quel périphérique compatible (et un contrat auprès d'un opérateur) de se connecter et d'échanger des données avec tous les autres périphériques du réseau.

En examinant les technologies utilisées dans d'autres domaines, on constate que rien ne fait totalement l'affaire : le Wifi est une technologie LAN et très énergivore, le Bluetooth concerne les liaisons point à point faible distance, ZigBee a une consommation réduite, mais une portée d'une dizaine de mètres seulement, etc. Une nouvelle technologie était donc nécessaire avec un cahier des charges précis : faible consommation, sans fil, portée de plusieurs kilomètres, bas débit, architecture WAN, communications sécurisées... L'une des technologies maintenant disponibles est LoRaWAN, l'objet du présent article.

1. LORAWAN

LoRaWAN est un protocole initialement développé par la société grenobloise *Cycléo* après son rachat par l'américain *Semtech* (ou *SMTC Corporation*) en 2012, utilisant



Le projet « The Things Network » (TTN) est né à Amsterdam, rien d'étonnant donc qu'il y ait une très forte densité de concentrateurs LoRaWAN dans cette région. Comme vous pouvez le constater, la France est encore assez pauvre en installations pour le moment, mais de nouveaux concentrateurs sont activés tous les jours. Nul doute que tout ceci évoluera, peut-être même grâce à vous...

Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)

une technologie de communication radio appelée LoRA pour *Long Range* (« longue portée »). LoRaWAN permet de créer ce qu'on appelle un LPWAN ou *Low-Power Wide-Area Network*, un réseau étendu à faible consommation énergétique, permettant de connecter des objets communicants (IoT). Attention cependant, il ne s'agit pas ici de créer quelque chose comme un réseau Wifi ou 4G, l'objectif est une communication longue portée, à bas débit et pour des échanges succincts. N'espérez pas transmettre des images ou un flux de données, LoRaWAN n'est pas fait pour cela.

Comprenez également que LoRaWAN n'est pas une technologie de communication (ça, c'est LoRa), mais une manière de construire une architecture complète, en utilisant des protocoles bien précis, permettant à des objets connectés de communiquer à travers une liaison radio et surtout via Internet. La technologie populaire la plus proche de LoRaWAN permettant de comprendre cette notion est le réseau de téléphonie mobile 3G/4G/GPRS. Ce réseau ne se résume pas à un smartphone et une antenne (*base station*), mais intègre des éléments invisibles pour le commun des mortels et en particulier toute la partie se trouvant derrière les antennes et le contrôleur du réseau d'antenne (RNC).

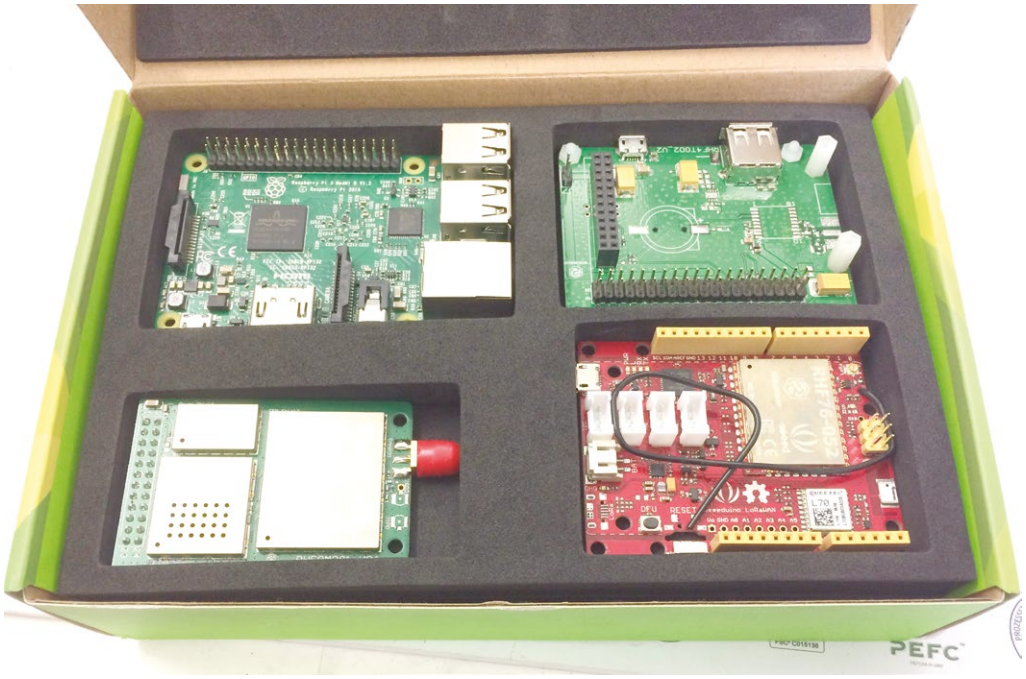
Il est possible d'utiliser LoRa sans LoRaWAN, pour une liaison entre deux points, mais tout l'intérêt est de permettre ici à des montages d'envoyer et recevoir des données via Internet, et non simplement entre eux. Il n'y aurait pas de réseau dans le cas contraire et encore moins de réseau étendu.

L'architecture LoRAWAN est constituée de plusieurs éléments :

- Les nœuds, *nodes* ou *end points* sont les objets connectés devant transmettre des informations (*uplink*) ou recevoir des ordres (*downlink*). Il peut s'agir de produits manufacturés, mais aussi, et surtout, dans notre cas, de montages reposant sur l'utilisation d'un module, d'un shield ou d'un hat LoRa.
- Les passerelles, *gateways* ou concentrateurs se trouvent à l'autre bout de la communication radio LoRa et sont de plus connectés à un serveur internet d'un opérateur LoRaWAN, via une liaison Ethernet, Wifi, 3G ou un autre réseau intermédiaire (dit *backhaul*). Le rôle de ces concentrateurs est de faire simplement transiter les données transmises via LoRa sur Internet en direction ou depuis le serveur. Un concentrateur n'utilise normalement pas le même matériel qu'un nœud puisque le standard précise qu'il doit être capable de supporter au minimum 8 connexions simultanées. Il est possible de créer un concentrateur sur la base d'un module LoRa ne supportant qu'une communication à la fois (c'est moins cher), mais ce n'est alors pas un vrai concentrateur LoRaWAN au sens strict du terme. On appelle normalement ce type d'installation un *forwarder* (relayeur).

Il faut faire très attention en achetant votre matériel LoRa et LoRaWAN, car sur les trois gammes de fréquences utilisées seules deux peuvent l'être en Europe (915 Mhz ne fait pas partie des bandes ISM ici). Le choix des 868 Mhz est donc presque obligatoire chez nous, puisque les 433 Mhz sont déjà très fortement saturés et la puissance utilisable 50 fois inférieure à la bande des 868 Mhz.

Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)



Le produit utilisé pour installer notre concentrateur LoRaWAN provient de chez SeeedStudio. Il comprend un module concentrateur LoRaWAN RisingHF, un adaptateur pour Raspberry Pi, une Raspberry Pi 3 et une carte Seeeduino LoRaWAN pour créer un node, ainsi que toute la connectique nécessaire. Deux points noirs cependant : le prix (~340€) et l'adaptateur d'alimentation au format US.

- Le serveur LoRaWAN, ou serveur réseau, recueille et gère les communications ayant transitées via les concentrateurs. C'est également lui qui prendra en charge la sécurité, autorise ou non un node pour une application donnée et déchiffrera une partie des données. Ce dernier point est très important, car il assure la sécurité du réseau. Un node LoRaWAN chiffre les données et s'authentifie auprès du serveur. Ce faisant, le concentrateur n'a aucune connaissance des données qui transitent, il ne fait que relayer l'information chiffrée. Ceci permet aux nodes d'utiliser n'importe quel concentrateur en toute sécurité, quelle que soit l'identité de celui qui le met à disposition.
- Le serveur d'application, ou une application, représente la destination finale des données envoyées par un node ou, dans le sens opposé, la source de contrôle. Cet élément sait quoi faire des données et, généralement les représente de façon adaptée. Les applications ont connaissance de la liste des nodes qui leur sont associés et du type de données à prendre en charge. L'application est connectée avec le serveur LoRaWAN d'une manière qui est propre au serveur utilisé. Il peut s'agir d'un programme utilisant une interface de programmation (API) dédiée, une application construite avec un kit de développement (SDK) ou, solution sans doute la plus simple, d'une intégration avec un autre service chargé de stocker et présenter les données (comme myDevices Cayenne par exemple).

En tant qu'utilisateur/développeur/bidouilleur vous pouvez, dans l'absolu, construire intégralement un réseau LoRaWAN en gérant chacun de ces éléments, mais cela ne représente pas vraiment d'intérêt sauf si vous êtes capable d'installer des concentrateurs partout. L'idée est justement de mettre en place un réseau massif permettant à tous les objets connectés de communiquer dans une architecture LoRaWAN.

Les opérateurs de téléphonie comme Orange et Bouygues (Objenious) l'ont bien compris et ont d'ores et déjà entrepris de déployer leur réseau LoRaWAN. Orange fait d'ailleurs partie de l'alliance LoRa depuis l'année dernière et rejoint ainsi Bouygues Telecom, IBM, Bosch, Swisscom ou encore Schneider dans cette association ayant pour objectif de standardiser LoRaWAN, promouvoir son utilisation et, bien entendu, procéder à des certifications

Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)

L'assemblage du concentrateur est un jeu d'enfant et c'est là l'un des avantages d'un kit par rapport à l'achat d'un module seul (plus de salade de câbles). Notez l'alimentation de l'ensemble se faisant par la carte adaptatrice du module qui fournit ensuite le courant à la Pi, et non l'inverse.

d'équipements. Bien entendu, bénéficier des services LoRaWAN futurs d'un tel opérateur impliquera très certainement la souscription d'un abonnement, exactement comme pour la téléphonie mobile.

Il existe cependant une alternative puisque, contrairement à la 3G, les fréquences radio utilisées par LoRa et donc pour toute la partie sans fil de LoRaWAN ne nécessitent pas de licence d'utilisation spécifique. Les équipements LoRa/LoRaWAN utilisent en effet les bandes de fréquences ISM (industriel, scientifique et médical) de 433 MHz, 868 MHz et 915 MHz. En Europe, 915 Mhz n'est pas dans les bandes ISM et ne doit pas être utilisé. Les équipements se basent donc tous sur 868 Mhz, car la bande de 433 Mhz est largement « polluée » par toutes sortes d'équipements radio et la puissance (PAR) est limitée à 10 mW (contre 500 mW en 868 Mhz).

Ce choix de 868 Mhz est un point important, car lors de l'achat de votre matériel, vous devrez choisir la fréquence sur laquelle il fonctionne. La bande des 915 Mhz **ne peut pas et ne doit pas** être utilisée en Europe.

L'utilisation de ces « fréquences non règlementées » (terme courant, mais faux, puisque justement leur utilisation fait l'objet d'une règlementation qui décrit leur « libre » utilisation) permet à tout un chacun de mettre en œuvre son équipement LoRaWAN. Il est donc possible d'avoir une approche communautaire où des particuliers installent leur concentrateur et permettent ainsi à tout le monde d'utiliser le réseau pour leurs objets connectés.

Cette dernière année, bon nombre de fournisseurs de service LoRaWAN (ou *network service providers*) ont vu le jour. On peut citer, par exemple Loriot, Lace, Senetco, ThingPark ou encore UbiQ. Certains sont partiellement communautaires tout en imposant des restrictions dans le cadre d'une utilisation gracieuse. C'est le cas par exemple de Loriot, permettant l'utilisation de leur serveur LoRaWAN pour connecter un concentrateur ou encore d'utiliser ce même serveur via un concentrateur qui y est connecté. Un abonnement payant devient rapidement nécessaire dès lors qu'on souhaite utiliser certaines fonctionnalités comme l'envoi de données du serveur vers un node (*downlink*), l'utilisation de plus d'une application, l'accès à l'interface de programmation (API) ou encore l'utilisation du mécanisme de sécurité le plus sûr (OTAA et non ABP).

Heureusement pour nous, il existe une autre solution : *The Things Network*.

2. THE THINGS NETWORK

The Things Network ou TTN pour les intimes est un véritable effort communautaire reposant sur trois principes simples :

- vos données sont vos données ;
- neutralité du net, toutes les données sont traitées de façon égale ;

Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)

• tout est open source (disponible via GitHub).

TTN a vu le jour à Amsterdam du fruit du financement participatif et fonctionne de façon décentralisée. Un ensemble de serveurs réseau est disponible, géré indépendamment de toute structure commerciale. TTN gère également l'aspect application en permettant une visualisation des données reçues des nodes, une action sur ces derniers, mais également une solution d'intégration en faisant office de relais vers une solution de stockage ou encore quelque chose de plus avancé comme myDevice/Cayenne. Il est également possible de créer sa propre application en Go, Java, Node.js ou Node-RED puisque TTN fournit une interface de développement (API) relativement complète et, bien entendu, totalement open source.

Cette approche très ouverte explique sans le moindre doute la popularité de cette solution auprès des hobbyistes comme nous, mais TTN va plus loin et est en mesure de fournir une plateforme tout aussi bien pour des particuliers, des entreprises ou même des villes. L'Internet des objets et LoRaWAN ne se limitent pas à la capacité à déclencher à distance une machine à café ou à obtenir régulièrement la température dans son salon. Les serveurs réseaux TTN associés à une couverture raisonnable par les concentrateurs offrent également bien des services pour une municipalité par exemple, et TTN ne fait pas de distinction à ce niveau, ni dans un sens ni dans l'autre. La ville d'Amsterdam par exemple, lieu de naissance de TTN, a été couverte à l'aide d'une dizaine de concentrateurs (même s'il y en a maintenant plus d'une cinquantaine).

La France est un peu à la traîne de ce point de vue en comparaison de l'Allemagne, la Suisse, la Belgique et les pays nordiques, mais qui sait, peut-être que cet article changera les choses. Vous pouvez vous informer très simplement sur la présence d'un concentrateur connecté à TTN en pointant votre navigateur sur **https://www.thethingsnetwork.org/map**. Comme vous pouvez le voir, il nous reste encore du travail, même si 24 concentrateurs sur Paris et 16 sur Grenoble est un bon début...

TTN fournit donc les serveurs réseau et des solutions pour l'intégration. Vous, en tant qu'utilisateur, pouvez intégrer cette plateforme de deux manières : en tant que simple client si vous disposez d'un concentrateur à portée, ou en installant un nouveau concentrateur pour couvrir une nouvelle zone. Les deux ne sont pas liés, ceci ne fonctionne pas comme la couverture en point d'accès Wifi Fon où les personnes partageant leur connexion peuvent utiliser le réseau gratuitement et les autres doivent payer. Ici, rien ne vous empêche d'utiliser un concentrateur pour vos nodes sans pour autant devoir en installer un.

Enfin, vous pouvez également participer à TTN en créant une communauté afin de regrouper les utilisateurs localement, dans une ville par exemple, pour échanger des informations, partager ses expériences, apporter de l'aide aux débutants ou encore organiser des rencontres, etc. La communauté de Paris, par exemple, regroupe 40 personnes et 8 concentrateurs, la ville est presque couverte, mais, là encore, il reste du travail. Pour voir si une communauté existe près de chez vous, visitez simplement la carte au bas de cette page **https://www.thethingsnetwork.org/community**.

3. CRÉONS NOTRE CONCENTRATEUR LORAWAN : LE MATÉRIEL

Certes, j'aurai pu commencer par le plus aisé en détaillant la mise en route d'un node sur base Arduino, mais... non. Instinctivement, en me penchant sur loRaWAN et en découvrant TTN, je n'étais pas à l'aise à l'idée d'utiliser ces ressources, un concentrateur proche et les serveurs réseau TTN sans avant tout apporter ma petite pierre à l'édifice.

Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)

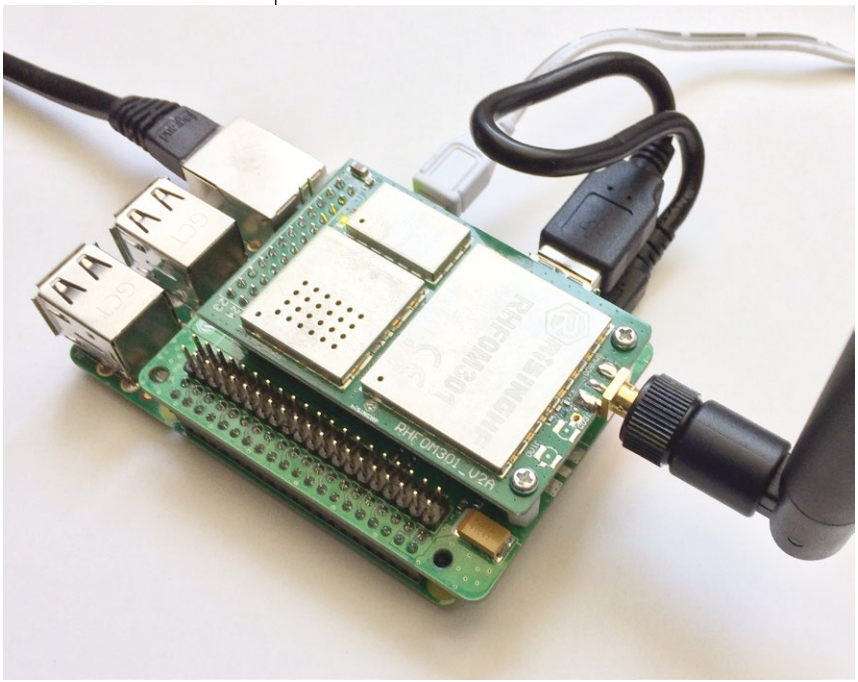
Colmar, ma bien aimée ville, ne disposant pas d'un seul concentrateur, j'aurai pu tenter d'en utiliser un se trouvant à Bâle ou encore plus loin. Ceci peut paraître surprenant, mais LoRa porte bien son nom et, avec une vue dégagée en direction d'un concentrateur donné, la portée peut être très impressionnante. Andreas Spiess (le gars avec l'accent suisse sur YouTube) a réussi à envoyer des données au concentrateur du Weissenstein (1291m d'altitude tout de même), depuis le château du Hohenbourg au sommet du Schlossberg (553m d'altitude) à la frontière nord du Bas-Rhin... à 201 kilomètres !

Cette incroyable distance de communication en LoRa reste toutefois exceptionnelle et des conditions spécifiques doivent être réunies pour y arriver. En temps normal les distances se mesurent en kilomètres ou en dizaines de kilomètres, mais pas en centaines. Il faut distinguer deux types de conditions lorsqu'on parle de la portée de LoRa (ou de tout autre système radio longue distance) :

- LOS (*Line Of Sight*) pour « en ligne de mire » ou « en vue dégagée ». La propagation des ondes radio est impactée par la présence d'objets massifs comme des bâtiments, des collines, des bois denses, ainsi que par d'éventuelles interférences électromagnétiques comme celles générées par les lignes à haute tension, les pylônes métalliques, etc. La distance de communication s'en trouve grandement réduite et, bien entendu, une ligne d'horizon clairement dégagée permet une portée maximum. On parle également tantôt de zone rurale pour désigner une communication en conditions LOS, et la portée peut normalement atteindre une vingtaine de kilomètres ou plus, en fonction de l'équipement utilisé.
- NLOS (*Non Line Of sight*) pour la condition inverse, avec une vue non dégagée donc. Il s'agit ici d'une communication dans un environnement urbain avec un nombre important de bâtiments limitant la propagation des signaux, parfois drastiquement. Dans ces conditions, une distance d'une paire de kilomètres tout au plus peut être envisagée, moins si le concentrateur ou le node sont en intérieur et/ou au sol.

Le peu de concentrateurs présents en France et une absence totale dans ma ville sont donc une excellente raison de procéder à une première installation. Il existe plusieurs solutions pour ce faire : acquérir un équipement tout fait, opter pour un module ou une carte (IMST iC880A, Multitech mCard-LoRa, Cisco LoRa card, Chistera-Pi, etc.) ou trouver un kit complet.

La carte adaptatrice présente une « copie » des broches de la Pi permettant ainsi de connecter différents montages. Ceci reste cependant assez secondaire, en dehors de la connexion d'un adaptateur USB/série, puisque dans la quasi-totalité des installations, un tel équipement ne fait qu'une chose et une seule : concentrateur LoRaWAN.



Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)

DISPONIBLE DÈS LE 14 JUILLET

LINUX PRATIQUE HORS-SÉRIE N°39 !

MÉMO
LIGNE DE
COMMANDES

NE LE MANQUEZ PAS

CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :

http://www.ed-diamond.com

Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)
Sous réserve de toutes modifications.

ACTUELLEMENT DISPONIBLE

LINUX PRATIQUE N°102

J'APPRENDS À PROGRAMMER

AVEC PROCESSING !

NE LE MANQUEZ PAS

CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :

http://www.ed-diamond.com

Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)

Pourquoi pas Loriot ?

Le kit de SeeedStudio était livré avec une carte microSD contenant un système Raspbian équipé des éléments logiciels nécessaires pour rejoindre le réseau Loriot. Cette plateforme propose un usage gratuit, appelé « *Tier Community Network* », mais celui-ci est grandement restreint.

Toutefois, le plus important à mes yeux était l'absence de réelle ouverture. Le téléchargement de l'image de la carte SD par exemple, nécessite un enregistrement en ligne (??), la documentation de RisingHF concernant son matériel est réduite et ne parle que de Loriot, les sources du serveur réseau ne sont pas disponibles, le forum est d'une vacuité effrayante, etc. Ce n'est pas ce que j'appelle un fonctionnement ouvert et un effort communautaire.

Après un rapide essai de Loriot, facilité par la préconfiguration du système, mais ne pouvant pas tester l'ensemble des fonctionnalités, j'ai donc tout aussi rapidement réutilisé la carte SD pour une installation fraîche de Raspbian. Je cherche toujours comment supprimer le compte crée pour l'occasion sachant qu'en contactant le support par mail un nouveau compte est automatiquement créé sur le « Support Desk » et une demande de confirmation s'en suit... Tout cela pour que le support me réponde que la suppression doit être faite manuellement de leur côté, mais qu'il n'y a pas de raison de le faire... Je déteste ce genre de chose.

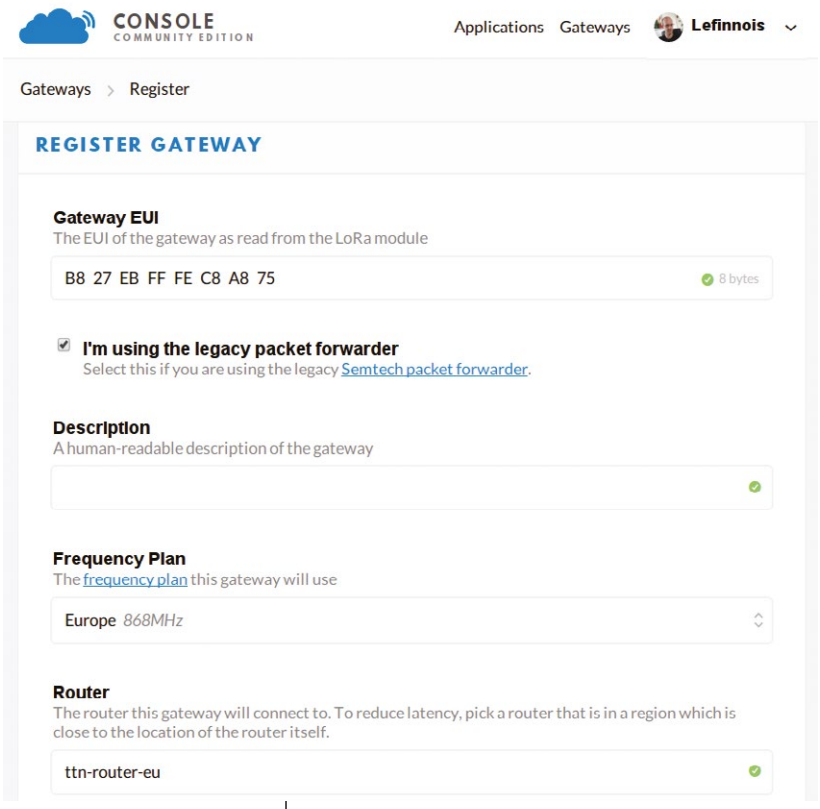
Bien que la carte concentrateur IMST iC880A soit très populaire et son utilisation avec une Raspberry Pi largement documentée, j'ai préféré opter pour une autre approche en jetant mon dévolu sur le kit LoRaWAN Gateway de SeeedStudio. Celui-ci comprend :

- un module concentrateur RHF0M301 de RisingHF construit autour d'un circuit intégré Semtech SX1301,
- un adaptateur RHF4T002 permettant la connexion du module à une Raspberry Pi,
- une antenne SMA à brancher au module,
- une Raspberry Pi 3 standard,
- une carte microSD SanDisk de 8 Go classe 10, avec une version préinstallée de Raspbian intégrant tout le nécessaire pour créer un concentrateur Loriot,
- un adaptateur USR/série pour la connexion console,
- un câble USB de 20 cm pour l'alimentation de la Pi à partir d'adaptateur pour le module (à la manière de l'écran officiel Pi),
- un câble USB de 1m pour l'alimentation,
- un adaptateur secteur 5V/2A pour l'alimentation, compatible ~240V, mais avec un connecteur US,
- un câble Ethernet de 2 m,
- et une carte Seeeduino LoRaWAN (microcontrôleur ATSAMD21G18 comme l'Arduino Zero) avec GPS pour créer votre premier node.

Le tout dans une jolie boite pour quelques 340€. Vous l'avez compris, on parle ici d'un budget relativement conséquent. Ce kit proposé par Seeed est une déclinaison personnalisée du « *RisingHF IoT Dicovery* », comprenant quelques éléments supplémentaires et/ou différents. Cela reste une grosse somme, certes, mais tout arrive presque clé en main, prêt pour être assemblé.

Les solutions n'incluant ni Pi, ni nodes ou shield pour créer un node, comme la carte IMST iC880A, nécessitent des achats complémentaires, l'immobilisation d'une Pi, et une interconnexion entre la Pi et le module LoRaWAN souvent source de problèmes. En faisant rapidement le calcul, une Pi 3, une bonne carte SD, une alimentation suffisante, une antenne, un module iC880A... il n'est pas vraiment possible de s'en sortir pour moins de 250€ ou 300€ et vous devrez tout connecter manuellement.

Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)



Une fois votre installation terminée, tant matérielle que logicielle, il ne vous restera plus qu'à créer un compte TTN et à enregistrer votre concentrateur. Celui-ci sera alors visible par l'ensemble des utilisateurs TTN qui, comme vous, pourront l'utiliser pour leurs nodes.

La décision de créer un concentrateur LoRaWAN n'est pas à prendre à la légère. Non seulement l'aspect financier est important, mais il est également très peu probable que vous utilisiez cette installation pour autre chose. Vous devez voir cela comme une sorte de déploiement d'un point d'accès Wifi communautaire, un investissement que vous faites pour vous-même, mais également pour les autres. Intégrer votre concentrateur dans le réseau TTN n'est pas un projet d'un week-end qui pourra ensuite être démonté pour un autre usage, ceci implique une certaine responsabilité vis-à-vis des utilisateurs... Enfin, je pense important de préciser qu'il n'est pas nécessaire qu'un grand nombre de personnes décident de déployer un concentrateur bien placé pour couvrir une ville, l'investissement est donc loin d'être obligatoire pour tous les utilisateurs.

L'assemblage des composants du kit est aisé, il suffit d'enficher le module LoRaWAN sur l'adaptateur et ce dernier sur la Pi. Des entretoises en plastique permettent d'assurer une certaine robustesse, mais vous devrez fouiller dans votre boite à vis pour fixer solidement le tout, car les vis ne sont pas fournies. On placera ensuite rapidement l'antenne avant n'importe quelle autre étape, car il est hors de question de mettre l'ensemble sous tension sans antenne.

On place ensuite le plus petit des câbles USB sur le port de type A (le grand) de l'adaptateur et sur le connecteur micro-B de la Raspberry Pi. La Pi se trouve donc alimentée par l'adaptateur qui, lui-même est ensuite raccordé au bloc d'alimentation avec le long câble USB. Notez qu'une alimentation standard de Pi 3 peut être utilisée en lieu et place de celle livrée au format US. Celle-ci devra pouvoir fournir un courant suffisant pour alimenter le module LoRaWAN et la Pi (5V et 2,5A est le choix standard).

Un concentrateur LoRaWAN est typiquement une installation sans écran/clavier, mais le démarrage d'un système Raspbian Jessie Lite récent sur une Pi 3 nécessite une légère modification du contenu de la carte microSD. Après téléchargement et copie de l'image sur le support, vous devrez donc accéder à la première partition pour éditer le fichier `cmdline.txt` et ajouter `console=ttyAMA0,115200` en lieu et place de `console=serial0,115200`. Ceci vous permettra de voir le système démarrer via un adaptateur USB/série connecté aux broches 8 (TX), 10 (RX) et 6 (masse), et bien entendu d'accéder au système.

On s'empressera ensuite d'activer le serveur SSH via un appel à `sudo raspi-config`, d'étendre la partition à la taille de la carte, de

Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)

configurer les locales (fuseau horaire, etc.) et d'activer le SPI puisque le module LoRaWAN est connecté de cette façon. Bien entendu, vous pouvez également temporairement brancher un écran HDMI et un clavier pour ces opérations, mais je trouve personnellement cela plus pratique avec une liaison série (question d'habitude). J'ai d'ailleurs profité de l'occasion pour désactiver l'audio et le Bluetooth via les lignes **`dtparam=audio=off`** et **`dtoverlay=pi3-disable-bt`** dans **`config.txt`**.

L'adaptateur connecte la Pi avec le module LoRaWAN selon le brochage suivant :

- MOSI broche 19 (BCM GPIO 10) ;
- MISO broche 21 (BCM GPIO 9) ;
- SCLK broche 23 (BCM GPIO 11) ;
- CS0 broche 24 (BCM GPIO 8) ;
- CS1 broche 26 (BCM GPIO 7).

Toutes les broches sont standards d'une connexion SPI hormis CS1 côté Pi qui est en réalité connecté à la broche de reset du module. Ce point est important, car c'est ce qui fait toute la différence avec un module iC880A basé sur le même circuit intégré et dont nous allons utiliser le logiciel. Un reset du module est systématiquement opéré avant tout accès et si cette broche est mal désignée dans la configuration, un certain nombre d'erreurs apparaissent.

4. INSTALLATION DE VOTRE CONCENTRATEUR

Une fois votre système tout neuf démarré, à jour, et personnalisé selon vos besoins avec vos applications préférées (au hasard *Vim* et *Midnight Commander*), vous devrez installer **`git`** avec **`sudo apt-get install git`** pour ensuite récupérer les éléments logiciels adéquats. Ce qu'il faut à notre système est un *packet forwarder* chargé de relayer les données reçues en LoRa sur Internet au serveur réseaux TTN et, inversement, envoyer en LoRa les données reçues du serveur TTN. Un concentrateur n'est rien d'autre qu'un relayeur copiant les données d'un protocole vers un autre, d'où la désignation tantôt utilisée de passerelle ou *gateway* en anglais.

La communauté TTN de Zurich et plus exactement trois développeurs de cette communauté ont fait un fantastique travail pour nous faciliter la vie et permettre d'installer automatiquement ce *packet forwarder* sur notre Pi et l'intégrer proprement au système. Vous devrez tout d'abord récupérer les fichiers avec Git :

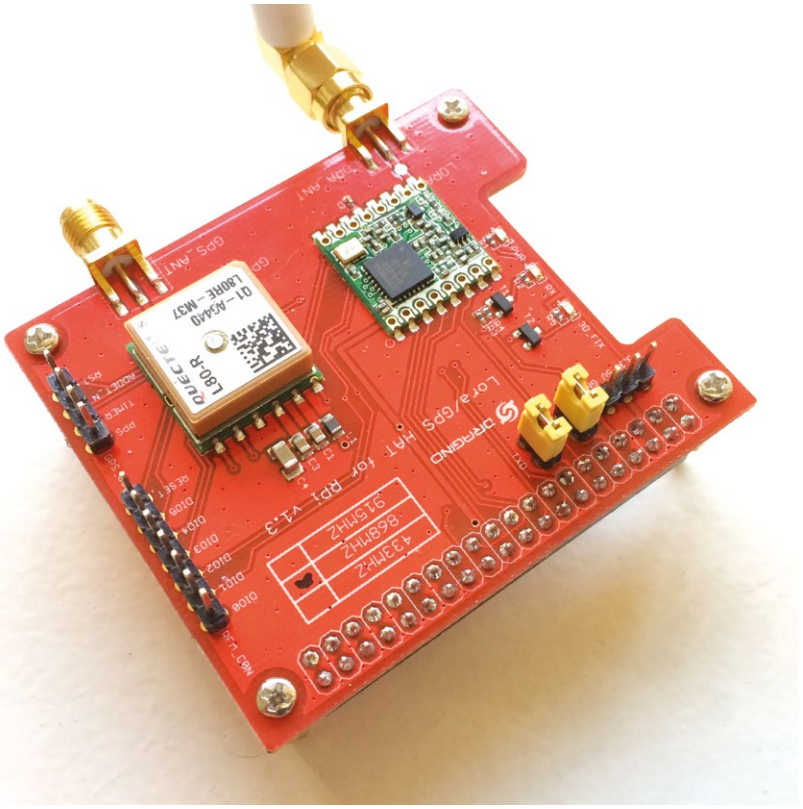
```
$ git clone https://github.com/ttn-zh/ic880a-gateway.git
Clonage dans 'ic880a-gateway'...
remote: Counting objects: 336, done.
remote: Total 336 (delta 0), reused 0 (delta 0), pack-reused 336
Réception d'objets: 100% (336/336), 7.88 MiB | 1.22 MiB/s, fait.
Résolution des deltas: 100% (201/201), fait.
Vérification de la connectivité... fait.
```

Ceci vous créera un répertoire **`ic880a-gateway`** contenant simplement quelques scripts shell et autres fichiers texte. Il s'agit d'un installateur et non des programmes eux-mêmes qui seront récupérés et compilés durant la procédure d'installation. Pour lancer l'installation, rien de plus simple, placez-vous dans le répertoire et lancez le script principal via **`sudo`** et en n'oubliant pas de préciser **`spi`** en argument :

Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)

```
$ cd ~/ic880a-gateway
$ sudo ./install.sh spi
The Things Network Gateway installer
Version spi
Updating installer files...
Already up-to-date.
New installer found. Restarting process...
The Things Network Gateway installer
Version spi
Updating installer files...
Already up-to-date.
Gateway configuration:
Detected EUI B827EBFFFEC8A875 from eth0
Do you want to use remote settings file? [y/N]
        Host name [ttn-gateway]:
        Descriptive name [ttn-ic880a]:Mon concentrateur
        Contact email: adresse@domain.tld
        Latitude [0]:
        Longitude [0]:
        Altitude [0]:
```

L'installateur va tout d'abord déterminer un identifiant unique pour votre concentrateur en se basant sur l'adresse matérielle de l'interface réseau de la Pi. Ceci permettra d'identifier votre concentrateur, plus tard, dans TTN. Vous serez ensuite invité à préciser un nom d'hôte, une courte description, une adresse mail et les coordonnées GPS de l'emplacement du concentrateur. Les valeurs par défaut sont spécifiées entre crochets et seront utilisées si vous validez simplement la question.



Si l'investissement de plusieurs centaines d'euros dans la construction d'un concentrateur est quelque chose que vous ne voulez ou ne pouvez pas faire, une solution possible est la construction d'un concentrateur monocanal. Au sens strict du terme, il ne s'agit pas d'un vrai concentrateur (8 canaux minimum), mais le coût descend à moins de 40 euros (plus la Pi et les accessoires), comme pour ce hat Dragino LoRa avec GPS.

Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)

Une fois ces informations entrées, la procédure d'installation commence et vous devrez en premier lieu valider l'installation de plusieurs paquets nécessaires à la compilation des briques de base :

```
Il est nécessaire de prendre 19,9 Mo dans les archives.
Après cette opération, 32,7 Mo d'espace disque
supplémentaires seront utilisés.
Souhaitez-vous continuer ? [O/n]
```

Après cette étape, le script téléchargera, configurera, compilera et installera les éléments dans **/opt/ttn-gateway**. Cette phase peut être relativement longue en fonction des performances de la carte SD et du modèle de Pi utilisé (au cas où vous n'optez pas pour le kit de Seeed), mais elle finira par aboutir en affichant un résumé des informations, juste avant d'installer les scripts de démarrage et de provoquer une réinitialisation de votre Pi :

```
Gateway EUI is: B827EBFFFEC8A875
The hostname is: ttn-gateway
Open TTN console and register your gateway using your EUI:
  https://console.thethingsnetwork.org/gateways

Installation completed.
Created symlink from /etc/systemd/system/multi-user.target.wants/
ttn-gateway.service to /lib/systemd/system/ttn-gateway.service.
The system will reboot in 5 seconds...

Broadcast message from pi@ttn-gateway on pts/0
(mar. 2017-05-23 16:04:07 UTC):
The system is going down for reboot NOW!
```

Ce *Gateway EUI* est important, c'est ce qui vous permettra d'enregistrer votre concentrateur sur TTN. Après ce redémarrage et avec le module Seeed/RisingHF le service ne sera pas lancé, car le script utilise par défaut une mauvaise broche pour réinitialiser le module LoRaWAN. Vous devrez éditer le fichier **/opt/ttn-gateway/bin/start.sh** et changer la ligne :

```
SX1301_RESET_BCM_PIN=25
```

en

```
SX1301_RESET_BCM_PIN=7
```

Enfin, vous pourrez redémarrer le service avec la commande **sudo systemctl restart ttn-gateway.service**. Après quelques secondes, vous pourrez vérifier le bon fonctionnement en utilisant **systemctl status ttn-gateway.service** et devrez alors voir quelque chose comme :

```
$ systemctl status ttn-gateway.service
  ttn-gateway.service - The Things Network Gateway
   Loaded: loaded (/lib/systemd/system/ttn-gateway.service; enabled)
   Active: active (running) since Wed 2017-05-24 10:00:43 CEST; 36s ago
Main PID: 3263 (start.sh)
   CGroup: /system.slice/ttn-gateway.service
           --3263 /bin/bash /opt/ttn-gateway/bin/start.sh
           --3282 ./poly_pkt_fwd
```

Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)

À présent, dès que votre Pi démarrera, le service sera lancé automatiquement et se connectera à TTN, prêt pour relayer les messages depuis et vers le module LoRaWAN.

5. CRÉATION DE COMPTE ET CONFIGURATION CÔTÉ TTN

Votre concentrateur est maintenant en marche et tente de se connecter au serveur réseau européen (par défaut) de TTN, mais il n'y a rien, pour l'instant, pour lui donner le change (en réalité si, mais les données seront marquées comme *untrusted*). La première chose à faire sera donc de vous créer un compte sur le serveur TTN en pointant votre navigateur sur **http://www.thethingsnetwork.org** et en cliquant sur le bouton ***SIGN UP*** en haut à droite de la page.

Vous serez invité à préciser un nom d'utilisateur (attention, celui-ci ne peut être changé par la suite), une adresse mail et un mot de passe. Vous recevrez ensuite un mail vous demandant de confirmer l'adresse mail utilisée et en cliquant sur le lien, votre compte sera immédiatement activé. Vous pourrez ensuite vous connecter et vous rendre sur la console TTN : **https://console.thethingsnetwork.org**.

Il est maintenant temps d'enregistrer votre concentrateur (ou *gateway*). La console présente deux grosses icônes, ***Applications*** et ***Gateway***, cliquez sur la seconde. Sur la page, choisissez ***Register gateway*** pour faire apparaître le formulaire d'enregistrement. Cochez l'option ***I'm using the legacy packet forwarder*** puisque nous venons d'installer le *packet forwarder* « classique » dérivé de celui créé par Semtech pour ses produits. Ce changement vous oblige à utiliser un identifiant unique composé de 8 valeurs hexadécimales pour votre concentrateur, c'est celui qui s'affiche au terme de l'installation et qui est issu de l'adresse matérielle de votre interface réseau de la Pi. Recopiez simplement ici cet identifiant.

Ajoutez ensuite une description et choisissez un « *frequency plan* » correspondant à l'endroit où vous vous trouvez. Ici, en Europe, la gamme de fréquences est nommée EU868 ou « Europe 868Mhz ». En sélectionnant cette option, le champ ***Router*** bascule automatiquement sur « *ttn-router-eu* », c'est le serveur réseau qui prendra en charge votre concentrateur, qui en toute logique, est celui utilisé pour l'Europe.

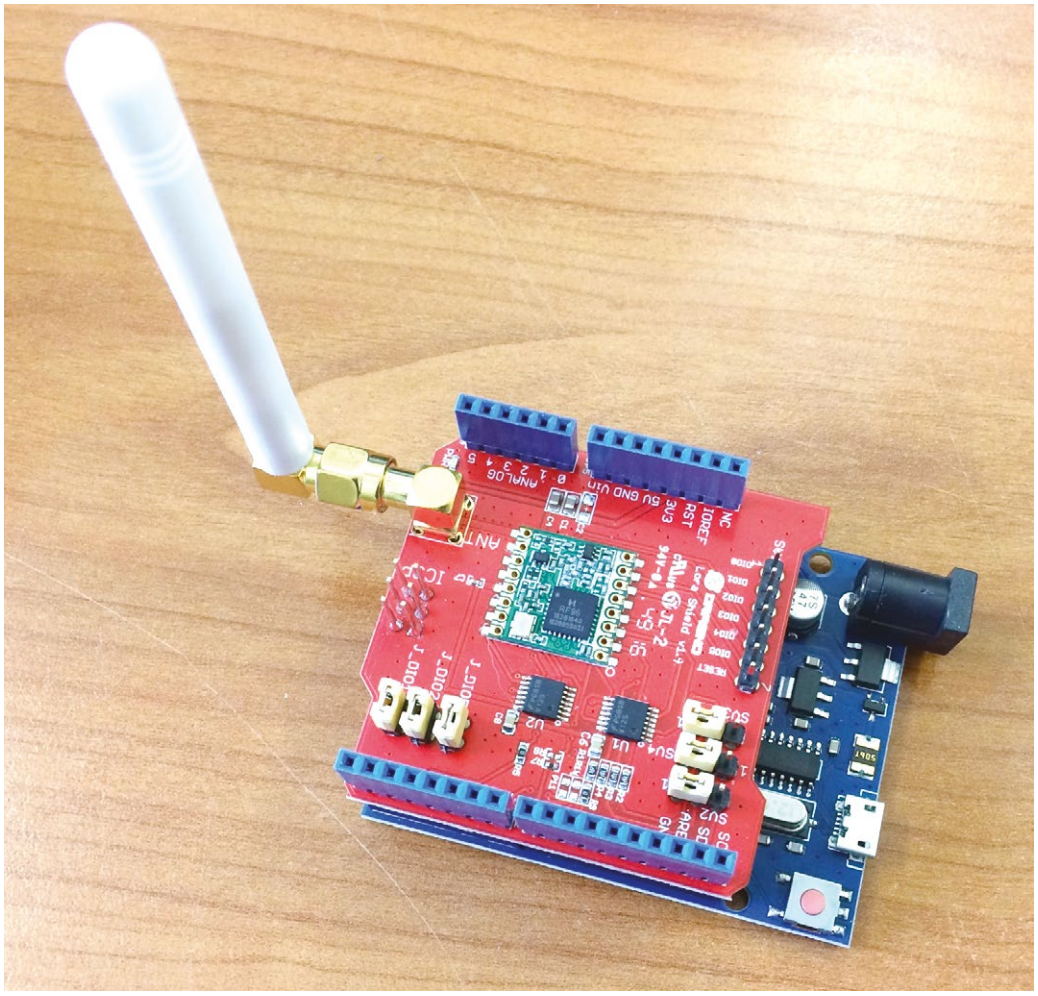
Enfin, placez votre concentrateur sur la carte de façon à permettre aux autres utilisateurs de savoir où il se situe et donc d'estimer s'il est à leur portée, précisez ensuite s'il se trouve en intérieur (*Indoor*) ou en extérieur (*Outdoor*), puis validez le tout en cliquant sur ***Register Gateway***. Votre concentrateur apparaît maintenant dans la liste et vous pouvez voir et/ou modifier les informations le concernant.

Les éléments importants sur cette page sont ***Status*** vous présentant normalement la mention « *connected* » si le concentrateur est bien en fonction et ***Last Seen*** vous indiquant la dernière fois où le concentrateur a donné de ses nouvelles. Si tout se passe bien, votre installation devrait être en permanence connectée et donner signe de vie toutes les minutes. Vous êtes donc maintenant l'heureux propriétaire d'un concentrateur LoRaWAN TTN accessible par tous les utilisateurs TTN à proximité !

6. CECI N'EST PAS LA SEULE FAÇON DE FAIRE

J'ai ici présenté une solution matérielle qui semblait adaptée à mes besoins et relativement facile à se procurer, mais il en existe beaucoup d'autres. La carte iC880A par exemple, interfacée avec une Raspberry Pi, se configurera presque exactement de la même manière. Il existe également des solutions plus économiques, reposant sur l'utilisation d'un module LoRaWAN ne pouvant utiliser qu'un seul canal. Il n'est pas exact de parler alors de concentrateur LoRaWAN, mais ceci peut toutefois être une option (c'est toujours mieux que pas de LoRaWAN du tout).

Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)



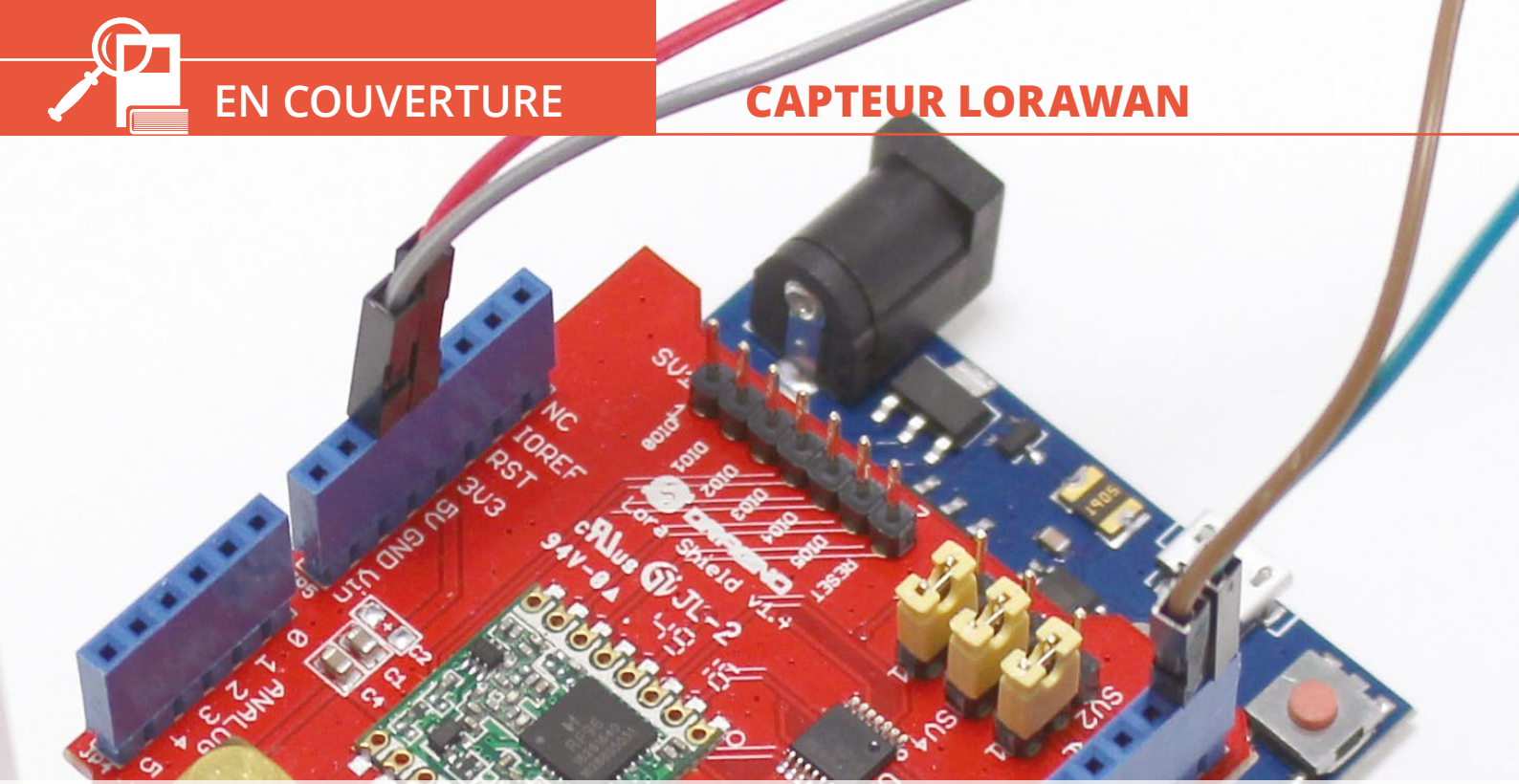
Les shields Arduino peuvent aussi servir de module pour créer un concentrateur monocanal, mais leur usage naturel est, bien entendu, d'équiper une carte Arduino pour servir de node LoRaWAN et « remonter » des informations via un concentrateur jusqu'au serveur réseau TTN, puis aux applications.

Concernant l'aspect logiciel, les éléments utilisés pour cet article sont forcément liés au matériel, mais là aussi, le script d'installation fourni par la communauté TTN de Zurich n'est pas la seule voie possible. Comme LoRa et LoRaWAN sont des technologies de Semtech, presque toutes les solutions logicielles se basent sur le code initial développé par cette société, mais il existe de nombreuses déclinaisons.

Enfin, pour une installation plus « professionnelle », il est important de mentionner l'existence de matériels clés en main comme ceux proposés par Yadom/Nemeus, Haxiot, Multitech, Kerlink, LinkLabs et bien d'autres. Ceci peut être une solution rapide pour une entreprise, une agglomération ou une collectivité locale n'ayant pas forcément les ressources humaines pour déployer, gérer et maintenir des concentrateurs « faits maison ». La plupart de ces solutions permettent d'une manière ou d'une autre une utilisation avec TTN.

Pour conclure, je préciserai que LoRaWAN, bien que grandement plébiscité, n'en est qu'à ses débuts. Ces initiatives comme TTN sont importantes, mais lorsque les fournisseurs de téléphonie mobile déploieront massivement LoRaWAN, la concurrence sera rude. Ces opérateurs possèdent en effet, indubitablement, les meilleurs emplacements pour des antennes et ainsi assurent rapidement une excellente couverture. Ceci ne sera bien entendu pas gratuit et nous n'en sommes pas encore là. Pour l'heure TTN reste, à mon sens, la meilleure option pour vous et moi. **DB**

Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)



CRÉEZ VOS MONTAGES ARDUINO COMMUNICANTS SUR LORAWAN

Denis Bodor

Dans l'article précédent, nous avons vu comment il était possible de déployer son concentrateur LoRaWAN avec TTN relativement simplement, mais ceci n'est toutefois pas indispensable. Si un concentrateur se trouve près de chez vous, vous pouvez l'utiliser pour vos projets et ainsi profiter, avec un investissement minimum, des bienfaits de LoRaWAN et des communications très longues distances pour vos projets.

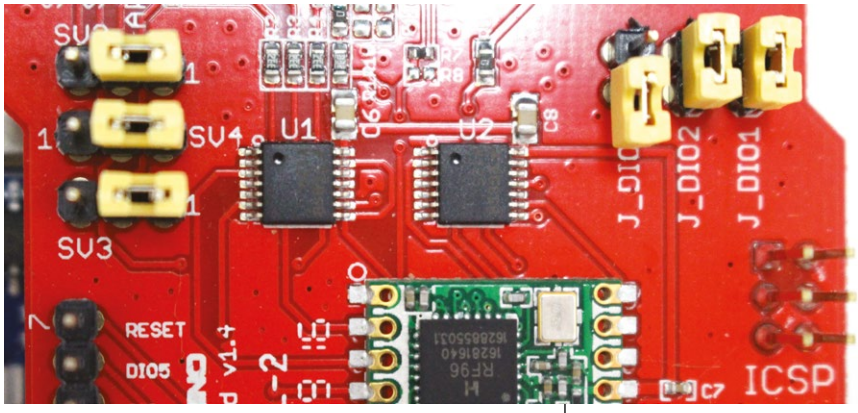
Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)

Le principe même du fonctionnement de LoRaWAN est de permettre la mise en place d'un vaste réseau. Dans le cas de *The Things Network* (TTN), ce réseau utilise des concentrateurs installés par ses membres, qu'il s'agisse de produits clés en main, de kits comme nous l'avons fait dans l'article précédent ou de l'utilisation de modules et cartes achetés séparément et utilisés avec une Raspberry Pi ou tout autre ordinateur mono-carte.

Le fait de rejoindre le réseau communautaire TTN, ou un autre du même type, ne vous oblige en rien à déployer un concentrateur. Si vous avez la chance d'en avoir un à proximité, vous pouvez tout simplement l'utiliser pour votre projet. Tout ce qu'il vous faut alors c'est un module LoRa qui permettra à votre montage de contacter ce concentrateur pour échanger des données avec TTN, puis vos applications. Créer un compte TTN suffit donc, en plus du module LoRa, à créer un capteur relié à Internet, ou en d'autres termes, un objet connecté.

1. DES ADRESSES, DES IDENTIFIANTS ET DE LA SÉCURITÉ

Un troupeau de concentrateurs installés par des particuliers, un réseau ouvert à tous, des communications radio en tous sens, des échanges sur Internet constants... LoRaWAN et *The Things Network* peuvent sembler être un véritable terrain de jeu pour qui voudrait écouter tout ce qui passe. Mais en réalité, la sécurité est au cœur de LoRaWAN, car les communications sont chiffrées entre un node et le réseau, mais également entre un node et l'application. Les données contenues dans les messages ne sont donc ni visibles du concentrateur, ni par le serveur réseau.



Le module qui équipe le shield Dragino Lora intègre une puce Semtech et divers composants annexes. Ce module peut être trouvé sur différents sites pour moins d'une dizaine d'euros, mais il ne peut être utilisé qu'en 3,3V. Sur le shield en revanche des convertisseurs de tension permettent une connexion à n'importe quelle carte Arduino.

Pour protéger ces messages et s'assurer par la même occasion des identités de chaque intervenant, on utilise des clés et des identifiants. Un processus nommé « activation » utilise ces clés et informations pour s'identifier et se connecter au réseau TTN. Ces clés sont créées ou obtenues lors de l'enregistrement d'un concentrateur ou d'un node dans votre interface TTN (console).

Il est important de comprendre ces mécanismes de sécurité, car il vous faudra les utiliser pour créer les croquis de vos nodes envoyant des données avec LoRaWAN. Il existe deux types d'activations :

- OTAA pour *Over The Air Activation* est la façon la plus sûre de se connecter au serveur du fournisseur réseau LoRaWAN (TTN dans notre cas), c'est donc celle à préférer. Avec cette méthode, on utilise l'identifiant de l'application et sa clé associée pour négocier la connexion et on obtient en retour une adresse pour notre node. Les autres clés permettant le chiffrement sont négociées automatiquement ainsi que de nombreux paramètres de communication.
- ABP pour *Activation By Personalization*. Ici, les clés et les identifiants ne sont plus négociés dans un échange durant la connexion, mais sont déterminés lorsque vous enregistrez votre périphérique côté TTN. Ces

Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)

Le module utilisé par la carte Seeeduino est très différent de celui du shield. Lui aussi intègre un circuit intégré Semtech, mais également un microcontrôleur STM32 qui sert d'interface de communication. La carte Seeeduino ne dialogue pas directement avec la puce LoRa en SPI, mais avec le STM32 via une liaison série, comme un modem.

informations doivent ensuite être utilisées dans votre croquis pour permettre la connexion. Comme le précise la documentation TTN, ceci peut paraître plus simple, car il n'y a pas de processus de connexion et de négociation, mais cela implique également une sécurité plus faible. De plus, vous pouvez rencontrer des problèmes à cause d'un autre mécanisme de sécurité.

Ce denier point est important et concerne le compteur de messages. Afin de garantir la sécurité du système, celui-ci doit s'assurer qu'un échange ne puisse pas être simplement rejoué (capturé et envoyé une seconde fois). Les messages chiffrés contiennent donc un numéro ou compteur qui permet au serveur de ne pas accepter une seconde fois un même message. Avec une activation OTAA, ce numéro fait partie de la négociation (en principe), mais avec ABP c'est à votre croquis de gérer ce mécanisme. En temps normal ceci est pris en charge par la bibliothèque Arduino, mais uniquement tant que votre montage n'est pas redémarré. En cas d'arrêt et de redémarrage, le compteur recommencera du début et le serveur va ignorer les messages pensant qu'il s'agit de copies. Une solution pour régler le problème est de désactiver cette fonction côté TTN, mais cela réduit grandement la sécurité.

Dans le mécanisme OTAA, on utilise l'identifiant de l'application (AppEUI) et la clé de l'application (AppKey). L'adresse du node (DevAddr), la clé de session réseau (NwkSKey) et la clé de session de l'application (AppSKey) sont générées à partir de l'identifiant et de la clé d'application durant la connexion. La première clé de session protège la communication entre le node et le réseau et la seconde étend cette protection jusqu'à l'application. Dans les deux cas, le concentrateur ne voit aucune des données qui circulent et dans le second, le serveur réseau lui-même n'a pas connaissance des données.

Le mécanisme ABP utilise également ces clés de session, mais elles ne sont pas négociées à la connexion, elles doivent alors figurer dans votre croquis et, bien entendu, correspondre aux mêmes informations présentes côté réseau lors de l'enregistrement d'un node.

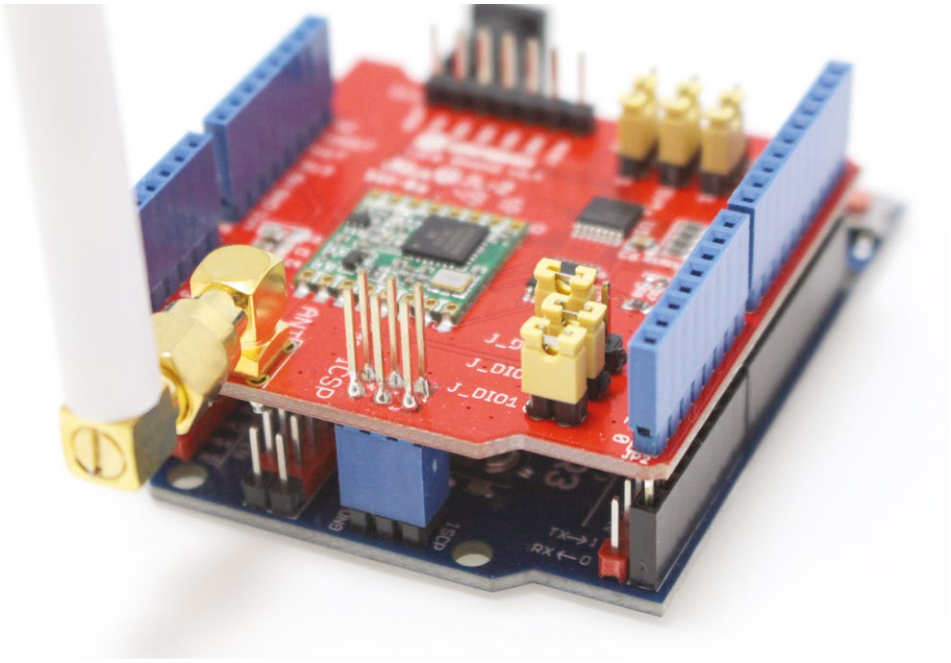
Dans le cadre de cet article, nous nous en tiendrons à l'activation OTAA, plus sûre, plus simple et recommandée par défaut par TTN. La procédure d'activation ne concerne en effet pas seulement la sécurité, mais également la transmission d'informations spécifiques et en particulier les fréquences utilisées. Dans la partie précédente, j'ai précisé que ce qui fait un véritable concentrateur LoRaWAN est sa capacité à gérer 8 canaux de communication et non un seul. Chaque canal possède une fréquence et des caractéristiques précises (débit, facteur d'étalement, rapport cyclique, bande passante). Trois de ces canaux sont définis par le standard LoRaWAN, mais les autres sont dépendants du réseau. Lors d'une activation OTAA, ces paramètres sont transmis au node lors de la connexion, mais dans le cas d'ABP ceci doit être configuré manuellement. Finalement, ABP est bien plus compliqué à mettre en œuvre et surtout, moins sûr...

En quelques mots donc : faites comme moi, tenez-vous-en à l'OTAA.

2. LE MATÉRIEL UTILISÉ

Il existe énormément de matériels permettant d'utiliser LoRa et donc de connecter un montage à LoRaWAN et TTN. Une simple recherche sur eBay des termes « *lora module* » retourne des dizaines de modèles différents, avec des prix variant entre 6€ et 30€. Les écarts de prix proviennent généralement du format

Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)



Le shield Dragino Lora peut être utilisé avec plusieurs modèles de cartes Arduino. Dans le cadre d'une carte UNO, la connexion SPI entre le module et la carte se fait de la même manière via les broches standards et le connecteur ICSP. Avec un autre modèle de carte, il est possible de choisir, via cavaliers, comment se fait la connexion.

proposé, de la qualité générale du module et du type d'antenne utilisable (si incluse).

Pour mes expérimentations, j'ai opté pour une solution plus simple que l'utilisation de modules d'entrée de gamme en choisissant d'utiliser un shield Arduino (Dragino LoRa Shield) et une carte compatible Arduino intégrant une puce LoRa (Seeeduino LoRaWAN avec GPS). Les deux produits sont construits autour de la puce Semetch SX1276. Dans le cas du shield Arduino, il s'agit d'un module HopeRF RFM95W prenant place sur le circuit imprimé et pour la carte, il s'agit d'un module RHF76-052AM de RisingHF. Le module HopeRF RFM95W est le type de produit qu'on trouve généralement pour quelques euros en ligne.

Semtech est le seul fabricant de puce LoRa/LoRaWAN et quel que soit le matériel que vous allez acheter, c'est avec une telle puce que votre croquis va dialoguer pour émettre et recevoir des données. Deux principaux modèles sont généralement utilisés pour les nodes, le SX1272 et le SX1276. Fort similaires, ils se différencient en particulier par la gamme de fréquences pouvant être utilisées avec, dans le premier cas, une plage de 850 Mhz à 1 GHz et dans le second cette même plage, plus la bande des 169 Mhz et des 433 Mhz. Dans la plupart des cas, en utilisant la bande 868 Mhz (Europe), le choix entre SX1272 et SX1276 est secondaire dans un premier temps. Ce n'est qu'avec une certaine expérience et une bonne connaissance du fonctionnement du système qu'il devient important de prendre cela en considération.

L'utilisation d'un shield est une bonne solution, mais il faudra faire très attention à la carte Arduino utilisée. En effet, l'usage de LoRaWAN est généralement lié au déploiement de capteurs divers. Rappelons au passage que LoRaWAN est prévu pour une communication longue distance (en kilomètres), avec un faible débit, des données succinctes et une faible consommation énergétique, et non pour les transmissions en temps réel, la voix ou l'image, ou encore pour le contrôle à distance. Autrement dit, si vous optez pour LoRaWAN c'est surtout parce que vous voulez déployer des capteurs...

Bien entendu, qui dit capteur dit bibliothèques et code pour gérer ces capteurs. Tout ceci, en plus de la gestion LoRaWAN, du chiffrement et du format de données consomme une place non négligeable en mémoire. À titre d'exemple, un simple capteur humidité/pression/température/lumière sur base Arduino UNO avec le shield Dragino occupera, après compilation, quelques 87% des 32Ko de flash intégrés dans le microcontrôleur (voire 110% avec certaines versions des bibliothèques).

On comprend mieux alors pourquoi la carte Seeeduino LoRaWAN avec GPS est construite non pas autour d'un Atmel AVR, mais d'un Atmel ATSAMD21G18 (comme l'Arduino Zero) à cœur ARM Cortex-M0+ avec 256 Ko de flash et 32 Ko de mémoire vive. Cette

Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)

carte, intégrée au kit Seeed pour LoRaWAN vaut une quarantaine d'euros (35€ en version sans GPS, le même prix qu'une Zero), mais intègre, en plus, un contrôleur de charge pour accu lipo et un connecteur dédié. Faire de cette carte un node autonome sera donc un jeu d'enfant et ne nécessitera rien d'autre qu'un accu adapté avec un connecteur JST2.0. Un dernier détail très important concernant cette carte concerne les tensions utilisées : maximum 3,3V, cette carte n'est pas tolérante au 5V !

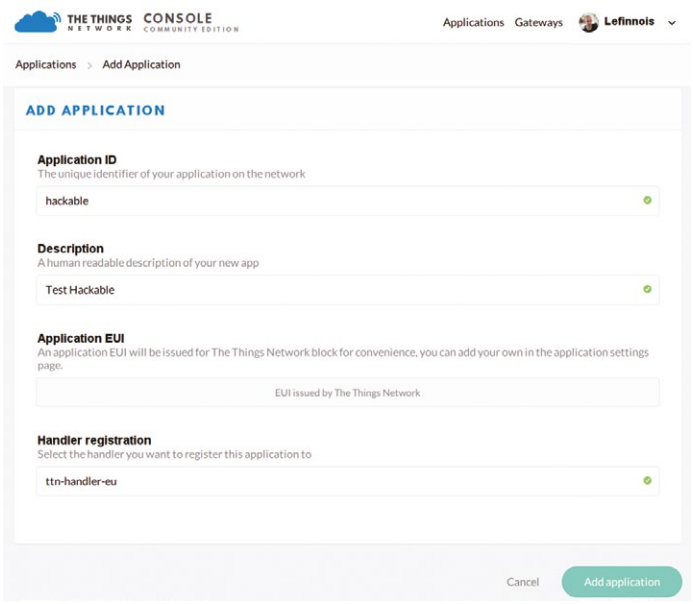
L'espace alloué pour cet article, déjà bien long, est insuffisant pour traiter à la fois du shield Dragino et de la carte Seeeduino. Cette dernière a tout de même fait l'objet d'expérimentations et vous trouverez le croquis commenté correspondant dans le dépôt GitHub du numéro (**https://github.com/Hackable-magazine**). Je ne cacherai pas une nette préférence pour le shield Arduino plutôt que pour la carte Seeeduino malgré la possibilité d'utiliser un accu facilement : la documentation est trop concise, la bibliothèque associée de piètre qualité et le contrôle de la communication et de la configuration du module trop limité à mon goût.

3. TTN ET ENREGISTREMENT DE L'APPLICATION ET DU NODE

Pour connecter un node au réseau TTN, vous devez bien entendu posséder un compte. Si vous avez déployé un concentrateur LoRaWAN, vous pouvez utiliser le compte déjà prévu à cet effet sans le moindre problème.

Une fois connecté à la console TTN (**https://console.thethingsnetwork.org/**), vous devez avant toutes choses créer une application. Ceci n'est pas une application dans le sens habituel du terme (comme une application Windows ou Android par exemple), mais davantage l'aspect applicatif de votre projet. C'est l'endroit où devront aboutir vos données. Nous verrons plus loin comment connecter cette facette à quelque chose qui ressemble à une application en utilisant Cayenne.

En ajoutant une application, vous serez invité à renseigner quelques éléments dont l'ID de l'application (alphanumérique minuscule uniquement, pas de « - » ou de « _ » consécutifs ou en début/fin), une description et le *handler* pour l'enregistrement (le serveur européen ici). L'identifiant unique de votre application (AppEUI) sera généré automatiquement par le réseau :

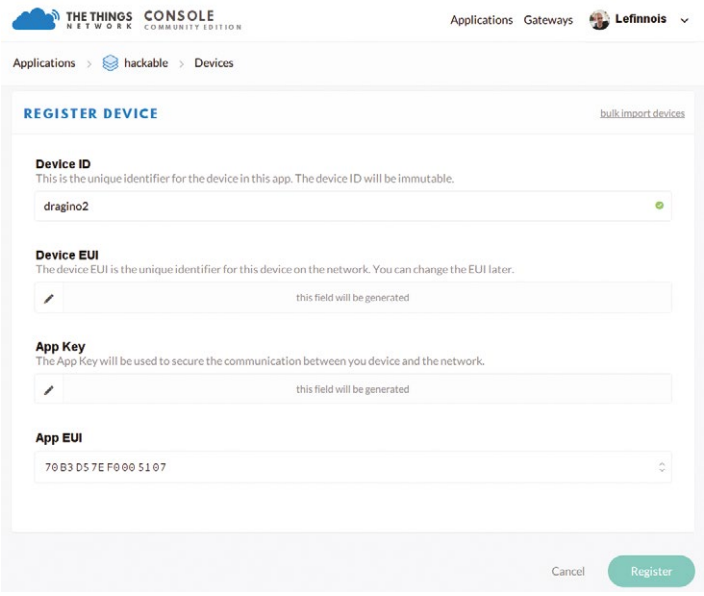


Cet identifiant unique apparaîtra dans le résumé une fois l'enregistrement validé. Vous avez maintenant le socle sur lequel va reposer votre ou vos nodes/périphériques (*devices*) et c'est dans une application que vous allez l'enregistrer. Dans le résumé de l'application, en haut à droite, vous trouverez un bouton ***Device*** vous affichant la liste des nodes gérés par l'application. Cliquez sur ***Register device*** pour en ajouter un nouveau.

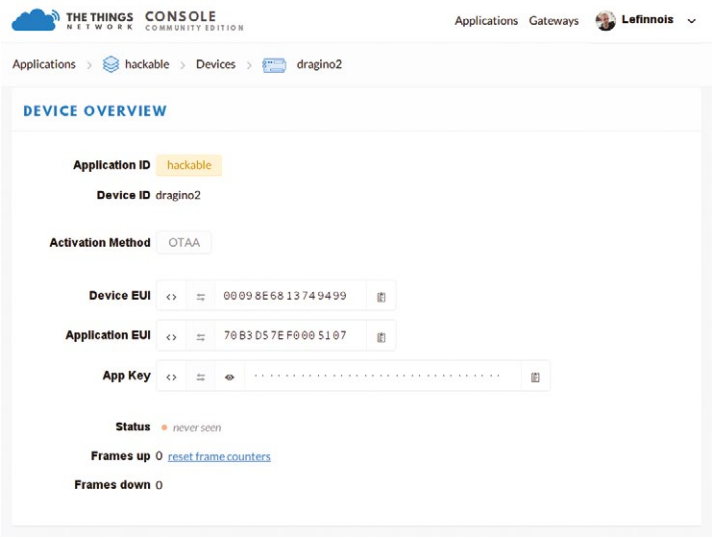
Dans le formulaire, précisez un ID pour votre node (son nom répond aux mêmes impératifs typographiques que l'ID de l'appli-

Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)

cation), c'est un simple nom qui n'est pas utilisé dans l'activation. Le ***Device EUI*** (DevEUI), en revanche, est un identifiant unique sur le réseau, vous pouvez le préciser vous-même ou tout simplement laisser le formulaire en choisir un à votre place en cliquant sur l'icône à gauche du champ de saisie. Un message précisera « *This field will be generated* » (« ce champ sera généré »). Cliquez ensuite sur ***Register*** et le tour est joué :



La page qui est affichée ensuite résume les informations pour ce node/périphérique :



La méthode d'activation par défaut est OTAA. Vous pouvez la changer en cliquant sur ***Settings*** en haut à droite, mais je ne couvrirai pas ABP ici comme précisé plus haut. En OTAA tout ce dont nous avons besoin pour composer notre futur croquis Arduino c'est l'identifiant unique du périphérique (DevEUI), l'identifiant unique de l'application (AppEUI) et la clé de l'application (AppKey), tout le reste sera généré lors de la connexion.

Pour vous faciliter les choses, l'interface web propose de formater ces informations de manière à les rendre directement utilisables dans un croquis en C/C++. C'est l'objet de l'icône « <> » à gauche des champs transformant, par exemple `00098E6813749499` en `{ 0x00, 0x09, 0x8E, 0x68, 0x13, 0x74, 0x94, 0x99 }`.

Notez également la seconde icône présentant une double-flèche permettant d'influer sur l'ordre des octets qui composent la donnée. En fonction des plateformes, modules et bibliothèques utilisées, l'ordre peut être différent et en cliquant sur cette icône vous pouvez donc réordonner les octets avant de les copier dans le presse-papier avec l'icône à droite du champ. Dans le cas du montage Arduino à base du shield Dragino, reposant sur la bibliothèque LMIC :

- l'AppEUI est en LSB (*Least Significant Byte*, le premier octet dans le sens de la lecture courante est celui valant le plus dans une valeur sur plusieurs octets) ;
- le DevEUI est également en LSB ;

Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)

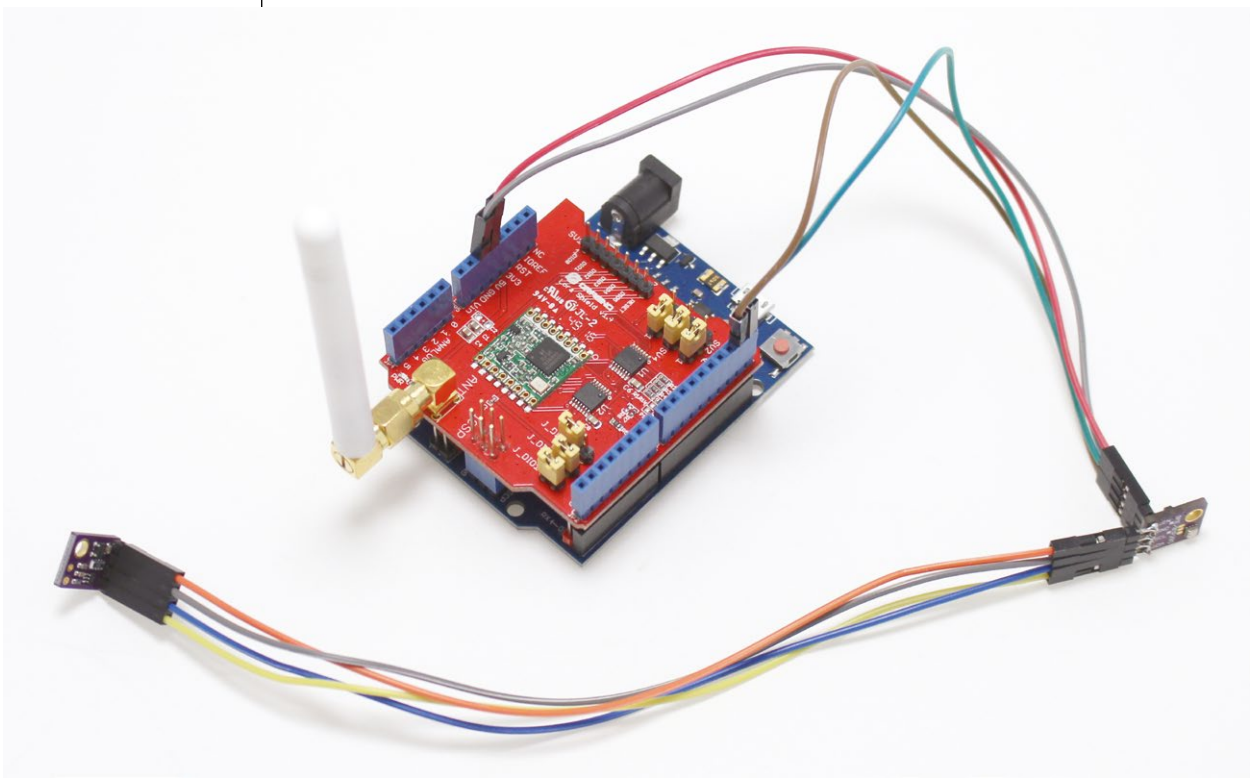
• l'AppKey est en MSB (*Most Significant Byte*), l'ordre inverse.

Il est important de bien comprendre cet ordre des octets, car dans le cas contraire, votre node tentera de se connecter à TTN avec de mauvaises informations. C'est une erreur courante et la principale source de problème des utilisateurs. Notez également au passage que tout ceci n'a rien à voir avec le concentrateur, ce n'est pas lui, mais le réseau TTN qui négocie avec votre node, le concentrateur ne fait que relayer les informations entre un média (radio LoRa) et un autre (Internet). C'est un simple répétiteur.

Une dernière étape concernant l'application concerne le choix d'un format de données. En effet, vous pouvez envoyer tout ce qui vous plaît via LoRaWAN (du moment que c'est petit et ponctuel), mais si vous comptez utiliser ces informations vous devrez vous conformer à un format de données compris par votre application. Ici, notre application (au sens « destination finale des données ») sera la plateforme Cayenne. TTN se chargera alors de renvoyer les informations à Cayenne par le biais d'un mécanisme d'intégration. Nous configurerons cela plus tard, mais devons toutefois préciser comment sont encodées les données : au format Cayenne LPP (pour *Lower Power Protocol*).

Ceci se fait très simplement en cliquant sur ***Payload Formats*** sur la page de notre application et en précisant ***Cayenne LPP*** via le menu déroulant. Après enregistrement par un clic sur ***Save***, nos données seront traitées comme étant au format en question. Ceci signifie également que notre croquis Arduino devra

Notre montage de test est relativement simple et utilise deux modules capteur en i2c. Le premier relève température, humidité relative et pression atmosphérique et le second l'indice UV et une indication de l'intensité de lumière visible. Ce sont ces informations que nous allons envoyer périodiquement à notre application via LoRaWAN et Internet.



Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)

encoder nos données dans ce format avant de les envoyer via LoRa. Notez également que le format, comme l'intégration, concerne une application dans son ensemble et donc tous les nodes qui y sont rattachés.

4. CRÉONS NOTRE NODE ARDUINO + LORA SHIELD

Tout est en place côté TTN, il ne nous reste plus qu'à utiliser ces informations pour créer notre node. Pour accéder au module LoRa sur shield Dragino, nous aurons besoin de la bibliothèque *Arduino-LMIC*. Celle-ci est installable depuis le gestionnaire de bibliothèques, mais mieux vaudra l'installer manuellement depuis **https://github.com/matthijskooijman/arduino-lmic**, car celle proposée via l'IDE est moins à jour (et occupe davantage d'espace en flash).

Nous allons utiliser deux capteurs en guise d'exemple, interfacés en i2c :

- un module BME280 mesurant température, hygrométrie et pression atmosphérique, pris en charge par la bibliothèque éponyme de *Tyler Glenn* disponible via le gestionnaire de bibliothèques ;
- un module SI1145 mesurant les rayonnements ultraviolets et la lumière ambiante visible, pris en charge par la bibliothèque Adafruit SI1145 également installable par le gestionnaire.

Enfin, nous avons besoin d'encoder nos données au format Cayenne LPP et devrons installer, via le gestionnaire, les bibliothèques TheThingsNetwork. La majeure partie de cet élément ne nous sera pas utile puisque nous ne ferons usage que de la bibliothèque *CayenneLPP* qui s'y trouve. Toute la partie matérielle concerne en effet les modules LoRa Microchip RN2xx3 dont un modèle est intégré à la « *The Things Uno* », un dérivé d'Arduino Leonardo spécialement conçu pour et par TTN.

Notre croquis débutera donc avec une belle série d'inclusions :

```
#include <lmic.h>
#include <hal/hal.h>
#include <SPI.h>
#include <BME280I2C.h>
#include <Adafruit_SI1145.h>
#include <CayenneLPP.h>
```

On définira ensuite quelques macros :

```
// Altitude
#define ALT 209
// Calcul de la correction de pression ajustée
#define COR (ALT/8.0)
// Taille maximum des messages LPP
#define MAX_SIZE 100
```

Les deux premières nous permettent de calculer, sur la base de la pression atmosphérique lue et de l'altitude où nous nous trouvons, une pression équivalente au niveau de la mer. Comme le précise un site de Météo France, en moyenne, la pression atmosphérique diminue de 1 hPa tous les 8 mètres, nous devons donc ajouter des hectopascals pour obtenir une valeur équivalente à celles utilisées conventionnellement en météo. La seconde macro fixe la taille maximum des messages au format LPP (c'est un maximum, les messages sont bien plus petits que cela).

Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)

On peut ensuite déclarer les variables et objets utilisés dans le croquis :

```
// On déclare les capteurs
Adafruit_SI1145 uv = Adafruit_SI1145();
BME280I2C bme;

// Identifiant tâche
static osjob_t sendjob;

// Émission toutes les 5 mn
const unsigned tx_interval = 300;
```

uv représente le module Silicon Labs SI1145 et **bme** le capteur Bosch BME280. **sendjob** représente une tâche (un *job*) devant être effectué et plus précisément la tâche consistant à envoyer notre message. La logique en œuvre dans la bibliothèque LMIC consiste à préparer une tâche d'envoi de message, accompagnée des données utiles et de fixer son exécution dans le temps. On ne pilote donc pas directement l'envoi comme on pourrait le faire avec une communication série ou autre. Enfin, **tx_interval** est l'intervalle en secondes entre les envois de messages, celui-ci est utilisé pour programmer la prochaine tâche une fois que la précédente est terminée.

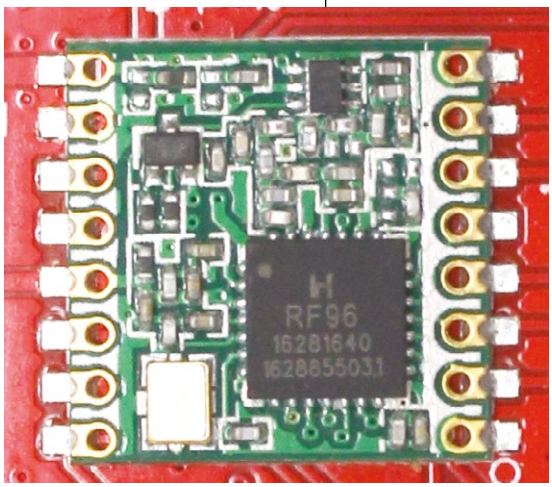
Il faut ensuite s'occuper de la connexion physique :

```
// Brochage
const lmic_pinmap lmic_pins = {
    .nss = 10,
    .rxtx = LMIC_UNUSED_PIN,
    .rst = 9,
    .dio = {2, 6, 7},
};
```

La bibliothèque LMIC n'est pas utilisable qu'avec le shield Dragino, mais également avec n'importe quel module reposant sur une puce Semtech de la famille SX1272 ou SX1276 et donc le kit d'évaluation Semtech et les modules HopeRF RFM92 et RFM95. Il est possible de l'utiliser avec presque n'importe quel matériel, à l'exclusion des modules plus « évolués » comme ceux de Microchip communiquant sur une liaison série et intégrant un microcontrôleur en complément (comme le module RHF76-052AM équipant la Seeeduino LoRaWAN et utilisant un STM32-L0 en plus de la puce SX1276).

Une puce Semtech utilise une liaison SPI pour la communication avec votre montage, mais également des broches supplémentaires DIO0 à DIO2, une ligne de reset, une ligne CS (alias NSS) comme pour n'importe quelle liaison SPI et éventuellement une broche RXTX permettant de sélectionner le mode d'utilisation de l'antenne. Dans le cas du shield Dragino, RXTX n'est pas utilisé, CS pour sélectionner le périphérique est sur 10 et le reset est sur 9. En

Les cavaliers SV2, SV4 et SV3 sur le shield Dragino permettent de choisir une connexion SPI via les broches 11, 12 et 13, ou via le connecteur ICSP. Dans le cas d'Arduino UNO, ceci n'a pas d'importance, ces broches étant reliées entre elles. Les cavaliers J_DIO5, J_DIO2 et J_DIO1 déterminent la connexion des broches DI05, DI02 et DI01 du module LoRa aux broches 8, 7 et 6 de la carte Arduino. DI00 est branché « en dur » à la broche 2.



Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)

mode LoRa, seuls DIO0 et DIO1 sont utilisés, respectivement reliés aux broches 2 et 6. Notez que trois broches doivent être déclarées, car le module peut être utilisé pour une communication autre qu'en LoRa et que la bibliothèque LMIC (mode FSK) supporte ce mode.

Tout est maintenant matériellement configuré et nous pouvons enfin nous occuper des identifiants et autres informations obtenues après enregistrement de notre node sur TTN :

```
// Identifiant d'application
// Format little-endian/LSB
static const u1_t PROGMEM appeui[8] = {0x07,0x51,0x00,0xF0,0x7E,0xD5,0xB3,0x70};
void os_getArtEui (u1_t* buf) { memcpy_P(buf, appeui, 8);}

// Identifiant de node/périphérique
// Format little-endian/LSB
static const u1_t PROGMEM deveui[8] = {0x99,0x94,0x74,0x13,0x68,0x8E,0x09,0x00};
void os_getDevEui (u1_t* buf) { memcpy_P(buf, deveui, 8);}

// Clé de l'application
// Format big-endian/MSB
static const u1_t PROGMEM appkey[16] =
  {0x00,0x01,0x02,0x03,0x04,0x05,0x06,0x07,0x08,0x09,0x0A,0x0B,0x0C,0x0D,0x0E,0x0F };
void os_getDevKey (u1_t* buf) { memcpy_P(buf, appkey, 16);}
```

Voilà qui peut être déroutant à première vue puisqu'on déclare des fonctions plutôt que d'en utiliser. Ceci vient du fait que la bibliothèque appelle ces fonctions pour paramétrer la communication LoRaWAN, d'où la présence de « **get** » dans leurs noms et non « **set** ». Nous créons donc des fonctions qui copient ces données en mémoire à partir du contenu de la variable déclarée juste dessus. Nous configurons ainsi l'identifiant de l'application, celui du node et la clé de l'application. Tout ceci est copié/collé depuis la page d'information du node dans la console TTN (attention à l'ordre des octets !).

Une autre fonction doit être déclarée par nos soins pour assurer le fonctionnement du croquis. Celle-ci est destinée à gérer les différents événements qui peuvent survenir dans l'exécution de nos demandes par le module LoRa :

```
// Gestion des événements
void onEvent(ev_t ev) {
    Serial.print(os_getTime());
    Serial.print(": ");
    switch(ev) {
        case EV_SCAN_TIMEOUT:
            Serial.println(F("EV_SCAN_TIMEOUT")); break;
        case EV_BEACON_FOUND:
            Serial.println(F("EV_BEACON_FOUND")); break;
        case EV_BEACON_MISSED:
            Serial.println(F("EV_BEACON_MISSED")); break;
        case EV_BEACON_TRACKED:
            Serial.println(F("EV_BEACON_TRACKED")); break;
        case EV_JOINING:
            Serial.println(F("EV_JOINING")); break;
        case EV_JOINED:
            Serial.println(F("EV_JOINED"));
            LMIC_setLinkCheckMode(0); break;
        case EV_RFU1:
```

Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)

```
            Serial.println(F("EV_RFU1")); break;
        case EV_JOIN_FAILED:
            Serial.println(F("EV_JOIN_FAILED")); break;
        case EV_REJOIN_FAILED:
            Serial.println(F("EV_REJOIN_FAILED")); break;
        case EV_TXCOMPLETE:
            Serial.println(F("EV_TXCOMPLETE (avec attente RX)"));
            if(LMIC.dataLen) {
                // données reçues ?
                Serial.print(F("Data RX: "));
                Serial.write(LMIC.frame+LMIC.dataBeg, LMIC.dataLen);
                Serial.println();
            }
            // Planifier la prochaine émission
            os_setTimedCallback(&sendjob, os_getTime()+sec2osticks(tx_
interval), do_send);
            break;
        case EV_LOST_TSYNC:
            Serial.println(F("EV_LOST_TSYNC")); break;
        case EV_RESET:
            Serial.println(F("EV_RESET")); break;
        case EV_RXCOMPLETE:
            Serial.println(F("EV_RXCOMPLETE")); break;
        case EV_LINK_DEAD:
            Serial.println(F("EV_LINK_DEAD")); break;
        case EV_LINK_ALIVE:
            Serial.println(F("EV_LINK_ALIVE")); break;
         default:
            Serial.println(F("inconnu")); break;
    }
}
```

Ces événements sont générés depuis l'intérieur de la bibliothèque LMIC via des appels à **reportEvent()**, elle-même appelant notre fonction ici présente. La plupart de ces événements ne sont traités qu'à titre d'information, pour suivre sur le moniteur série le déroulement des opérations. **EV_TXCOMPLETE** est particulier, il est généré dès qu'un message a été envoyé et nous l'utilisons pour pouvoir planifier un nouvel envoi avec **os_setTimedCallback()**. On précise en argument la tâche dont il s'agit, le moment où elle doit être exécutée (maintenant + **tx_interval**) et la fonction utilisée pour générer les données du message. Chaque fin de transmission provoquera donc la création d'une nouvelle tâche, exécutée plus tard.

Ceci nous conduit donc naturellement à l'écriture de **do_send()**, la fonction formant la partie « active » de notre node :

```
// Notre tâche
void do_send(osjob_t* j){
  // variable pour les capteurs
  float temp, hum, pres, uvindex, lum;

  // Communication déjà en cours ?
  if (LMIC.opmode & OP_TXRXPEND) {
    Serial.println(F("OP_TXRXPEND, pas d'envoi"));
  } else {
    // lecture des capteurs
```

Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)

```
    bme.read(pres, temp, hum, true, B001);
    uvindex = uv.readUV()/100.0;
    // Attention ! Il ne s'agit pas de lux comme avec un TSL2561
    lum = uv.readVisible();

    // Affichage des valeurs
    Serial.print("uv: ");
    Serial.print(uvindex);
    Serial.print("\tlum: ");
    Serial.print(lum);
    Serial.print("\ttemp: ");
    Serial.print(temp);
    Serial.print("\thum: ");
    Serial.print(hum);
    Serial.print("\tpress: ");
    Serial.println(pres+COR);

    // Composition du message
    lpp.reset();
    lpp.addTemperature(0, temp);
    lpp.addRelativeHumidity(1, hum);
    lpp.addBarometricPressure(2, pres+COR);
    lpp.addLuminosity(3,(int)uvindex);
    lpp.addLuminosity(4,(int)lum);

    // Enregistrement du message à envoyer
    LMIC_setTxData2(1, lpp.getBuffer(), lpp.getSize(), 0);
    Serial.println(F("Packet queued"));
  }
}
```

Pour relever les informations pertinentes, nous utilisons les méthodes de chaque bibliothèque prenant en charge les capteurs. Le point important est cependant tout autre : la construction de notre message au format Cayenne LPP. Nous commençons par réinitialiser notre variable **`lpp`** avec la méthode **`reset()`** afin de partir sur un message vide. Nous ajoutons ensuite, élément par élément, les données collectées en spécifiant un canal en premier argument (celui-ci sera utilisé plus tard sur Cayenne).

Enfin, nous utilisons **`LMIC_setTxData2()`** pour définir les données à envoyer en précisant où elles se trouvent (**`lpp.getBuffer()`** retourne l'adresse) et leur taille. Comprenez bien que ceci ne provoque pas l'envoi, nous ne faisons que « charger » le contenu du futur message, un peu comme un boulet dans un canon.

À présent que les macros, variables, objets et fonctions de notre croquis sont en place, nous pouvons nous occuper de **`setup()`** :

```
// configuration
void setup() {
  Serial.begin(115200);
  Serial.println(F("Go Go Go !"));

  // initialisation capteurs
```

Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)

```
  while (!bme.begin()) {
    Serial.println("Erreur BME280! !");
    delay(2000);
  }
  while (!uv.begin()) {
    Serial.println("Erreur SI1145 !");
    delay(2000);
  }

  // LMIC init
  os_init();
  LMIC_reset();

  // Démarrage tâche
  do_send(&sendjob);
}
```

La partie intéressante est celle se situant, bien entendu, après l'initialisation des capteurs. On initialise le support LoRa/LoRaWAN avec `os_init()` qui lui-même, dans la bibliothèque initialise tous les éléments de la communication (mémoire, matériel, radio), puis on déclenche un reset pour supprimer toutes données en attente de traitement et stopper toutes actions en cours. Enfin, comme l'envoi répétitif de messages se base sur une tâche planifiée par la fin d'un précédent envoi, il est nécessaire d'appeler une première fois `do_send()`.

Le reste du croquis tient en une simple fonction `loop()` se contentant d'appeler constamment la fonction de gestion générale :

```
void loop() {
  // boucle de gestion
  os_runloop_once();
}
```

Une fois le croquis programmé dans la carte, son exécution devrait rapidement découler à l'arrivée d'un premier message sur TTN. Dans le moniteur série, « `EV_JOINING` » devrait apparaître, puis « `EV_JOINED` » et enfin « `Packet queued` », puis « `EV_TXCOMPLETE` ». En vous rendant sur la console TTN, sur la page de votre node (*device*), un clic sur ***Data*** devrait alors montrer ce premier message, puis des suivants (ceux-ci apparaissent également dans ***Data*** de l'application). Il en va de même, si vous avez et utilisez votre propre concentrateur, à la différence que le contenu n'est pas lisible, car chiffré.

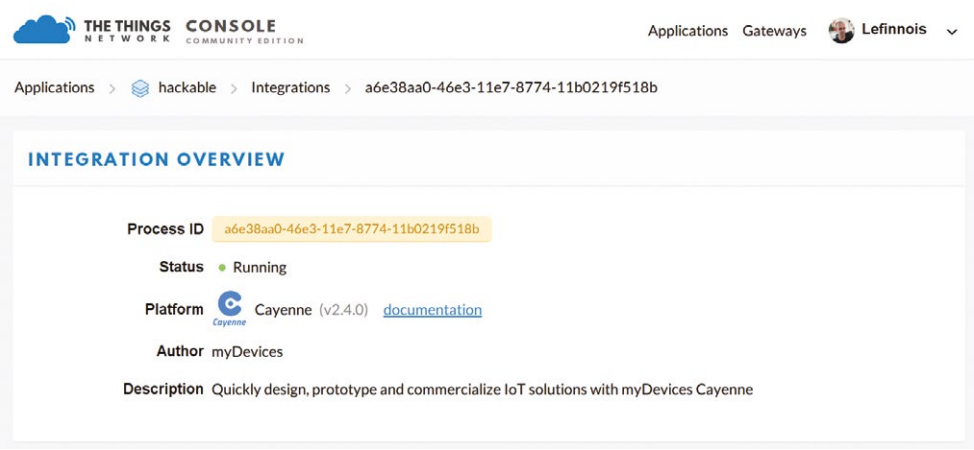
5. LES DONNÉES SONT LÀ, FAISONS-EN QUELQUE CHOSE

Sur la page de l'application et du périphérique côté console TTN, nous pouvons voir les données décodées et en clair. Sur ce point, il faut bien comprendre qu'une partie de la console TTN présente également le côté « Application » d'un réseau LoRaWAN et de ce fait, les données sont déchiffrées. Ceci vous permettra de valider le contenu de vos messages, comme ici, en lisant directement les valeurs présentes dans les données LPP.

Vous conviendrez cependant avec moi que ceci n'est pas très attrayant pour le commun des mortels. Pour réellement impressionner vos amis ou vos collègues, il est préférable de présenter ces informations de façon un peu plus évidente en utilisant une intégration. En vous rendant sur la page de votre application dans la console TTN, cliquez sur ***Integration*** et ***Add integration***. Là vous aurez le choix entre plusieurs options vous permettant de « faire quelque chose » avec les données des messages. À cette date, quatre options sont possibles et nous allons utiliser la plus simple : Cayenne.

Cayenne MyDevice est un site et une application Android/iOS permettant de centraliser la gestion des objets connectés. Vous pouvez y afficher et stocker des données de capteurs, piloter des montages connectés et analyser

Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)

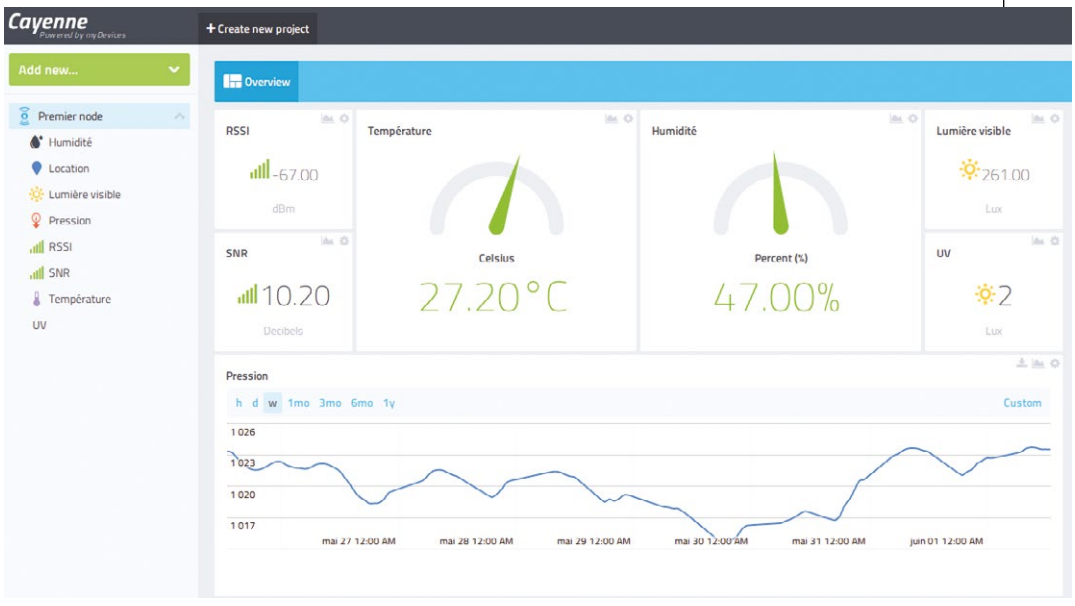


Le lien entre TTN et Cayenne se fait par le biais d'une « intégration » (au sens mathématique du terme), un joli mot pour dire que les données sont « intégrées » ou accumulées quelque part.

leur fonctionnement. Cayenne n'est pas spécifique à LoRaWAN, mais permet une gestion d'objets connectés de plusieurs façons (Wifi, Ethernet, 3G, etc.). LoRaWAN avec TTN n'est qu'un moyen parmi plusieurs. Notez également que Cayenne n'est pas le seul site à proposer ce genre de choses, mais il est fort pratique pour obtenir un premier résultat présentable.

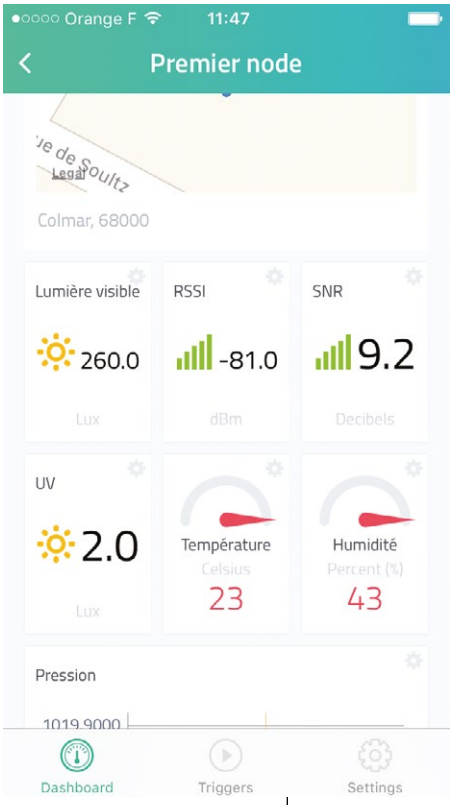
Vous devrez commencer par créer gratuitement un compte sur **https://cayenne.mydevices.com** et ce faisant vous arriverez directement sur une page vous permettant de sélectionner un périphérique pour créer un projet. Choisissez alors ***LoRa*** (en bêta actuellement), ***The Things Network*** (à gauche) et dans la liste à droite choisissez ***Cayenne LPP*** puis renseignez les informations utiles :

- ***Name*** : le nom du périphérique/node ;
- ***DevEUI*** : l'identifiant unique du node, à copier/coller depuis la console TTT ;



Dès que les informations de votre premier node arrivent sur Cayenne via TTN et un concentrateur LoRaWAN, vous devrez les voir apparaître dans votre dashbboard. Vous pourrez alors configurer la manière dont ces informations doivent être présentées...

Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)



Cayenne n'est pas spécifique à LoRaWAN, à TTN ou même au Web. C'est une solution prenant également la forme d'une application Android ou iOS (iPhone) destinée à représenter les informations de plusieurs types de capteurs communiquant de différentes façons.

- ***Activation Mode*** : le mode d'activation du node, ici seul ***Already Registered*** (déjà enregistré) est utilisable, ça tombe bien parce que c'est le cas ;
- ***Location*** : choisissez ***This device doesn't move*** et spécifiez une adresse pour le placer sur la carte OpenStreetMap.

Valider le formulaire et le *Dashboard* devrait apparaître vous présentant une carte avec votre node. Revenez ensuite sur la console et choisissez une intégration avec Cayenne. Le « *Process ID* » doit être la longue chaîne de caractères se trouvant dans l'URL du dashboard Cayenne, juste après « /lora/ », vous devez copier/coller ce texte puis choisir ***Default key*** dans ***Access Key*** avant de valider. Ceci fait, TTN commencera automatiquement à relayer les données des messages de votre node à Cayenne et les informations devraient alors apparaître dans votre dashboard.

Ce que vous voyez apparaître dans votre navigateur ce sont les mesures faites par votre node, envoyées via les airs en LoRa à un concentrateur LoRaWAN, transitant par TTN pour finalement arriver sur Cayenne. Si vous installez l'application Android ou iOS, vous verrez les mêmes informations, présentées peu ou prou de la même façon...

6. LIMITATIONS, USAGE RAISONNÉ ET SERVEUR PRIVÉ

Nous avons couvert ici la base de l'utilisation de LoRaWAN et avec l'article précédent (et un certain budget) vous devriez être en mesure de déployer vos nodes avec ou sans concentrateur déjà en place dans votre voisinage. TTN n'est pas le seul fournisseur existant, mais il est à mon sens le plus intéressant, car réellement communautaire.

Mais avec cette notion de partage des concentrateurs et cette liberté d'utilisation vient également l'importance d'un comportement raisonnable. Le fonctionnement même de LoRaWAN, les canaux, le débit, le rapport cyclique pour chaque canal et le nombre de nodes supportés par chaque concentrateur, implique une politique de partage claire et stricte. L'élément de base pour cet usage raisonné est le « temps dans l'air » (*air-time*), autrement dit le temps total de communication radio entre un node et un concentrateur. La règle de base est de ne pas dépasser 30 secondes sur une période de 24h.

Le calcul de ce temps quotidien n'est pas simple puisqu'il dépend de la taille des messages et de la vitesse de transmission lui même dépendant de la qualité du signal et donc de la distance entre le node et le concentrateur. Le débit est déterminé par la notion de facteur d'étalement ou *spreading factor* en anglais. Noté de SF7 à SF12, plus le facteur d'étalement est important, plus la distance peut être grande, mais plus la communication est lente, et donc de ce fait, le temps « dans l'air » est important. À titre d'exemple avec un message de 10 octets et un facteur d'étalement SF12, la limite d'utilisation raisonnée est de 20 messages par jour, mais sera de 500 en SF7.

Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)

La communication *downlink* (du concentrateur vers un node) est encode plus réduite puisque la limite est fixée à 10 messages par jour. Tout ceci permet d'assurer le support de 1000 nodes par concentrateur et un fonctionnement optimal du réseau. Si cela ne vous convient pas, rien ne vous empêche de vous passer de TTN et de faire fonctionner votre propre serveur réseau, TTN met à disposition les sources et les documentations utiles pour cela. Mais vous perdrez alors l'idée même derrière LoRaWAN et la construction d'un réseau complet communautaire, alors que le but est justement de permettre à des utilisateurs ne pouvant pas installer un concentrateur de bénéficier de LoRaWAN dans leur zone.

Côté application, nous avons vu qu'il était possible très simplement d'utiliser l'intégration pour présenter les données dans une interface web ou une application mobile avec Cayenne. Là encore, de nombreuses alternatives existent aussi bien sous la forme de services qu'avec des solutions « maison » via une intégration HTTP (les données sont envoyées avec des requêtes **GET** ou **POST** à un serveur web).

Enfin, un point que je n'ai pas du tout abordé, car l'idée était ici de ne se pencher que sur LoRaWAN, est le fait de tout simplement communiquer entre deux points en utilisant LoRA. Un exemple de ce type d'usage est disponible avec la bibliothèque LMIC et pourra avantageusement remplacer une autre forme de communication comme l'utilisation de module nRF24L01.

Dragino propose également un hat pour Raspberry Pi utilisant le même module LoRa, accompagné d'un module GPS complémentaire (livré sans antenne). Ce type de carte additionnelle permet de créer un node, mais également un pseudo-concentrateur LoRaWAN à un seul canal (qui ne sera donc pas à proprement parler un concentrateur compatible).

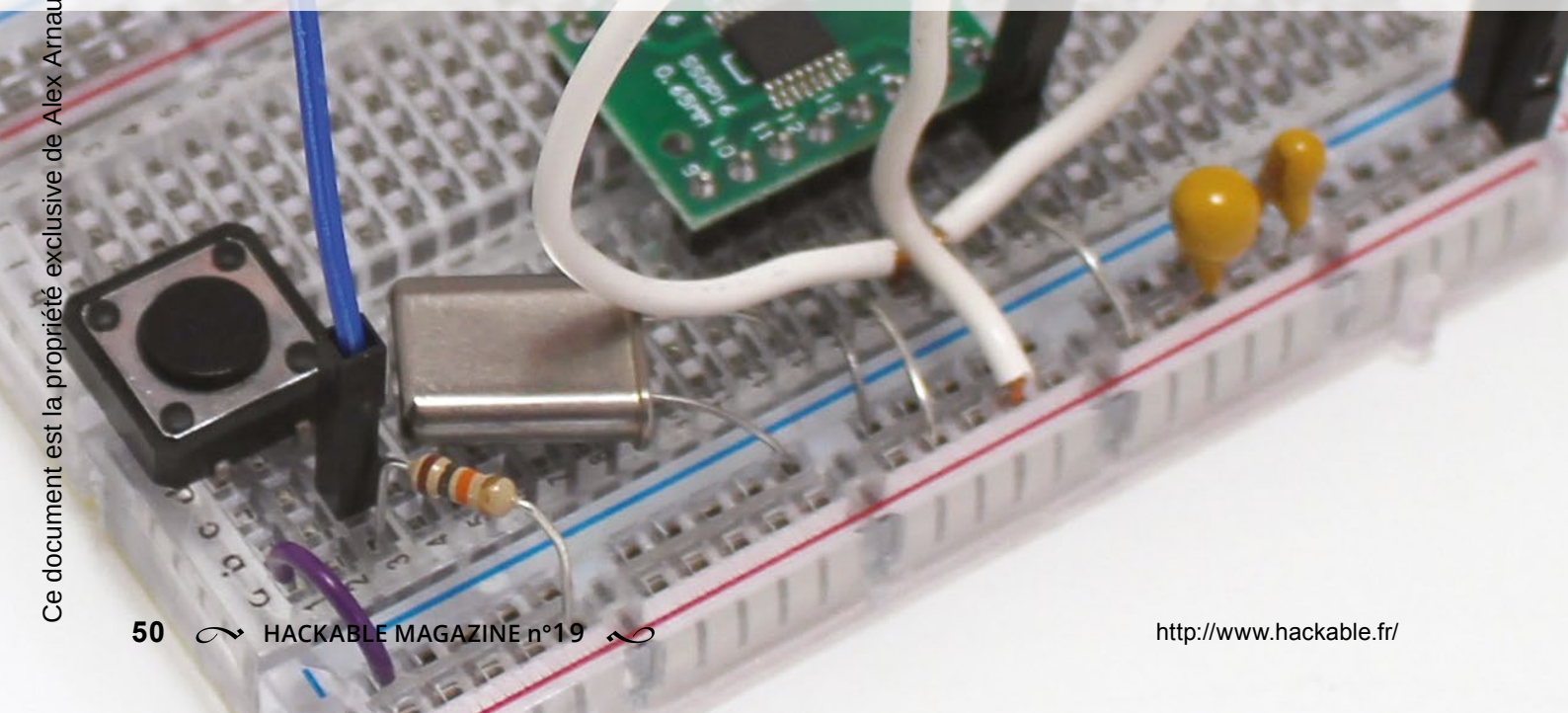
LoRa et LoRaWAN sont deux technologies qui gagnent rapidement du terrain et se popularisent à vitesse grand V, tout autant que TTN partout dans le monde. Au point qu'il est tout à fait raisonnable de penser que ceci deviendra le standard de fait avec une couverture importante d'ici peu de temps. Vous êtes, comme moi, aux premières loges et avez désormais les bases pour vous lancer... **DB**

Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)

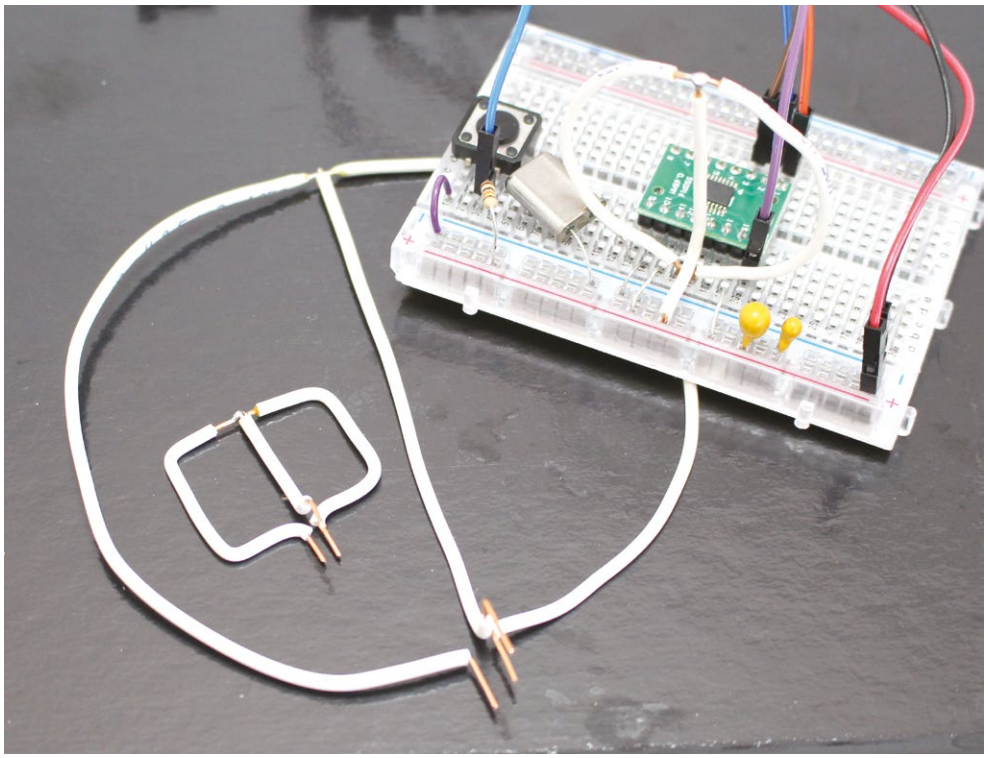
CONSTRUISEZ UN ÉMETTEUR 433 MHZ POUR REMPLACER VOS TÉLÉCOMMANDES

Denis Bodor

Dans le 12ème numéro de *Hackable*, j'avais exposé l'utilisation d'une clé USB TNT pour capter et analyser les signaux radio d'une télécommande à fréquence ainsi que le détournement d'un circuit d'une autre télécommande afin d'envoyer ces mêmes signaux via carte Arduino. Ce bricolage imposait l'utilisation d'une télécommande identique et vous avez été nombreux à vouloir en savoir davantage sur le sujet. Voyons donc à présent comment, à partir d'un hack, obtenir quelque chose d'un peu plus générique et utilisable.



Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)



La conception et la fabrication d'antennes, quels que soit leurs types ou leurs usages, est un vaste et complexe domaine nécessitant des connaissances et une certaine expérience. Lorsqu'on n'en dispose pas, la solution la plus rapide (et très certainement la moins efficace) consiste à tout simplement tenter de copier ce qui existe puis à vérifier le résultat.

L'objet du présent article n'est pas tant de créer un module d'émission radio piloté par Arduino que d'étudier comment il est possible, en partant de ce qui est clairement un bricolage, d'arriver à un circuit complet et reproductible. Le hack que j'avais utilisé reposait sur les entrailles des restes d'une télécommande de source inconnue (l'objet traînait simplement dans une de mes boites « ne pas jeter ») et consistait à utiliser un circuit construit autour d'un composant si4021 de *Silicon Labs* (gamme de produits *EZRadio*).

Ces « restes » réutilisés, après légère modification (retrait d'une résistance), regroupaient la plupart des éléments nécessaires, et ce en offrant une connectique adaptée à une utilisation avec une carte Arduino. Inutile de le nier, ceci était un coup de chance puisque sans cela, il aurait été difficile de rapidement arriver au résultat attendu. Vraiment ? Pas si sûr, il existe en effet des solutions permettant de prendre les composants individuellement et de les rassembler sur une platine à essai pour reproduire l'élément disponible clé en main.

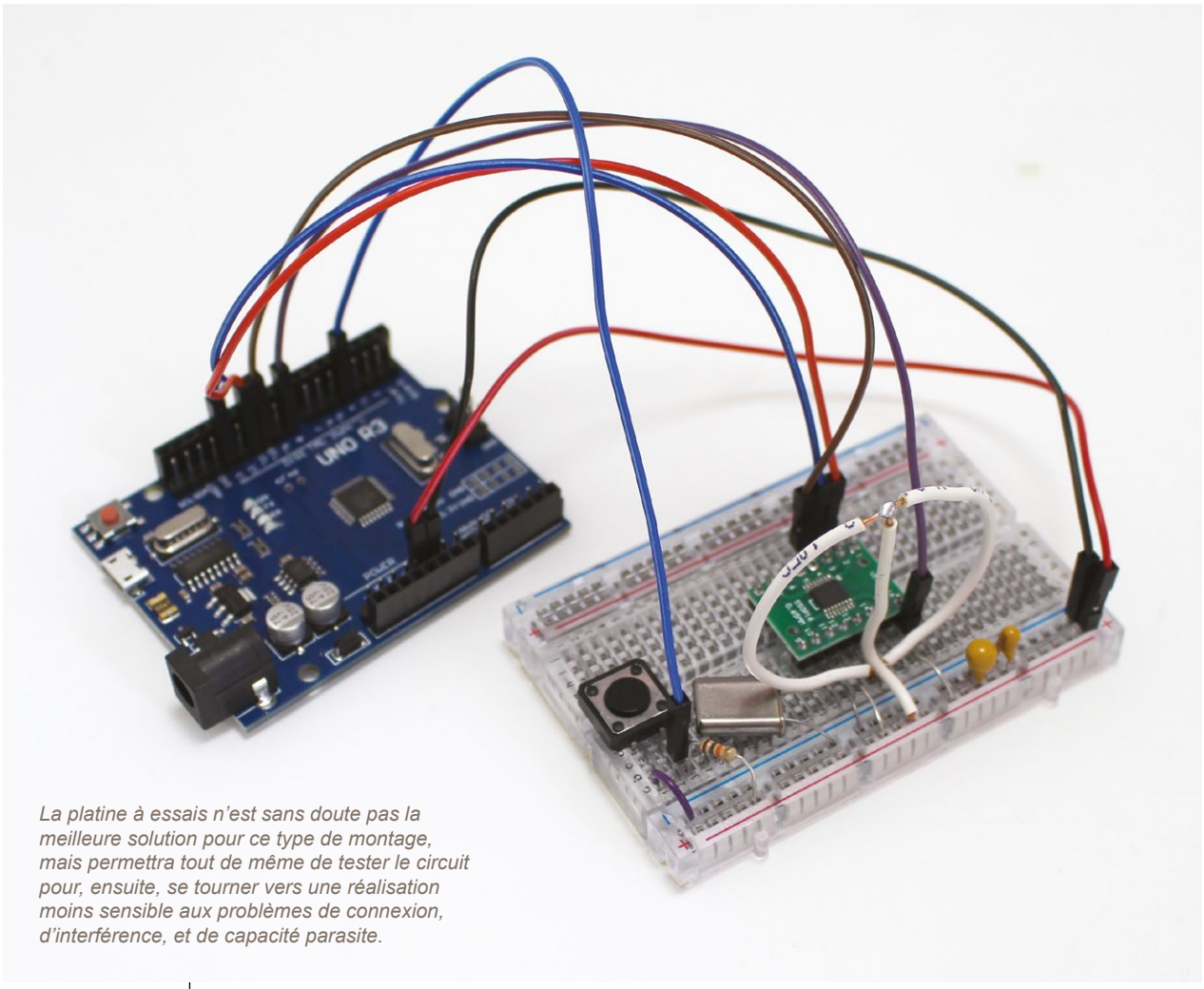
La logique est donc la suivante : on commence par réutiliser comme on peut des éléments rapportés, on analyse le fonctionnement et on se documente sur le composant central, on reproduit le circuit initial pour créer un prototype identique et, éventuellement, on en fait un circuit bien propre pour disposer d'un « module » générique et reproductible. La partie qui nous intéresse ici est le prototypage sur platine à essai.

1. DANS LES ÉPISODES PRÉCÉDENTS…

Je vais rapidement revenir ici sur ce que j'avais couvert dans le numéro 12 afin de rafraîchir vos mémoires et/ou ne pas exclure les lecteurs ayant raté ces fameux articles.

Bon nombre de télécommandes à fréquence et de dispositifs simples de communication radio utilisent les bandes ISM (pour Industrielle, Scientifique et Médicale). Il s'agit de plages de fréquences utilisables sans licence spécifique pour les dispositifs et appareils utilisés dans le

Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)



La platine à essais n'est sans doute pas la meilleure solution pour ce type de montage, mais permettra tout de même de tester le circuit pour, ensuite, se tourner vers une réalisation moins sensible aux problèmes de connexion, d'interférence, et de capacité parasite.

domaine domestique. Parmi ces plages de fréquences, on trouve celle allant de 433,05 Mhz à 434,79 MHz. Bon nombre d'équipements (télécommandes, capteurs divers, sondes météo, prises de courant intelligentes, etc.) utilisent une fréquence proche de 433,92 Mhz.

En utilisant un récepteur RTL-SDR, qui n'est autre qu'une clé USB de réception TNT (DVB-T) à l'origine, et avec les logiciels adaptés sous Windows, macOS ou GNU/Linux, il est possible de capter et d'étudier les signaux radio émis par de nombreux équipements et dispositifs. Certains d'entre eux sont relativement complexes, d'autres plus simples et enfin, d'autres, facilement accessibles sans même avoir besoin de se documenter grandement sur le sujet.

Depuis l'arrivée en masse de ces récepteurs DVB-T pour un usage radio amateur, et plus exactement ce qu'on appelle la radio logicielle, tout le domaine est devenu bien plus accessible et les solutions pour hobbyistes ont énormément gagné en maturité. Il existe maintenant des applications et outils très stables pour toutes sortes d'usages, de l'étude des signaux eux-mêmes à des applications plus précises comme la réception de messages d'aéronefs (ADS-B), la simple réception de radios FM, les applications en radio-astronomie, la réception de messages de pagers, etc.

Dans le numéro 12 du magazine, nous avons utilisé un récep-

Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)

teur de ce type pour détecter, capturer et analyser les messages d'une télécommande de prise de courant grâce à un outil appelé `rtl_433` (**https://github.com/merbanan/rtl_433**) sur Raspberry Pi (ou un PC GNU/Linux). Ce petit programme utilise un récepteur RTL-SDR afin de capter les messages de quelques 80 équipements, généralement utilisant la fréquence de 433,92 Mhz, et affiche les résultats interprétés lorsqu'un message est reçu. Il peut s'agir d'un bouton appuyé sur une télécommande ou une sonnette, une information d'un capteur de température ou encore, par exemple, un message concernant une installation de suivi de consommation électrique.

Mais `rtl_433` va plus loin. Lorsqu'un message est reçu, mais ne peut pas être interprété, car le périphérique qui l'a envoyé n'est pas connu, il est possible d'afficher des informations complémentaires concernant le signal et ainsi découvrir les données qui le composent. On peut donc littéralement voir les données transportées et les utiliser de la façon qui nous chante.

Dans l'article du numéro 12, nous avons utilisé cette fonctionnalité pour obtenir le message binaire correspondant à chaque pression de bouton sur une télécommande et avons ensuite réutilisé ces données pour envoyer les mêmes messages à partir d'un montage « maison » constitué d'une carte Arduino et de l'élément récupéré d'une autre télécommande.

Ceci a été rendu possible par le fait que ce type de produit est généralement dédié à une tâche précise, mais les composants qui en forme le cœur sont génériques et souvent présents dans différents modèles similaires, toutes marques confondues. Le composant en question est ici une puce si4021 de chez *Silicon Labs*.

Cette puce peut être utilisée dans plusieurs gammes de fréquences (433 Mhz, 868 Mhz, 915 Mhz) et dans deux modes : FSK et OOK. FSK pour *Frequency-Shift Keying* ou modulation par déplacement de fréquence, permet d'envoyer des données binaires (0 et 1) en faisant varier la fréquence du signal de 30 à 210 kHz par rapport à la fréquence de base (porteuse). Ceci est rarement utilisé avec des systèmes simples comme des télécommandes où OOK est plus populaire. OOK signifie *On-off keying*, c'est une modulation d'amplitude très simple où ce sont la présence et l'absence de signaux à la fréquence définie qui forment les données binaires (un peu comme du morse, mais avec la présence/absence de signal et non des blancs, des courts et des longs).

Dans le cas qui nous intéresse ici, comme pour la plupart des communications de ce type, la fréquence est sur la plage de 433 Mhz et en OOK. Le si4021 possède également deux modes de fonctionnement : en mode EEPROM où en mode microcontrôleur. Le premier mode utilise une simple mémoire SPI et le si4021 se débrouille seul sans partie « active », et le second fait du si4021 un simple outil de communication piloté par un microcontrôleur (comme celui d'une carte Arduino). C'est ce second mode qui est utilisé ici.

2. RECONSTRUIRE APRÈS AVOIR RÉUTILISÉ

Je ne m'étendrais pas davantage concernant la réalisation elle-même si ce n'est en précisant qu'on utilise `rtl_433` avec l'option `-a` pour obtenir les données de chaque bouton et qu'on réutilisera celles-ci directement dans un croquis Arduino reposant sur la bibliothèque `RCSwitch`. Ce croquis a pour tâche de configurer le si4021 (fréquence utilisée, paramètres, mode, etc.) et une sortie de la carte Arduino est utilisée pour activer/désactiver l'émission du signal pour former les 1 et les 0. Ceci est le travail de `RCSwitch` alors que la configuration du si4021 en SPI est faite « manuellement ».

Dans le précédent article, nous avons récupéré un morceau de circuit « tout fait » que nous avons simplement mis en œuvre pour une utilisation différente. C'est très bien, cela fonctionne, mais... que faire si nous voulons reproduire

Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)

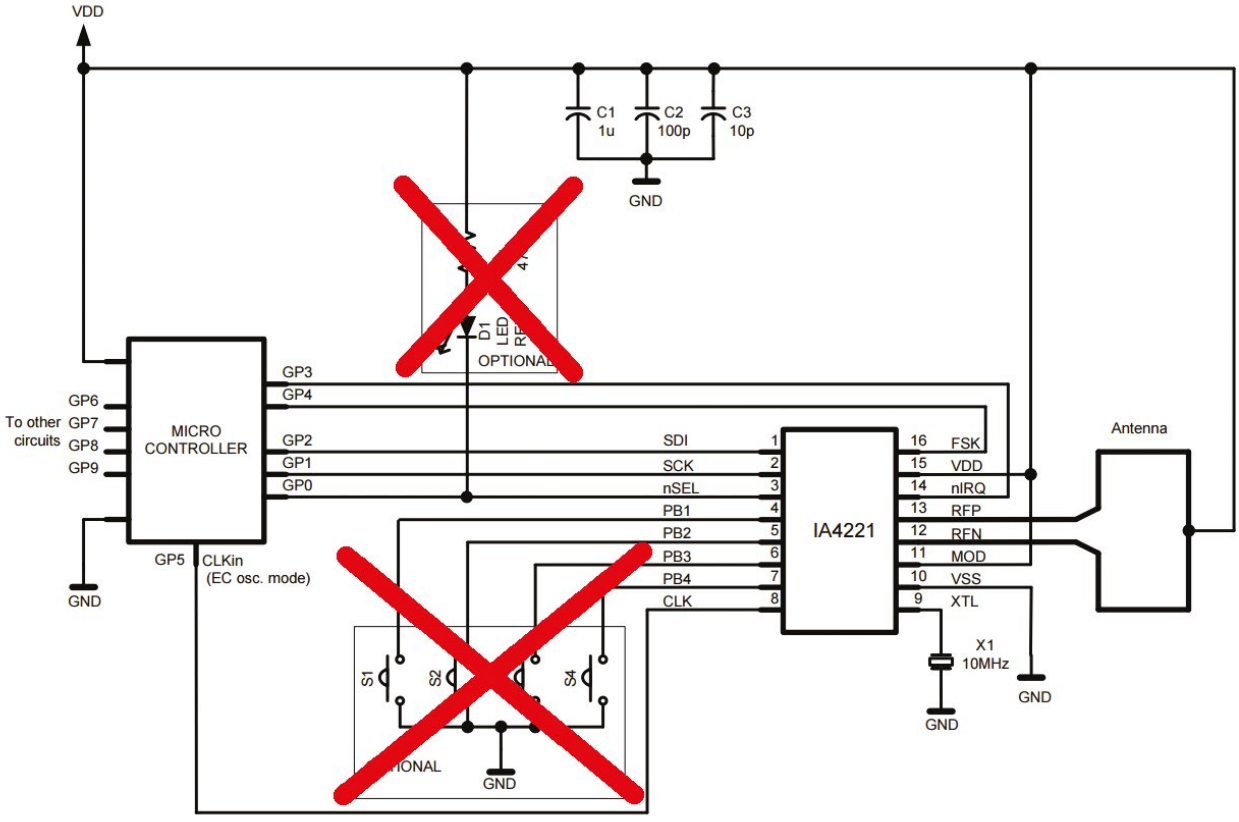
le montage ? Trouver d'autres entrailles de télécommandes ? Ceci semble assez difficile... L'approche et l'objet de cet article, consisteront donc à se documenter et à reproduire le circuit en question, du mieux possible (et ce sera loin d'être parfait).

Vous l'avez compris, avant toutes choses, il faut mettre la main sur des si4021. Fort heureusement, ceux-ci se trouvent assez facilement en ligne (sur eBay par exemple, mais aussi Banggood, Aliexpress, Etsy, Tindie, etc.). J'ai trouvé les miens sur eBay auprès d'un vendeur appelé *liaoxiyuan* pour 25€ pour 10 pièces (avec 6€ de port). Ce type de composant n'est pas vraiment monnaie courante, mais reste relativement facile à trouver en cherchant sur plusieurs sites.

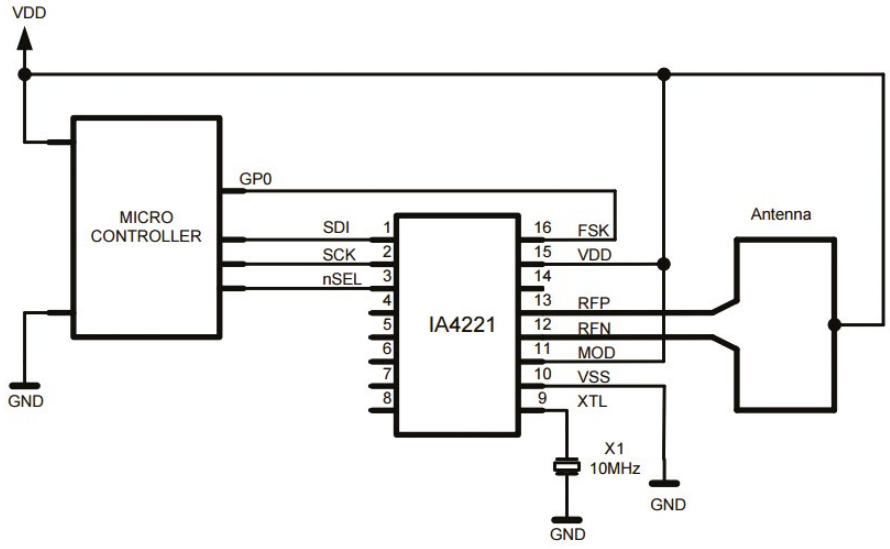
En attendant patiemment la livraison on peut ensuite se pencher sur la documentation technique (*datasheet*) regroupant toutes les spécifications et informations sur la manière d'utiliser le composant, mais aussi, et surtout présentant des applications typiques aussi bien pour le mode EEPROM que pour le mode microcontrôleur. On trouve ainsi un sympathique schéma correspondant peu ou prou à ce qui est utilisé sur le circuit précédemment récupéré.

Ce schéma présente une utilisation complète des fonctionnalités du composant. Nous n'avons pas besoin de tout cela et pouvons donc simplifier les choses. La led connectée à la broche nSEL du si4021 par exemple ne fait que signaler le passage à l'état bas de la ligne CS (*Chip Select*) de la connexion

La documentation technique du si4021 propose une mise en œuvre typique avec un microcontrôleur. Nous pouvons nous en inspirer pour reproduire le circuit qui nous intéresse, tout en écartant les fonctionnalités qui n'ont aucune importance.



Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)



En simplifiant à l'extrême le circuit proposé, les choses deviennent drastiquement plus claires. Nous n'avons pas besoin des boutons, ni de la led sur nSEL (/CS), ni de la broche nIRQ, ni même du signal CLK provenant du si4021 à destination du microcontrôleur. Ici, j'ai même retiré du schéma les condensateurs de filtrage, partant du principe que notre tension d'alimentation sera bien propre et fournie par la carte Arduino (un ou deux condensateurs placés sur la platine à essais sont néanmoins de bon ton).

SPI. En d'autres termes, elle indique lorsqu'on communique avec le circuit intégré. Ce n'est pas utile. Pas plus d'ailleurs que les boutons-poussoirs connectés à PB1, PB2, PB3 et PB4. Nous pilotons directement l'émission via la broche FSK et n'avons que faire des boutons.

Nous avons ensuite la broche CLK destinée à fournir au microcontrôleur un signal d'horloge configurable entre 1 et 10 Mhz, inutile ici. Et enfin nous avons la broche nIRQ qui est surtout utilisée en rapport avec les changements d'état du si4021 et en particulier la mise en sommeil. Là encore, ceci n'a pas grand intérêt pour nous pour le type d'utilisation que nous envisageons.

En retirant du schéma proposé tout ce qui n'est pas strictement utile pour notre application, les choses deviennent bien plus simples. Au final, nous n'avons besoin que de quatre connexions entre la carte Arduino et le si4021 (en plus de l'alimentation et la masse) : trois pour la communication SPI (MOSI, SCK et CS) et une pour contrôler l'émission du signal (en OOK).

Seuls trois « problèmes » doivent alors être réglés : la connexion physique du composant, l'ajout d'un oscillateur à quartz et l'antenne.

2.1 Adaptation du composant

Lorsqu'on parle d'électronique hobbyiste, la plupart des composants utilisés sont généralement du type traversant (ou traditionnels), car ceux-ci peuvent facilement être utilisés lors du prototypage sur platine à essais et ensuite, dans une réalisation finale, sur plaques pastillées ou à bandes. De taille assez conséquente, ils sont donc parfaitement adaptés pour ce type d'usage.

Lorsqu'on parle cependant d'électronique professionnelle ou de réalisation industrielle, ce type de format est presque totalement écarté à présent, au bénéfice de composants montés en surface (SMD ou CMS). Cette transition, qui s'est déroulée en plusieurs années, présente de nombreux avantages : des composants et donc des circuits plus petits (intégration), il y a moins d'étapes pour la réalisation (peu de trous à percer), il y a plus d'automatisation de la fabrication, ce qui induit une réduction des coûts, etc. Seuls problèmes : la maintenance et les réparations sont plus difficiles, et l'utilisation hobbyiste nécessite davantage d'équipement si on veut travailler confortablement (four à refusion, station à air chaud, etc.). Il reste possible d'utiliser ces composants, jusqu'à une certaine taille, avec un simple fer à souder ou une station de soudage, mais

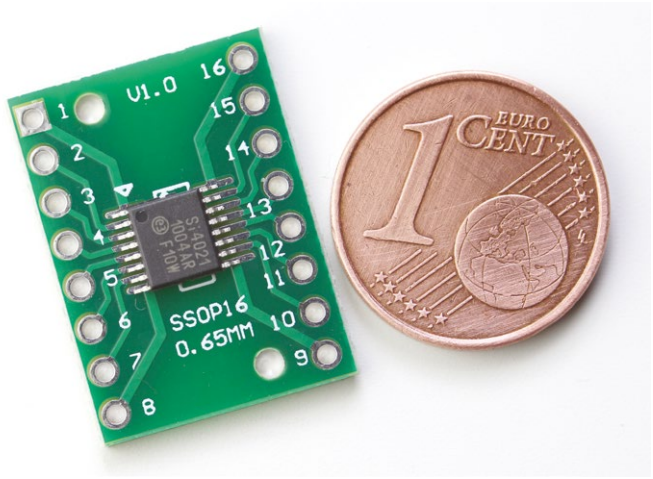
Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)

ceci demande de la minutie, de la précision et un peu de pratique. En cas d'utilisation courante de ces formats, cela conduira forcément, à un moment, à l'achat (ou la construction) d'équipements plus adaptés.

Dans le cas qui nous intéresse ici, nous n'avons pas le choix du format de composant, le si4021 n'est disponible qu'en TSSOP16 (pour *Thin-Shrink Small Outline Package*, 16 broches). Ce format utilise des broches entre 0,19 et 0,30 mm de large dont le centre est espacé de 0,65 mm. Le si4021, dans son ensemble avec ses pattes, ne fait que 6,4 mm sur 5 mm. Il est possible, à cette échelle, de s'amuser à souder des fils sur les pattes qu'on connectera ensuite à des broches au pas de 2,54 mm, mais il y a bien plus simple...

La solution consiste à utiliser un petit circuit imprimé, généralement double face, proposant un emplacement au format du composant au centre et des emplacements à souder au pas de 2,54 mm sur les côtés, donnant ainsi un format DIP (*Dual In-line Package*). Il existe de nombreux modèles de circuits imprimés permettant ainsi d'adapter plus ou moins tous types de formats de composants. Généralement, chacun d'eux supporte deux formats différents, un sur chaque face, et fait donc du composant un ensemble utilisable sur platine à essai.

Dans le cas du si4021, le circuit imprimé à utiliser est un adaptateur TSSOP16 vers DIP16, qui accessoirement peut aussi, de l'autre côté, servir d'adaptateur SSOP16 vers DIP16. J'ai acheté les miens sur eBay auprès d'un vendeur appelé *happy-zp* pour trois malheureux euros (port offert) pour 5 pièces.



Les petits circuits imprimés permettant l'adaptation des formats de composants sont sans le moindre doute l'approche la plus efficace et rapide si on a affaire à des composants comme le si4021 au format TSSOP16.

Vient alors l'étape de la soudure du composant sur le circuit imprimé. Il est important de signaler ici que, même s'il peut être tentant de souder en premier les connecteurs au pas de 2,54 mm pour ensuite placer l'ensemble sur une platine à essai pour plus de stabilité, c'est généralement une mauvaise idée. Mieux vaut s'assurer une certaine liberté de mouvement et commencer par la partie la plus difficile et la plus petite.

L'usage d'une pince brucelles est un avantage certain, et de préférence un modèle de qualité, à pointes fines, assurant un maximum de précision. Vous pouvez cependant faire sans, à condition d'utiliser quelque chose de pointu pour déplacer et maintenir le composant en place. De l'étain d'un diamètre le plus fin possible est également un plus, vous permettant de doser avec précision la quantité utilisée. Enfin, du flux de soudure (liquide de préférence) est quelque chose qui vous rendra la tâche véritablement aisée en permettant à l'étain de se glisser rapidement aux bons endroits et, par la même occasion, de nettoyer les connecteurs et les pattes (même si les circuits imprimés de ce genre arrivent généralement étamés).

La technique est relativement simple pour un circuit intégré. On commence par faire fondre un peu d'étain sur un connecteur à l'un des coins de l'emplacement, puis on place délicatement le composant en place. Cette première soudure permet de fixer le composant et d'éventuellement ajuster sa position en chauffant à nouveau

Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)

Abonnez-vous !

C'est simple... c'est possible sur

http://www.ed-diamond.com

... OU 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 :	

❑ 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 : http://**boutique.ed-diamond.com/content/3-conditions-generales-de-ventes** et reconnais que ces conditions de vente me sont opposables.

HACKABLE MAGAZINE

Les Éditions Diamond
Service des Abonnements
10, Place de la Cathédrale
68000 Colmar – France
Tél. : + 33 (0) 3 67 10 00 20
Fax : + 33 (0) 3 67 10 00 21

Vos remarques :

.....

.....

Ce document est la propriété exclusive de Alex Arnaud(baarnuxdroid@gmail.com)

Bon d'abonnement

CHOISISSEZ VOTRE OFFRE !

Prix TTC en Euros / France Métropolitaine*

Offre	SUPPORT / ABONNEMENT	PAPIER Réf	PAPIER Tarif TTC	PAPIER + BASE DOCUMENTAIRE (1 connexion BD) Réf	PAPIER + BASE DOCUMENTAIRE (1 connexion BD) Tarif TTC
HK	6n° Hackable	HK1	39,-	HK13	169,-
LES COUPLAGES AVEC NOS AUTRES MAGAZINES					
i	6n° Hackable + 6n° MISC	i1	79,-	i13	419,-
i+	6n° Hackable + 6n° MISC + 2n° Hors-Série	i+1	99,-	i+13	439,-
J	11n° GNU/Linux Magazine France + 6n° Hackable	J1	105,-	J13	399,-
J+	6n° Hackable + 11n° GNU/Linux Magazine France + 6n° Hors-Série	J+1	159,-	J+13	459,-
K	6n° Hackable + 6n° Linux Pratique	K1	75,-	K13	329,-
K+	6n° Hackable + 6n° Linux Pratique + 3n° Hors-Série	K+1	99,-	K+13	359,-
LA TOTALE DIAMOND !					
L	11n° GLMF + 6n° HK* + 6n° LP + 6n° MISC	L1	189,-	L13	839,-
L+	11n° GLMF + 6n° HS + 6n° HK* + 6n° LP + 3n° HS + 6n° MISC + 2n° HS	L+1	289,-	L+13	939,-

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

Particuliers = CONNECTEZ-VOUS SUR :

http://www.ed-diamond.com

pour consulter toutes les offres !

*Les tarifs hors France Métropolitaine, Europe, Asie, etc. sont disponibles en ligne !

Professionnels = CONNECTEZ-VOUS SUR :

http://proboutique.ed-diamond.com

pour consulter toutes les offres !

*Les tarifs hors France Métropolitaine, Europe, Asie, etc. sont disponibles en ligne !

Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)

avec la panne du fer. Avant de souder cette patte, assurez-vous que le composant est correctement orienté. Une erreur à ce niveau n'est pas une catastrophe, mais il n'y a rien de plus énervant qu'une numérotation de broches (sérigraphiée sur le circuit) qui ne corresponde pas à celle du composant (surtout quand on le remarque par la suite).

Une fois le placement totalement satisfaisant, on soudera la patte diamétralement opposée de façon à bien fixer l'ensemble. Ce n'est qu'alors qu'on soudera patte après patte. En cas d'excédent d'étain et formation de ponds, pas de panique, il vous suffira d'utiliser de la tresse à dessouder ou simplement la panne du fer bien propre pour corriger l'erreur. L'étain fondu « veut » aller au bon endroit naturellement et la tension de surface de l'alliage en fusion fait tout le travail (c'est là que le flux est véritablement pratique).

Une autre technique appelée *drag soldering* débute de la même manière en soudant une patte du composant en place. On balaye ensuite les pattes du côté opposé en une fois avec la panne du fer tout en ajoutant de l'étain. On peut également faire de même en faisant fondre l'étain sur la panne pour ensuite la glisser sur les pattes. Ici l'utilisation de flux est presque indispensable et mieux vaut éviter l'étain en intégrant (*flux core*), car celui-ci risque de brûler sur la panne du fer avant l'opération et la rendra plus difficile.

Personnellement, la première technique me convient parfaitement et donne généralement de

Le produit utilisé pour tester le montage cloné sur la base des restes d'une vieille télécommande à fréquence est cette prise radiocommandée de marque AquaForte modèle RC-400.

très bons résultats. Une inspection minutieuse, à la loupe, vous permettra de confirmer que le composant est bien soudé. Vous pouvez ensuite ajouter les broches sur les côtés du circuit. Là, la solution la plus simple est de placer les broches (barrettes sécables) sur une platine à essais et placer ensuite le circuit dessus. On pourra alors souder rapidement et facilement chaque broche sans avoir à jongler inutilement.

D'autres techniques existent, impliquant de la pâte de soudure (mélange de micro-billes d'étain et de flux) et éventuellement un matériel adapté (station à air chaud, four à refusion, etc.). Eh oui, il est effectivement possible aussi d'utiliser une plancha pour ce genre de choses (n'est-ce pas Nathaël).

2.2 Le quartz

À présent que le si4021 se trouve dans un format utilisable plus facilement, il est temps de passer au montage lui-même. Sur la platine à essai, on disposera simplement le composant au centre en établissant les connexions (ou futures connexions) suivantes avec la carte Arduino UNO :

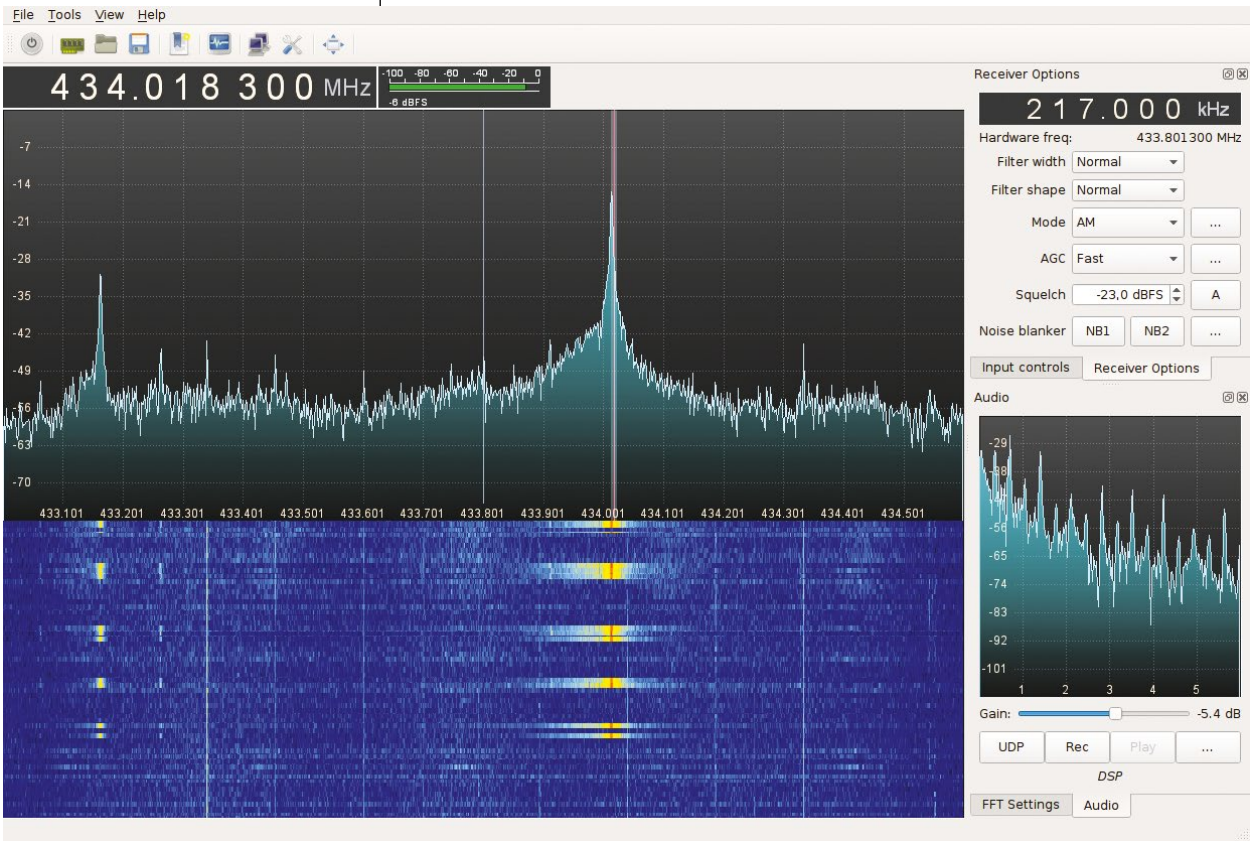
Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)

On voit ici clairement que même en ayant bien configuré le si4021 pour une émission sur 433,92 Mhz, le signal détecté par le récepteur RTL-SDR est décalé de quelques 100 Khz. Ceci provient d'un mauvais réglage des condensateurs de charge pour le quartz attaché au si4021. En augmentant cette valeur à son maximum (16 pF) dans la configuration, tout rentre dans l'ordre.

- 1 MOSI sur 11,
- 2 SCK sur 13,
- 3 nSEL sur 10,
- 9 XTL : quartz,
- 10 VSS : masse,
- 11 MOD : +5V (si4021 en mode microcontrôleur),
- 12 RFN : antenne -,
- 13 RFP : antenne +,
- 15 VDD : +5V,
- 16 FSK : 8.

Comme le montre le schéma simplifié tiré de la documentation du si4021, la broche 9 est connectée à un oscillateur à quartz dont l'autre broche est reliée à la masse. Ce type de composant se trouve ou se récupère facilement un peu partout. La fréquence du quartz est généralement imprimée, sérigraphiée ou gravée sur l'enveloppe métallique, mais ceci n'est pas suffisant. Dans notre cas, nous avons besoin d'une fréquence de 10 Mhz, mais un autre paramètre doit être connu pour garantir la précision de la fréquence d'émission : la valeur du condensateur de charge du quartz.

Les quartz sont taillés de façon à fournir une oscillation à la fréquence pour laquelle ils sont conçus en utilisant en parallèle un condensateur d'une valeur précise. En d'autres termes, la fréquence ne sera juste qu'à



Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)

Si vous souhaitez disposer d'une documentation complète pour un oscillateur à quartz, inutile de vous tourner vers un achat sur eBay ou une récupération dans un autre matériel. Il est presque impossible de trouver ne serait-ce que le fabricant à partir du composant lui-même. L'achat chez un détaillant de composants électroniques après sélection du modèle aux caractéristiques précises qui vous conviennent est la seule approche raisonnable.

condition d'utiliser la valeur de condensateur adéquate. Si la valeur est plus importante que celle préconisée par le fabricant, la fréquence sera plus basse, et inversement. Le si4021 dispose d'une fonction d'ajustement automatique de l'antenne, mais ceci n'a aucune importance si le quartz ne résonne pas à la bonne fréquence à la base.

Dans bien des cas, lorsque vous récupérez un quartz sur un circuit ou en achetez sur un site d'enchères en ligne (ou autre), vous n'avez pas de mention du modèle exact (tantôt pas même la marque) et n'avez donc aucune idée des spécifications techniques.

Ici, nous avons la chance de pouvoir régler cette valeur de façon logicielle puisque ces condensateurs de charge sont intégrés au si4021. Il ne s'agit donc que de choisir une configuration pour utiliser des valeurs entre 8,5 pF et 16 pF (par incrément de 0,5 pF). Ceci permet de procéder par tâtonnement en réglant une valeur de configuration et en utilisant le montage, tout en inspectant avec un logiciel SDR si l'émission se fait sur la bonne fréquence.

Le croquis utilisé dans le précédent article a sensiblement évolué depuis sa parution en définissant un certain nombre de fonctions et de macros permettant de s'y retrouver facilement dans les bits servant à la configuration (cette nouvelle version est disponible dans le dépôt GitHub du numéro). Ainsi, la commande envoyée pour la configuration du si4021 est, à présent, « calculée » avec cette fonction :

```
uint16_t confset(float freq) {
  if(freq < 440) {
    return(CONFSET | CS_CLK10000 | CS_BAND433 | CS_CAPA160);
  } else if (freq < 880) {
    return(CONFSET | CS_CLK10000 | CS_BAND868 | CS_CAPA160);
  } else if (freq < 930) {
    return(CONFSET | CS_CLK10000 | CS_BAND915 | CS_CAPA160);
  } else {
    return(0);
  }
}
```

On lui passe simplement la fréquence à utiliser et celle-ci sert de base pour le choix des bits de configuration. La valeur sur 16 bits retournée comprend les trois bits correspondants à la commande, la configuration de la fréquence générée sur la broche CLK, les bits permettant le choix de la bande et ceux définissant la capacité pour le quartz. Il suffit donc de choisir la bonne macro définie dans `si4021.h` et de faire un test. On observe ensuite le signal dans le logiciel

Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)

utilisant une clé RTL-SDR calée sur 433 Mhz et on ajuste en conséquence : trop haut on augmente la capacité, trop bas on la réduit. Comme le précise la documentation du si4021 (lorsqu'on la lit entièrement), du fait du réglage automatique de l'antenne, un décalage de fréquence ne peut venir que de la fréquence de référence : celle du quartz utilisé.

Bien entendu, la solution la plus simple est de tout bonnement commander un (ou des) quartz de 10 Mhz avec une capacité de charge précise, connue et choisie, auprès d'un détaillant de composants (Farnell, Mouser, Digikey, etc.). Le petit problème dans ce cas, surtout pour moi, est le fait que le port n'est offert qu'à partir d'un certain montant de commande, ce qui incite à ajouter des choses dans son panier et à finir avec une facture bien loin de la poignée d'euros initialement prévue pour quelques quartz...

2.3 L'antenne

Nous avons réglé le problème de format du composant et celui de l'oscillateur permettant au composant de fonctionner en étant correctement configuré. Mais il reste maintenant ce qui est, à mon sens, la partie la plus délicate : l'antenne.
La conception, le réglage, l'utilisation et la mise au point d'antennes sont un domaine à part entière, reposant sur des concepts et des connaissances relativement complexes.

Le si4021 intègre, heureusement pour nous, un circuit de réglage automatique de l'antenne (*automatic antenna tuning circuit* dans la documentation) et le contrôle direct d'une antenne cadre (*loop antenna* en anglais) via un amplificateur. Aucun composant ou circuit supplémentaire n'est nécessaire pour connecter l'antenne et dans ce cas précis, celle-ci est généralement intégrée au circuit imprimé sous la forme de pistes sur le support (il est possible d'utiliser un autre type d'antenne avec le si4021, mais il faut alors créer un balun nécessitant différents composants décrits dans la documentation, page 28).

L'antenne-cadre est directement connectée à l'amplificateur qui présente deux sorties à collecteur ouvert notées RFN et RFP. Ce schéma de branchement est désigné, du moins chez Silicon Labs, d'antenne différentielle à haute impédance. Comme le recommande un message dans la base de connaissances Silicon Labs, il est important de connecter l'antenne directement au plus près des pattes du si4021, ce qui est le cas du circuit de télécommande de récupération, par exemple.

Trouver des informations accessibles sur ce type d'antenne-cadre n'est pas chose facile et on en arrive invariablement à la récupération (tant bien que mal) du

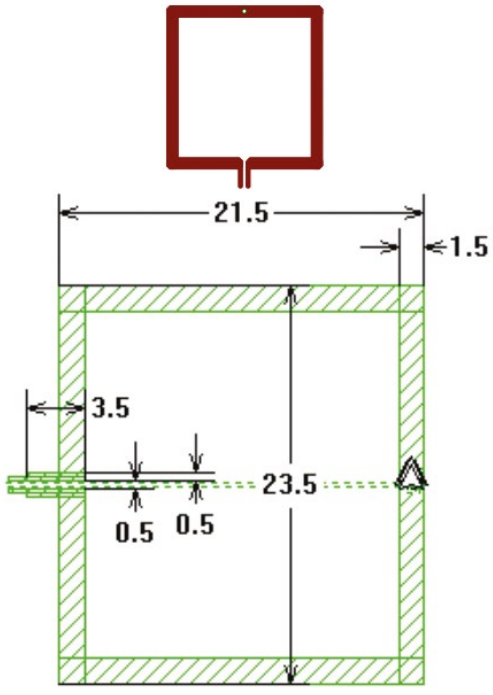


Fig 2.7 434 MHz TX loop antenna (dimensions in mm)

Circuit avec mesures pour la création d'une antenne-cadre pour la bande 434 Mhz telle que présentée dans le guide de sélection d'antennes EZRadio (la gamme de circuits intégrés à laquelle appartient le si4021). Ceci peut nous servir de base pour la confection d'une antenne « maison », qui sera loin d'être optimale, mais qui, grâce au réglage automatique intégré au si4021, devrait finalement faire un travail acceptable.

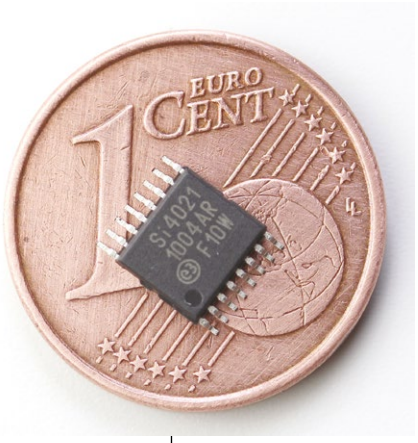
Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)

guide de conception d'antennes EZRadio (référence IA ISM-AN2), un document de 46 pages, certainement parfaitement intelligible par quelqu'un ayant de l'expérience dans ce type d'exercice (ce qui n'est pas mon cas). On pourra aussi consulter le guide de sélection d'antennes (IA ISM-AN1) qui présente un circuit imprimé (page 14) pour l'antenne d'émission en 434 Mhz ressemblant fortement à celle du circuit de récupération et celle dans la documentation du si4021. Mais là, nous avons des mesures !

La solution que j'ai donc choisi d'utiliser est assez empirique puisqu'il s'agit de simplement utiliser ces informations et mesures pour construire une antenne plus ou moins similaire. Ceci pourra être fait assez facilement en utilisant un morceau de câble en cuivre rigide gainé d'environ 9 cm de long (+ la partie s'enfonçant dans la platine à essai). Celui-ci se verra soudé en son centre pour ajouter une connexion à la tension d'alimentation et ainsi former quelque chose d'assez similaire aux illustrations présentes dans les documentations et au circuit imprimé initial.

Bien entendu, sur platine à essais, les connexions et l'ensemble du montage impliquent des capacités qui perturbent le fonctionnement de l'antenne, mais qui peuvent, dans une certaine mesure être prises en charge par le circuit de réglage automatique du si4021. Au final, cela fonctionne, avec une portée tout à fait identique à la fois à la télécommande d'origine, mais aussi à celle du circuit de récupération du précédent article. Il n'y a cependant pas de quoi fanfaronner, car l'aspect « bricolo » du montage est plus qu'évident et ferait grincer des dents, je pense, n'importe quel radio-amateur digne de ce nom.

D'autres facteurs que la réalisation de l'antenne et sa symétrie sont importants. On peut par exemple, en cas de problème filtrer davantage l'alimentation en plaçant plusieurs condensateurs entre +5V et la masse au plus proche de l'antenne. On peut également ajuster la puissance de l'amplificateur (atténuation) dans le croquis ou encore tout simplement souder l'antenne au circuit d'adaptation TSSOP/DIP et non l'enficher sur la platine à essais. À chaque modification, vérifiez le résultat avec votre application SDR pour vous assurer de la qualité du signal et l'absence d'harmoniques ou de signaux sur d'autres fréquences qui apparaissent en même temps que sur la fréquence fondamentale (celle que vous avez réglée).



Le si4021 en boîtier TSSOP16 est absolument minuscule, mais il reste possible, avec un simple fer à souder et beaucoup de patience, de le mettre en œuvre assez facilement. Pour d'autres formats, comme le BGA par exemple, un équipement spécifique sera en revanche indispensable.

CONCLUSION

Nous avons ici montré que, même si un certain nombre de problèmes pratiques peuvent sembler nous barrer la route, il existe des moyens d'arriver à ses fins lorsqu'on souhaite reproduire un circuit qui n'existe pas sous forme de module. Ainsi, sans nécessairement devoir investir dans de l'équipement coûteux ou des techniques très avancées, on peut, dans une certaine mesure, créer ce qui n'existe pas sous une forme directement utilisable. Mieux encore, il devient possible de s'inspirer de circuits existants pour créer des déclinaisons simplifiées en achetant, à bas coût, le minimum de composants utiles. Tout ce qu'il vous faut c'est de l'imagination, un peu de patience et apprendre à parfois faire quelques concessions.

Le croquis associé avec ce projet, même s'il est ici secondaire, est disponible sur le dépôt GitHub du magazine à l'adresse **https://github.com/Hackable-magazine/Hackable19**. Il contient les commentaires nécessaires pour en comprendre le fonctionnement. **DB**

Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)

OBTENEZ N'IMPORTE QUELLE TENSION À PARTIR DES 5V USB

Denis Bodor

On trouve de tout sur eBay et généralement à des prix très intéressants. Parmi cette myriade de produits électroniques, certains sont vendus pour un usage spécifique, mais dès lors qu'on en étudie le fonctionnement, on découvre rapidement qu'ils peuvent faire bien plus que ce qui est annoncé. C'est le cas de ces convertisseurs de tension USB vers 12V ou 9V qui cachent de captivants secrets…

Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)

Payer peu pour un produit est intéressant à bien des niveaux, mais surtout, cela permet des expérimentations parfois hasardeuses sans vraiment avoir à s'inquiéter de la destruction probable du matériel. Notre victime pour cet article sera un convertisseur de tension (*step-up* ou *boost converter* en anglais) USB vers 12V vendu sur eBay pour un peu plus d'un euro. L'objet de ce produit est de proposer d'un côté une entrée 5V USB type A mâle et de l'autre un connecteur d'alimentation 2,1 x 5,5 mm avec une tension de 12V.

Ceci permet d'alimenter, par exemple, un équipement en 12V à partir d'un bloc d'alimentation USB 5V ou encore une de ces fantastiques batteries d'appoint pour smartphone. Ce produit, désigné par « *USB DC 5V to DC 9V/12V Step-up Module Converter 2.1×5.5mm Male Connector Plus* » dans l'annonce par un vendeur appelé *shieldsfan* se trouve un peu partout et se décline généralement en deux versions : sortie 12V ou 9V. La description de l'objet précise un courant maximum en entrée de 2,1A et 800 mA en sortie, avec une note recommandant de ne pas dépasser 500 mA (ce qui fait tout de même 6 watts).

Le simple fait que le produit se décline en une version 12V et 9V nous donne déjà un indice intéressant quant à sa construction. Deux cas sont envisageables, soit il s'agit de deux puces ou circuits différents, soit c'est une même puce, mais avec des réglages variables. Dans cette seconde éventualité, ceci signifie qu'en étudiant le produit il doit être est possible d'ajuster la tension en sortie selon nos besoins...

1. DÉMONTAGE ET DOCUMENTATION

Comme souvent à la réception d'un produit, et en particulier un produit vraiment peu coûteux commandé en quantité (6 exemplaires ici), mon premier mouvement consiste généralement à partir du principe qu'un sacrifice rituel sur l'autel de la science est justifié. Comprenez par là qu'un exemplaire est donc disséqué, ne serait-ce que pour en juger de la qualité, tout en gardant l'espoir (parfois vain) qu'il puisse être éventuellement réassemblé et tout de même utilisé par la suite (j'avoue, c'est rare).

Fort heureusement ici, le circuit de ce convertisseur est placé dans une coque en plastique qui n'est ni collée ni soudée. L'ouverture du boîtier est donc un jeu d'enfant puisque les deux parties qui le compose sont simplement emboîtées et se désolidarisent sans même utiliser d'outil.

Le démontage révèle peu de composants : une bobine, quelques condensateurs de filtrage, une diode, des résistances et un circuit intégré. Ce dernier élément est très intéressant, car il s'agit d'un MT3806 d'*Aerosemi Technology*. Une rapide recherche sur le Web permet d'apprendre, via la

Ceci est un convertisseur de tension destiné à fournir 12V à partir d'un connecteur USB délivrant 5V. Attention toutefois, un port USB d'un ordinateur ou d'un hub n'est pas capable de fournir beaucoup de courant (500 mA max). Ce type de matériel est donc généralement destiné à une utilisation avec les blocs d'alimentation et les batteries d'appoint pour smartphones.

Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)

Une chance pour nous, l'ouverture du boîtier et la mise à nu du circuit est aisée, car simplement clipsé. On pourra donc non seulement modifier le circuit comme bon nous semble, mais également refermer proprement le tout sans le moindre problème.

documentation technique du composant (*datasheet*), qu'il s'agit d'un convertisseur de tension (*step-up converter*) avec des caractéristiques très intéressantes :

- tension d'entrée entre 2V et 24V ;
- tension en sortie ajustable jusqu'à 28V ;
- haute efficacité (97%) et donc idéal pour les applications sur batterie par exemple ;
- protection thermique en cas de surcharge de courant en sortie ;
- mécanisme de *soft start* pour éviter un afflux de courant en entrée au démarrage ;
- nombre de composants annexes réduit.

Voilà qui est captivant, car en observant le circuit utilisé dans le convertisseur, on remarque que ce n'est ni plus ni moins que celui décrit dans la documentation. L'entrée USB est directement connectée au circuit intégré et à la masse ce qui signifie une chose très sympathique : ce convertisseur, sans la moindre modification peut donc fonctionner avec bien autre chose que les 5V d'un port USB. Et en effet, après un petit test avec une alimentation de laboratoire, l'adaptateur est effectivement capable de fournir 12V à partir d'une tension de 2V, et même en mesure de faire parfaitement fonctionner un équipement comme un routeur Linksys LRT214 de cette façon (je ne sais pas quand, mais je suis sûr qu'un jour cela pourra être utile). Nous avons donc déjà gagné quelque chose du simple fait d'avoir lâché la bride à notre curiosité sans même avoir sorti le fer à souder.

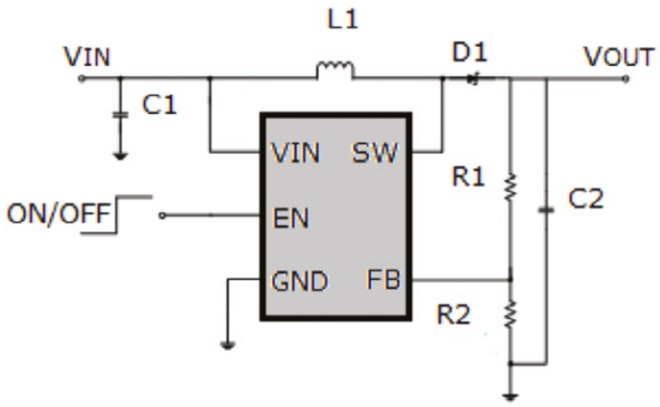
Bien entendu, il existe une relation évidente entre la tension en entrée, le courant, la tension en sortie et le courant fourni. Ce type d'adaptateur est un *step-up* ou *boost converter* ou, en d'autres termes, une alimentation à découpage reposant sur l'utilisation d'une bobine, qui stocke l'énergie sous forme de champ magnétique. C'est cette énergie qui s'additionne avec celle fournie par l'alimentation source, le tout cadencé par un MOSFET, qui permet d'augmenter la tension tout en reposant sur un condensateur pour produire un courant continu.

Le circuit intégré MT3806 embarque le MOSFET et toute la logique permettant d'obtenir ainsi une tension précise via l'utilisation d'une broche dédiée permettant d'avoir une indication de la tension en sortie. Le composant se charge alors tout seul d'arriver à la tension choisie en jouant sur le rapport cyclique appliqué au MOSFET et donc à l'alimentation de la bobine.

Le circuit de l'adaptateur est relativement simple et surtout monocouche. Ceci signifie qu'il est très facile d'en suivre la logique à l'aide de la documentation du circuit intégré.

Il devient donc très facile de repérer les composants et de faire

Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)



La documentation du circuit intégré MT3608 présente une « utilisation typique » correspondant parfaitement au circuit utilisé par l'adaptateur, au composant près.

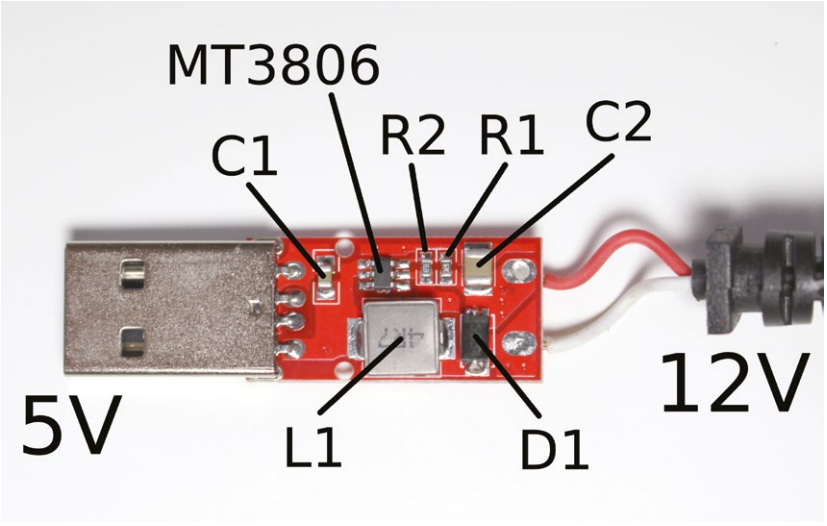
le rapprochement entre le schéma et la réalisation. La correspondance entre l'application décrite dans la documentation et le produit n'a rien de surprenant, c'est généralement le cas dans 90% des modules et/ou des produits d'entrée de gamme de ce type. Tantôt des raccourcis sont faits ou des composants sont omis pour réduire les coûts de fabrication, mais dans les grandes lignes, ceci est tout à fait caractéristique de ce type de matériel. Ceci permet donc tantôt d'en améliorer la qualité, en changeant par exemple les condensateurs par des composants de meilleure qualité, mais aussi, comme ici, d'adapter le matériel à nos besoins.

Sur le circuit présent dans le produit, on remarque donc immédiatement qu'en influant sur les résistances R1 et R2 il est possible de changer la tension en sortie. Tout ce que nous avons à faire est finalement de changer un de ces composants, ou les deux, pour adapter le matériel à nos besoins.

2. ADAPTEZ LA TENSION À VOS BESOINS

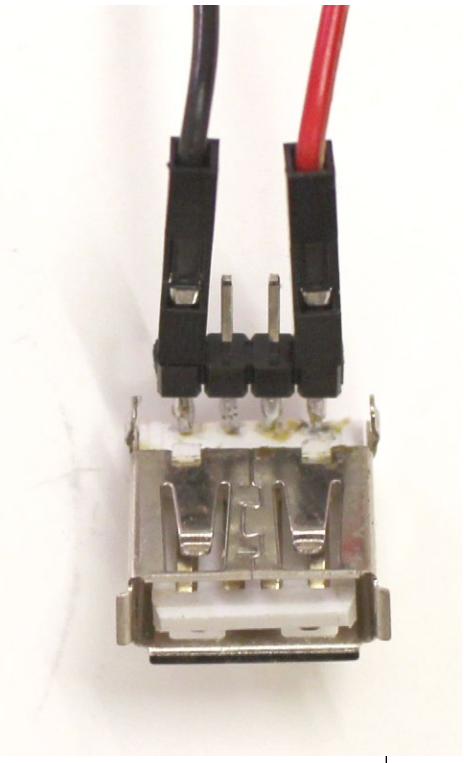
Le schéma de la documentation nous montre un circuit relativement simple. R1 est connecté entre la tension de sortie et la broche FB (*feedback*) du MT3608. Cette broche est également reliée à la masse commune via la résistance R2. Ceci forme donc un diviseur de tension tout à fait classique où la différence de potentiel entre FB et la masse sera Vout x (R2/(R1+R2)).

Le mode de fonctionnement du MT3608 consiste à produire sur Vout une tension de manière à ce que FB présente 0,6 volts. En d'autres termes, en fonction des valeurs utilisées pour R1



Le circuit imprimé dans le convertisseur de tension peut être analysé rapidement en suivant simplement les pistes. Ceci permet d'identifier précisément les composants également mentionnés dans la documentation du MT3608. Sur cette illustration, le travail est déjà fait, mais prendre un cliché pour y noter les désignations des composants est la marche à suivre la plus efficace.

Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)



Pour procéder aux tests sans dessouder le connecteur USB du convertisseur, rien de plus simple : il suffit de bricoler un connecteur USB femelle en lui soudant des broches au pas de 2,54 mm. De simples câbles standards « dupont » peuvent ensuite être utilisés pour la connexion à l'alimentation de laboratoire ou à n'importe quelle autre source.

et R2, le circuit intégré ajustera Vout jusqu'à obtenir 0,6V sur FB. C'est donc le calcul du diviseur de tension qui détermine la mise à 0,6V de FB pour une tension Vout donnée.

Nous pouvons vérifier simplement cela en faisant les calculs sur la base des résistances déjà en place. R1 est marqué 63C et R2 36B, il s'agit d'un codage normalisé E96. En cherchant un peu sur le Web, on apprend ainsi que 63C correspond à 442 x 100, soit 44200 ohms et 36B à 232 x 10, soit 2320 ohms. On peut alors faire le calcul 12V x (2320/(44200+2320) = 0,5984V, soit environ 0,6V. Mais il est plus intelligent de faire le calcul inverse puisque le MT3608 fera en sorte de voir 0,6V sur FB :

```
FB = 0,6 = Vout * (R2/(R1+R2))

1 = Vout * (R2 / (R1+R2)) / 0,6

1 / Vout = (R2 / (R1+R2)) / 0,6

Vout = 1 / ((2320 / (44200+2320)) / 0,6) = 12,0310V
```

On retombe donc effectivement sur les 12 volts présents à la sortie du convertisseur. Mais en triturant un peu la formule, on peut également arriver à quelque chose de plus simple qui nous permettra de choisir plus facilement les valeurs de résistance :

```
FB = 0,6 = Vout * (R2 / (R1+R2))

0,6 = Vout * (R2 / (R1+R2))

Vout / 0,6 = (R2 / (R1+R2))

Vout / 0,6 = (R1 / R2) + 1

(Vout / 0,6) - 1 = R1 / R2

(12 / 0,6) - 1 = R1 / R2 =~ 19
```

Nous pouvons donc nous amuser à changer R1 et/ou R2 pour modifier la tension en sortie. Imaginons alors que vous ayez besoin d'une tension de 15V pour un montage ou un appareil quelconque, et que nous ne souhaitons changer qu'un composant :

```
(15 / 0,6) - 1 = R1 / R2

25 - 1 = R1 / R2

24 = R1 / R2

24 = 44200 / R2

24 x R2 = 44200

R2 = 44200 / 24 = 1841,66
```

Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)

En remplaçant R1 par une résistance de 55,650 Kohms, nous obtiendrons la tension souhaitée. Mais comme une telle résistance n'existe pas dans une série normalisée, nous opterions pour 56 Kohm avec pour résultat : Vout = 1/(1800/(1800+44200)/0,6) = 15,33, ce qui généralement fera l'affaire.

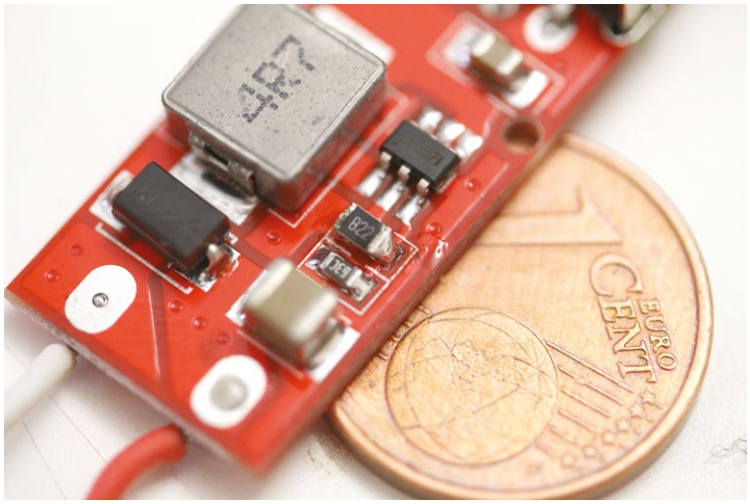
Dessouder et ressouder des composants CMS comme ceux-ci est cependant très pénible. Ce qu'on pourra faire, en revanche, c'est placer une résistance en parallèle à R2, en soudant tout simplement une nouvelle résistance de plus grande valeur par-dessus celle déjà présente.

Rappelons que les valeurs des résistances connectées en série s'additionnent, mais que celles en parallèle répondent au calcul : 1/Req = 1/R1 + 1/R2 + 1/R3 +... où *Req* est la résistance équivalente, ce qui se transforme dans le cas de deux résistances en Req = (R1 x R2) / (R1 + R2). Comme nous avons déjà en place une résistance R2 de 2320 ohms, il nous suffit de mettre d'en parallèle une seconde de 8,2 Kohms pour obtenir : (2320 x 8200) / (2320 + 8200) = 1808,36 ohms. C'est une valeur suffisamment proche des 1841,66 pour obtenir alors Vout = 1/(1800,36/(1800,36+44200)/0,6) = 15,26 volts. Une résistance de 10 Kohms fonctionnerait peut-être aussi selon l'utilisation des 15V, mais je vous laisse faire le calcul afin de savoir pourquoi...

2.1 Pour finir

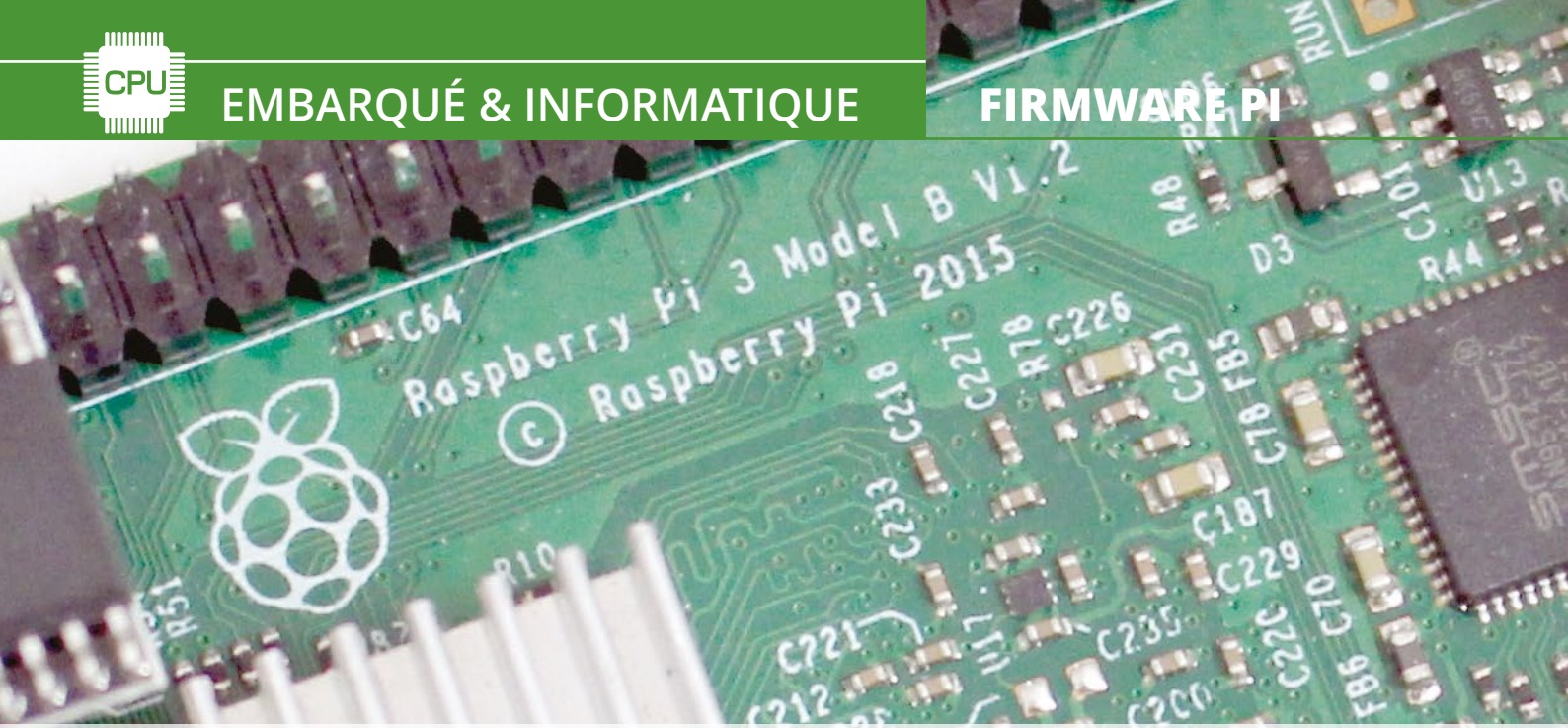
Ici, nous avons joué avec R2, mais il est également possible de s'occuper de R1 ou des deux résistances de concert de la même manière. On pourra même éventuellement retirer les deux résistances et mettre en place un montage un peu plus évolué en reliant la masse, Vout et FB à une platine pastillée équipée d'une résistance standard (non CMS) à la place de R2 et d'un potentiomètre pour remplacer R1. On obtiendrait alors une alimentation avec une tension réglable pour des usages spécifiques. Nous serons très loin d'une alimentation ajustable digne de ce nom, mais pour quelques tests ponctuels cela sera bien suffisant.

Enfin, comme le MT3608 accepte une tension en entrée d'un minimum de 2 volts, cette même technique pourra être utilisée pour obtenir 5V à partir de 3,3V ou encore à partir d'une paire de piles AA ou AAA. Là, comme la tension de coupure du MT3608 est de 1,98V, le risque de sur-déchargement des piles est limité et il sera toujours possible d'ajouter une liaison entre le montage et l'anode des piles pour surveiller la tension et en déduire la capacité restante. Ceci ouvre clairement la porte à une solution très économique permettant d'utiliser, par exemple, une carte Arduino sur piles à très faible coût et pour peu d'effort. **DB**



L'ajout d'une résistance CMS supplémentaire par dessus celle existante demande un peu de finesse et de dextérité, mais cela reste bien plus facile que de dessouder le composant déjà en place pour le remplacer. En plaçant sa langue selon le bon angle et en retenant son souffle, on arrive à faire quelque chose qui ne fasse pas trop « pâté » et on finit par avoir les 15V attendus.

Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)

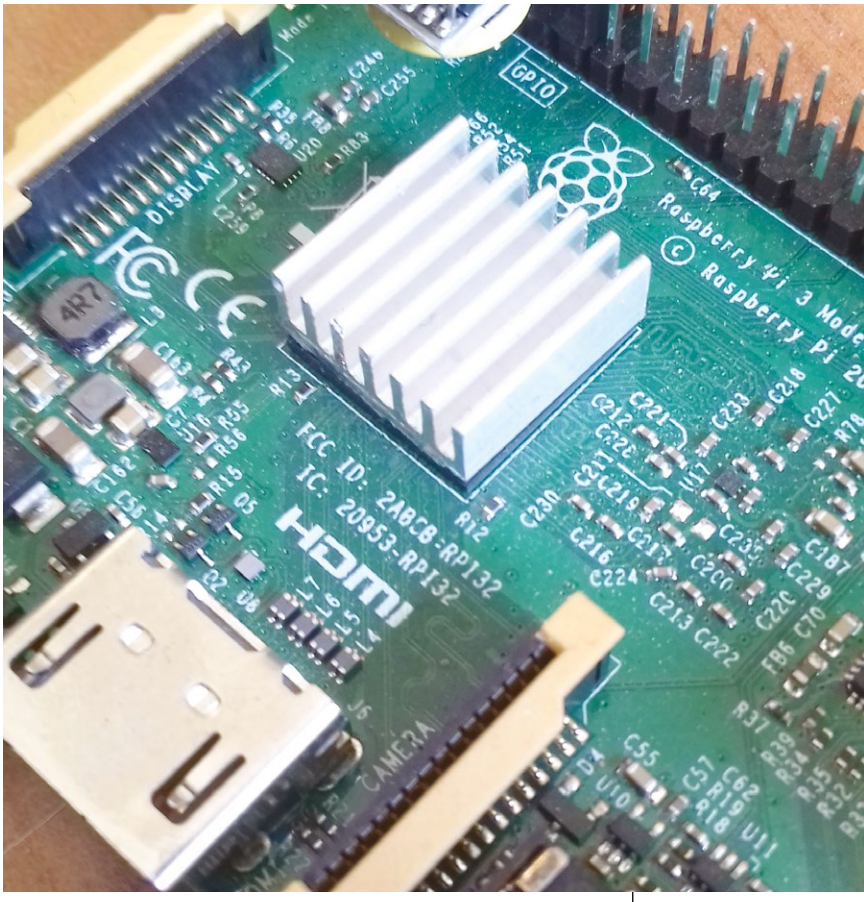


OBTENIR LES INFORMATIONS DU FIRMWARE DE VOTRE RASPBERRY PI

Laurent DELMAS

Sur chaque ordinateur, il existe un logiciel embarqué qui permet de démarrer et de vérifier le matériel, et ce, avant même que votre système d'exploitation ne soit démarré. Il s'agit du BIOS, UEFI ou bien d'un firmware spécifique comme dans le cas du Raspberry Pi. Nous allons voir comment obtenir les informations matérielles de notre machine au travers de ce logiciel embarqué.

Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)



1. DÉMARRAGE DU RASPBERRY PI

Avant même de pouvoir utiliser un système d'exploitation GNU/Linux, Mac OS, Windows… tout système doit démarrer, initialiser et vérifier les composants constituant la machine que ce soit une tablette, un serveur, un ordinateur de bureau, un smartphone ou bien un nano-ordinateur. C'est le but du BIOS, système élémentaire d'entrées sorties (*Basic Input Output Système*), recenser et vérifier les éléments de la machine. En cas de problème, il émet une succession de bips permettant ainsi de diagnostiquer le problème. Le Raspberry Pi, quant à lui, ne possède pas de BIOS, mais un firmware spécifique qui se trouve sur la première partition de la carte mémoire. Comme vous avez pu le constater par ailleurs, une carte mémoire contenant une distribution Raspbian présente deux partitions, l'une au format VFAT dont le point de montage est **/boot** qui contient le noyau statique, les fichiers de configuration et le firmware lui-même puis une partition au format ext4 avec la racine du système **/**.

```
pi@raspberrypi:~$ lsblk
NAME        MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
mmcblk0     179:0    0 14.9G  0 disk
├─mmcblk0p1 179:1    0   63M  0 part /boot
└─mmcblk0p2 179:2    0 14.8G  0 part /
```

Le firmware est constitué de trois fichiers : **bootcode.bin** , **start.elf** et le fichier de configuration associé **config.txt**. L'ensemble est exécuté à la mise sous tension par le processeur graphique (GPU) intégré dans le cœur du SoC (*System on Chip*) Broadcom du Raspberry Pi.

Contrairement au BIOS qui émet une succession de bips en cas de problème lors de l'initialisation du matériel, le firmware du Raspberry Pi affiche des icônes différentes en fonction du problème détecté. Trois types de problèmes peuvent survenir :

- une chute de tension, c'est-à-dire un problème d'alimentation, représenté par un éclair à l'écran ;
- une surchauffe du processeur signalée par un thermomètre rouge à mi-niveau ;
- une surchauffe du processeur graphique (GPU) pour lequel s'affiche un thermomètre rouge plein.

Lorsque le firmware a terminé d'initialiser et vérifier les composants du Raspberry Pi, il démarre ensuite le noyau Linux également situé dans la partition **/boot**.

Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)

2. INSTALLATION

Le firmware du Raspberry Pi n'étant pas un BIOS, les outils classiques tels que **dmidecode** ne fonctionnent pas, il faut donc utiliser un outil spécifique nommé **vcgencmd** pour accéder aux informations du firmware. **vcgencmd** est installé avec le paquet **rpi-update**. Cet outil a la particularité de lire les caractéristiques du SoC (*System On Chip* ou système embarqué) via diverses commandes qui lui sont passées en paramètres. L'installation s'effectue de façon toute à fait classique :

```
pi@raspberry:~$ sudo apt-get install rpi-update
```

3. PRISE EN MAIN

Nous allons apprendre maintenant comment utiliser l'outil **vcgencmd**. Tout d'abord, utilisons le paramètre **commands** afin d'obtenir la liste complète des commandes accessibles.

```
pi@raspberry:~$ vcgencmd commands
commands="vcos, ap_output_control, ap_output_post_processing, vchi_
test_init, vchi_test_exit, pm_set_policy, pm_get_status, pm_show_stats,
pm_start_logging, pm_stop_logging, version, commands, set_vll_dir, set_
backlight, set_logging, get_lcd_info, arbiter, cache_flush, otp_dump,
test_result, codec_enabled, get_camera, get_mem, measure_clock, measure_
volts, scaling_kernel, scaling_sharpness, get_hvs_asserts, get_throttled,
measure_temp, get_config, hdmi_ntsc_freqs, hdmi_adjust_clock, hdmi_status_
show, hvs_update_fields, pwm_speedup, force_audio, hdmi_stream_channels,
hdmi_channel_map, display_power, read_ring_osc, memtest, dispmanx_list,
get_rsts, schmoo, render_bar, disk_notify, inuse_notify, sus_suspend, sus_
status, sus_is_enabled, sus_stop_test_thread, egl_platform_switch, mem_
validate, mem_oom, mem_reloc_stats, hdmi_cvt, hdmi_timings, file, vctest_
memmap, vctest_start, vctest_stop, vctest_set, vctest_get"
```

Il apparaît une liste de commandes conséquentes. Nous allons découvrir les principales. Tout d'abord, prenons par exemple la commande **measure_clock** qui permet de mesurer la fréquence d'horloge d'un des paramètres du microprocesseur. La commande est suivie du nom de l'horloge souhaitée parmi la liste suivante : **arm core h264 pixel emmc uart v3d isp uart hdmi** :

```
pi@raspberrypi:~$ vcgencmd measure_clock arm
frequency(45)=600000000
pi@raspberrypi:~$ vcgencmd measure_clock core
frequency(1)=250000000
pi@raspberrypi:~$ vcgencmd measure_clock uart
frequency(22)=48000000
pi@raspberrypi:~$ vcgencmd measure_clock emmc
frequency(47)=250000000
pi@raspberrypi:~$ vcgencmd measure_clock pixel
frequency(29)=0
pi@raspberrypi:~$ vcgencmd measure_clock vec
frequency(10)=108000000
pi@raspberrypi:~$ vcgencmd measure_clock hdmi
```

Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)

```
frequency(9)=0
pi@raspberrypi:~$ vcgencmd measure_clock h264
frequency(28)=250000000
pi@raspberrypi:~$ vcgencmd measure_clock isp
frequency(42)=250000000
```

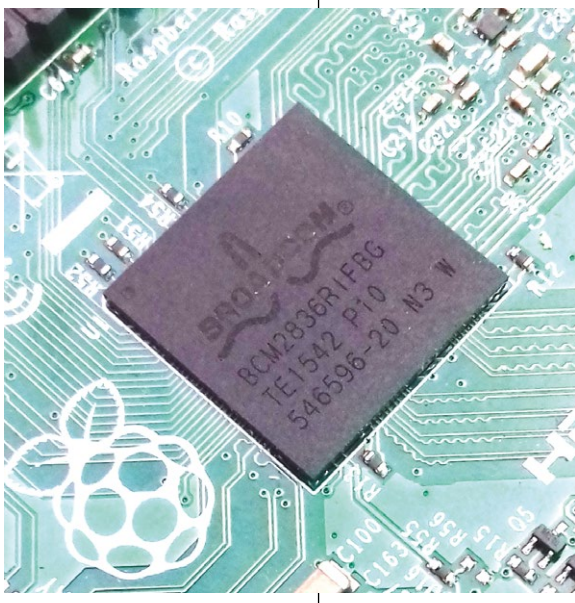
En écrivant un petit script Shell, il est possible d'afficher les fréquences de toutes les horloges. Ouvrons un éditeur de texte, **nano** par exemple suivi du nom du fichier à créer, ici **horloge**.

```
pi@raspberrypi:~$ nano horloge
```

Puis saisissons les lignes suivantes :

```
# !/bin/bash
for src in arm core h264 isp v3d uart pwm emmc pixel vec hdmi dpi ;
do
    echo -e "$src:\t$(vcgencmd measure_clock $src)" ;
done
```

Pour enregistrer le tout dans un fichier, faisons [CTRL] + [O], puis nous confirmons avec la touche [Entrée]. Pour quitter nano, appuyons sur les touches [CTRL] + [X]. Pour en savoir plus sur les scripts Shell, vous pouvez consulter le hors-série n°89 de *GNU/Linux Magazine* **[1]**. De même vous pouvez découvrir l'éditeur nano dans *Linux Pratique n°100* **[2]**.




```
pi@raspberrypi:~$ ./horloge
```

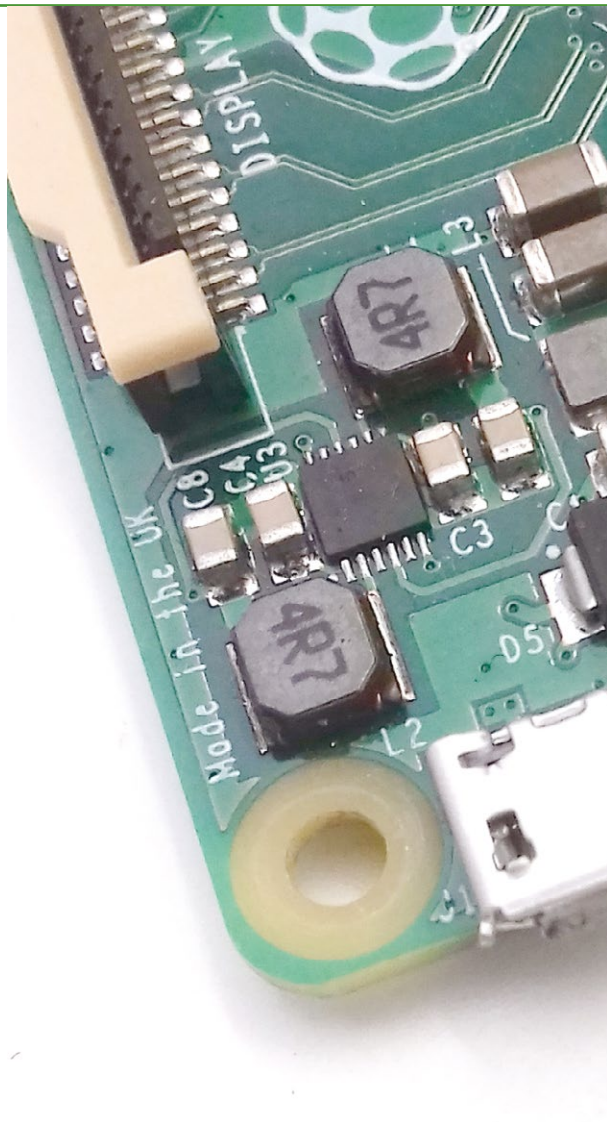
En utilisant la commande **measure_temp**, nous pouvons afficher la température du processeur.

```
pi@raspberrypi:~$ vcgencmd measure_temp
temp=40.8'C
```

La commande **get_config** est intéressante dans le sens où elle permet d'afficher tous les paramètres de configuration du Raspberry Pi. Paramètres que nous retrouvons dans le fichier **config.txt**. Nous verrons un peu plus tard dans cet article comment modifier et paramétrer le firmware du Raspberry Pi.

```
pi@raspberrypi:~$ vcgencmd get_config int
arm_freq=1200
audio_pwm_mode=1
config_hdmi_boost=5
core_freq=250
desired_osc_freq=0x36ee80
disable_commandline_tags=2
disable_l2cache=1
enable_uart=1
```

Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)




```
force_eeprom_read=1
force_pwm_open=1
framebuffer_ignore_alpha=1
framebuffer_swap=1
gpu_freq=300
hdmi_force_cec_address=65535
init_uart_clock=0x2dc6c00
lcd_framerate=60
over_voltage_avs=0x1cfde
overscan_bottom=32
overscan_left=32
overscan_right=32
overscan_top=32
pause_burst_frames=1
program_serial_random=1
sdram_freq=450
temp_limit=85
```

Il est également possible de connaître la mémoire allouée au processeur et celle allouée au GPU avec la commande **get_mem** suivie respectivement de **arm** ou **gpu**.

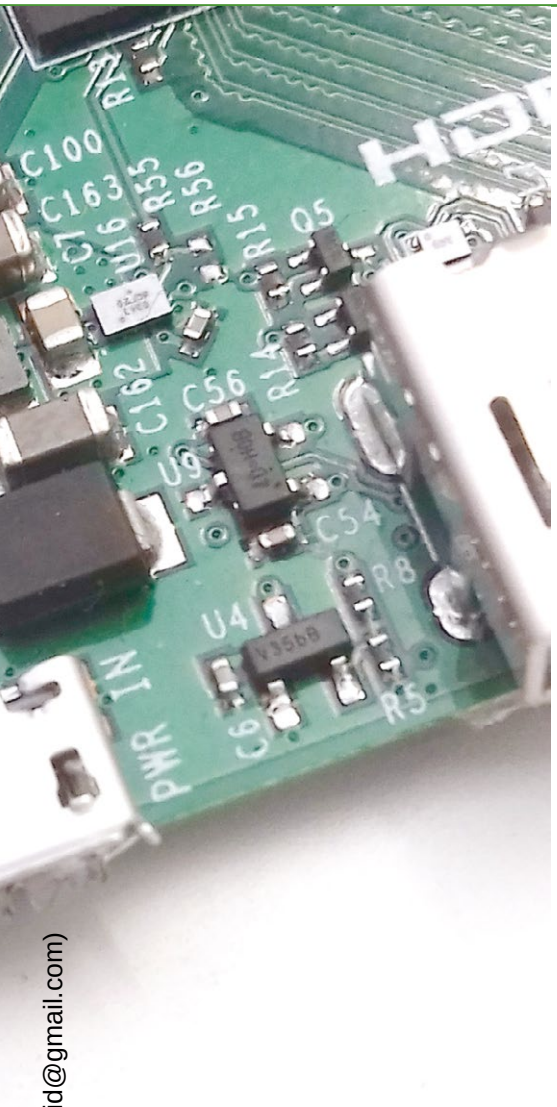
```
pi@raspberrypi:~$ vcgencmd get_mem arm
arm=944M
pi@raspberrypi:~$ vcgencmd get_mem gpu
gpu=64M
```

Une autre information utile, en particulier, lorsque nous souhaitons utiliser le Raspberry Pi comme lecteur multimédia et de vérifier quel codec vidéo est activé ou non avec la commande **codec_enabled** suivie du nom du codec à vérifier **h264 mpeg4 mpeg2 mjpg wm9**. Pour activer le codec MPEG2, il est nécessaire de faire l'acquisition d'une licence depuis le site officiel du Raspberry Pi.

```
pi@raspberrypi:~$ vcgencmd codec_enabled MPEG4
MPEG4=disabled
pi@raspberrypi:~$ vcgencmd codec_enabled H264
H264=enabled
pi@raspberrypi:~$ vcgencmd codec_enabled MPEG2
MPEG2=disabled
pi@raspberrypi:~$ vcgencmd codec_enabled MJPEG
MJPEG=disabled
```

Nous avons vu tout au long de cet article comment obtenir les paramètres et caractéristiques de notre Raspberry Pi en interrogeant directement le firmware comme cela peut être fait sur un ordinateur classique via le paramétrage du BIOS.

Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)



4.CONFIGURATION DU FIRMWARE

Non seulement nous pouvons lire les paramètres du firmware et ainsi accéder à nombre d'informations intéressantes, mais nous pouvons aussi configurer les paramètres du firmware comme cela peut être fait sur un ordinateur de bureau ou serveur dans le menu du BIOS. Nous avons mentionné en début d'article que le firmware est composé de trois fichiers, dont un de configuration **config.txt**. Dans un premier temps, nous allons ouvrir et lire ce fichier avec l'éditeur **nano**. Pour cela, plaçons-nous dans la partition VFAT où se trouve le fichier en question puis ouvrons le fichier.

```
pi@raspberrypi:~$ cd /boot
pi@raspberrypi:/boot$ nano config.txt
```

Chaque ligne est composée d'une expression **propriété = valeur**. Nous retrouvons les paramètres obtenus avec la commande **get_config**. Si vous avez fait l'acquisition d'une licence MPEG2 et désirez l'installer dans votre lecteur multimédia réalisé à partir Raspberry Pi (comme explicité dans le hors-série 38 de *Linux Pratique*), il suffit d'ajouter en fin du fichier avec votre éditeur de texte habituel la ligne suivante :

```
decode_MPG2=0x12345678 #remplacez 0x12345678 par votre clé
```

CONCLUSION

Il est possible d'aller plus loin dans l'accès et surtout dans la mise en forme des résultats obtenus à partir des données du firmware en faisant appel aux commandes présentées dans cet article via l'écriture de scripts Shell, Python... Pour cela, vous pouvez notamment vous référer aux hors-séries n°86 et 89 de *GNU/Linux Magazine* **[1] [4]**. LD

RÉFÉRENCES

[1] *GNU/Linux Magazine Hors-série n°89*, « Maîtrisez la programmation script Shell », **https://boutique.ed-diamond.com/home/1231-gnulinux-magazine-hs-89.html**

[2] *Linux Pratique n°100*, « Nano : l'éditeur petit, mais costaud », **https://boutique.ed-diamond.com/home/1219-linux-pratique-100.html**

[3] *Linux Pratique Hors-série n°38*, « C'est décidé, je débute sous Linux avec la Raspberry Pi », **https://boutique.ed-diamond.com/en-kiosque/1234-linux-pratique-hs-38.html**

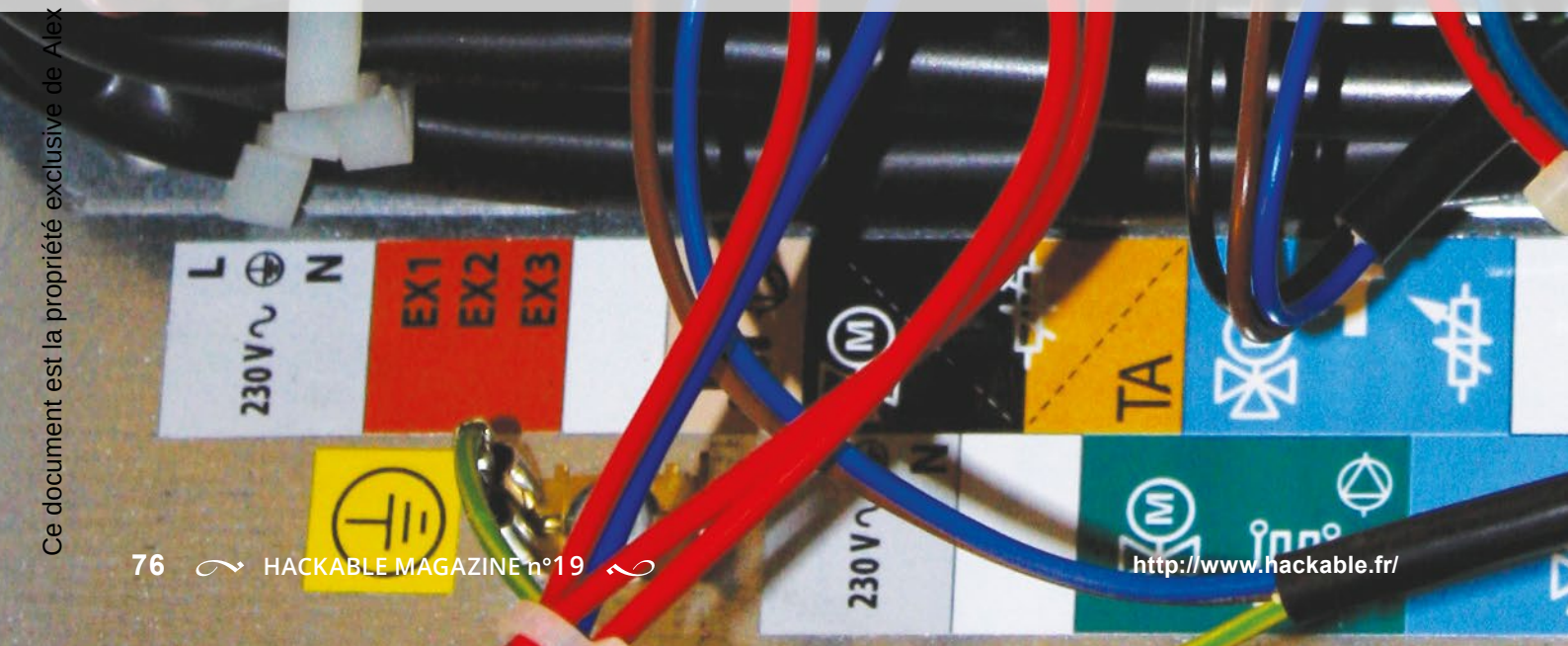
[4] *GNU/Linux Magazine Hors-série n° 86*, « Mémo Python », **https://boutique.ed-diamond.com/anciens-numeros/1085-linux-pratique-98.html**

Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)

ANALYSER LE BUS SIEMENS BSB D'UNE POMPE À CHALEUR ATLANTIC

Erwan Loaëc

Les pompes à chaleur installées dans beaucoup de constructions récentes sont de plus en plus évoluées. Malheureusement, ces systèmes complexes sont aussi totalement « fermés », et ne permettent pas d'être pilotés aussi librement qu'on le souhaiterait. Cet article montre qu'après avoir analysé la façon dont communique la PAC avec la sonde d'ambiance, il devient possible d'accéder à presque toutes ses fonctionnalités.



Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)

Installé dans une maison neuve depuis quelque temps, un élément indispensable au confort m'échappait complètement jusqu'à ce que je me penche sérieusement sur son cas. Il s'agit de la pompe à chaleur. Celle-ci assure la production d'eau chaude sanitaire toute l'année, ainsi que l'alimentation d'un plancher chauffant.

1. CONTEXTE

Je tiens à préciser que tout le travail décrit dans cet article n'est valable que pour mon cas, je ne prétends pas avoir mis en place un système générique qui puisse s'appliquer à toutes les pompes à chaleur, ni même à toutes celles de la marque.

L'étude porte sur une pompe à chaleur **Atlantic Alfea Extensa Duo+**. Sa particularité est d'intégrer un réservoir d'eau de 190 litres afin de prendre en charge la production d'eau chaude sanitaire.

Dans le salon, une sonde d'ambiance **Atlantic T55**, directement reliée à la PAC, permet de mesurer la température intérieure, ainsi que de modifier le mode de fonctionnement et la consigne de chauffage.

Figure 2 : Sonde d'ambiance Atlantic T55, installée dans la pièce de vie. Ce module permet de régler la consigne de température ainsi que le mode de fonctionnement du chauffage.

En ouvrant le boîtier électrique du module intérieur de la pompe à chaleur, on distingue une carte qui semble servir d'automate, avec la référence **SIEMENS RVS21.827/127**. La sonde intérieure est reliée à cette carte (Figure 3, page suivante).

Après quelques recherches sur Internet, il semblerait qu'il existe un module Siemens, désigné sous la référence **OZW672**, qui permette de connecter le système au réseau et de piloter l'ensemble des paramètres à travers un serveur web. Ce module ne semble pas faire partie du

Figure 1 : Module intérieur de la pompe à chaleur Atlantic Alfea Extensa Duo+ sur lequel on distingue le panneau de contrôle en haut à droite.

Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)

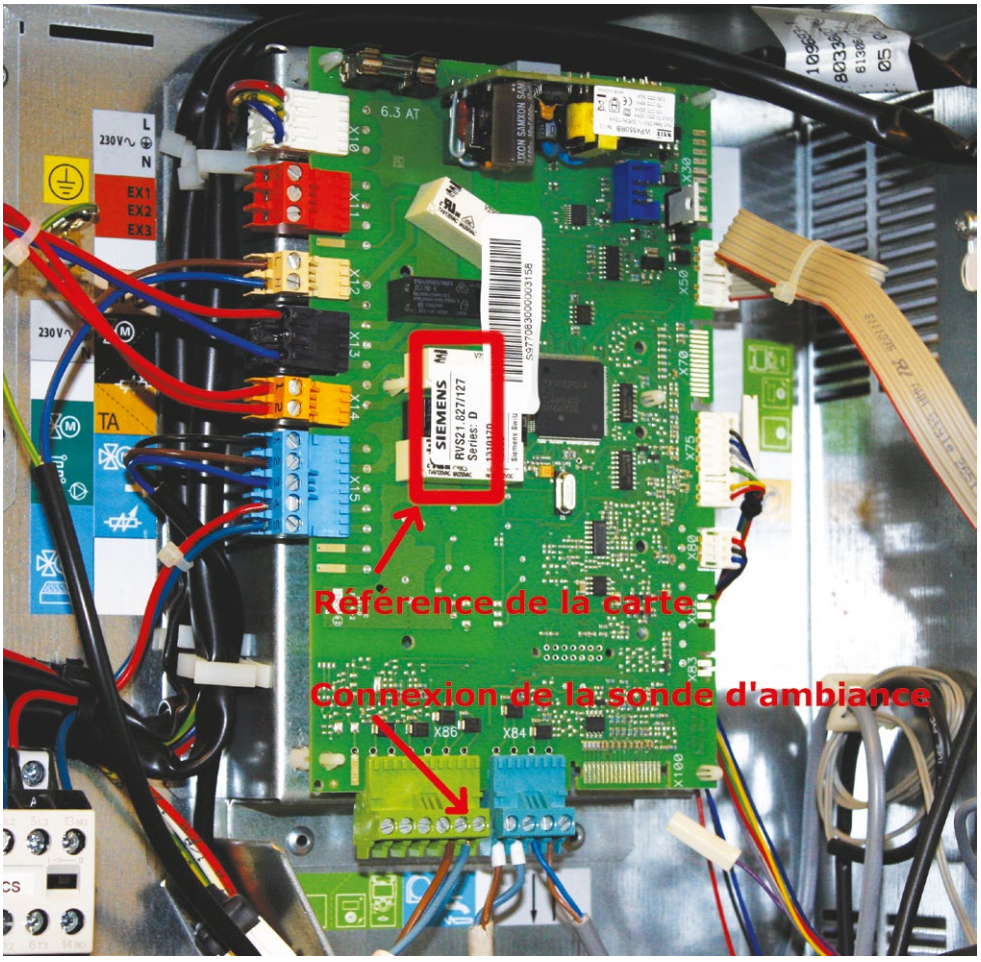


Figure 3 : Carte Siemens RVS21.827/127. Il s'agit vraisemblablement de l'automate qui pilote l'ensemble des actionneurs de la PAC. La sonde d'ambiance est câblée sur le bornier vert X86 localisé sur le bas de la carte.

catalogue Atlantic, qui est le fabricant de ma pompe à chaleur. Il existe un échange assez complet sur un forum qui évoque le pilotage de ce type de pompe à chaleur depuis ce module **[1]**.

Le principal obstacle à cette solution est son prix assez élevé. Il semble que pour l'obtenir il faille débourser autour de 400 euros. Tenter de réaliser un équivalent soi-même est bien sûr amusant, mais présente également un intérêt économique.

2. ANALYSE DU SIGNAL

La première idée qui m'est venue est de tenter d'analyser le signal qui transite sur la connexion entre la sonde intérieure (le module T55) et la carte Siemens. En effet, ce module permet de récupérer la consigne d'ambiance, la modifier, ainsi que changer le mode de fonctionnement (arrêt, confort, réduit...). Cela signifie que ces informations transitent forcément vers ce module.

La sonde intérieure est connectée avec 2 fils, et ne contient pas de pile ou batterie. Cela signifie que l'alimentation et la transmission de données utilisent la même paire.

En plaçant un multimètre sur les 2 bornes de la sonde, on constate une tension continue de 11,48V (sans la sonde, 11,62V). Toutes les quelques secondes, il semble se produire une chute de tension très brève, probablement créée par une transmission de données.

Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)

Figure 4 : Extrait de la capture réalisée sur le bus avec un analyseur logique qui met en évidence une transmission de données.

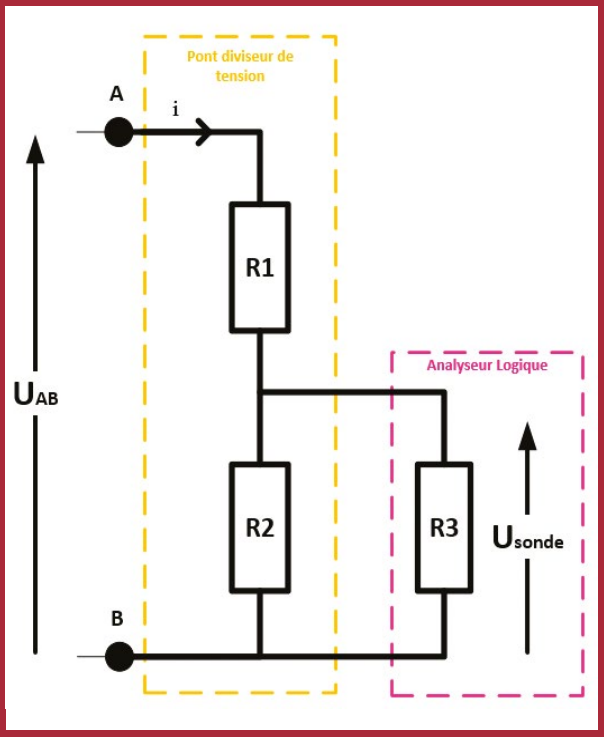
Afin de comprendre ce qui se passe exactement au moment des chutes de tension constatées au multimètre, je suis passé par un analyseur logique compatible avec le logiciel de Saleae Logic. Vous pouvez retrouver un article complet consacré aux analyseurs logiques dans le numéro 5 de *Hackable Magazine* de 2015.

En lançant une capture de 10 secondes, avec 1 million d'échantillons par seconde, on peut identifier 3 séquences de transmission distinctes (voir Figure 4).

Les séquences capturées durent entre 25ms et 35ms. Le changement de niveau le plus rapide dure 208ns. D'après le théorème de Shannon **[2]**, on peut donc en déduire qu'un échantillonnage tous les 100ns serait suffisant pour capturer le signal, c'est-à-dire 10000 échantillons par seconde. Pour garder une marge, les captures seront réalisées avec 25000 échantillons par seconde, au lieu des 1 million pour la première capture.

Notez qu'il est nécessaire de veiller à ne pas dépasser la tension admissible sur les entrées de l'analyseur logique. Les sondes Saleae Logic acceptent une tension maximale de 5,5V (5,25V pour les anciennes versions) [3], avec une impédance de 1MOhms. Il existe beaucoup de clones de ces sondes, et les caractéristiques peuvent varier, en particulier la résistance interne.

Dans le cas où le signal que l'on souhaite capturer dépasse la tension maximale, il est alors nécessaire d'adapter le montage afin de ne pas risquer d'endommager l'analyseur. La solution la plus simple lorsqu'elle est applicable est probablement de mettre en place un pont diviseur de tension avec deux résistances R1 et R2. La résistance R3 matérialise l'impédance interne de l'analyseur.



Si on néglige la valeur de R3 qui est généralement élevée, il faut donc que `(R1+R2)/R2 > U(AB)/U(sonde)`. Avec `U(AB)`, la tension maximale du point d'analyse, et `U(sonde)`, la tension maximale admissible par l'analyseur.

Selon le contexte de l'analyse, il peut être nécessaire de veiller à ne pas drainer un courant trop important (`i`), ce qui reviendrait à choisir des valeurs faibles pour R1 et R2, ou à l'inverse de travailler avec des valeurs trop élevées qui rendraient l'impédance interne R3 de la sonde non négligeable.

Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)

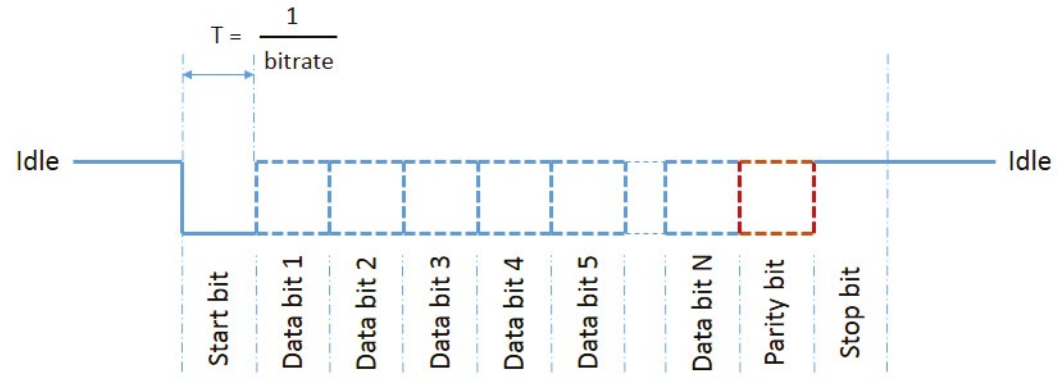


Figure 5 : Format d'une trame UART (Universal Asynchronous Receiver Transmitter).

3. DÉCODAGE DES TRAMES

Comme évoqué précédemment, la période mesurée la plus courte pour un état logique est 208ns, ce qui nous donne une fréquence de 4808Hz. Cette fréquence rappelle une des fréquences standards d'une communication de type UART, 4800 bps **[4]**, c'est donc la piste que nous allons suivre dans un premier temps.

Il est peut-être bon de commencer par un rappel du fonctionnement des communications UART.

Lorsqu'il n'y a aucune transmission, la ligne reste à un niveau haut. Chaque bit est émis pendant une période T qui dépend de la vitesse de transmission. Pour une vitesse de 4800 bits/s, la durée T est égale à 208ns (1/4800).

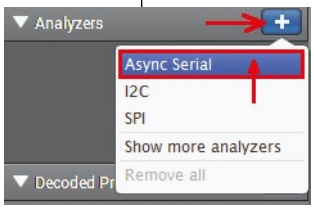


Figure 6 : Création d'un analyzer depuis le logiciel Saleae Logic.

Chaque trame est composée des éléments suivants :

- 1 bit de départ *(Start bit)*, qui correspond à un niveau bas ;
- N bits de données ;
- 1 bit de parité, optionnel. Il peut s'agir d'une parité paire *(even)* ou impaire *(odd)* ;
- 1 bit de fin *(Stop bit)*, qui correspond à un niveau haut. Ce bit de fin peut être émis pendant une durée T, 1,5T ou 2T.

Le bit de départ est systématiquement précédé par un état haut. En effet, soit il s'agit de la première trame, dans ce cas le canal de communication est en « *idle* » sur un niveau haut, soit il est précédé par un bit de fin, caractérisé également par un niveau haut.

Ces éléments vont nous aider à décoder les séquences capturées sur le bus.

Depuis le logiciel Logic, nous créons un « Analyseur » de type « Communication Série Asynchrone ».

Dans les options de paramétrage, nous allons juste choisir de configurer la vitesse de transmission automatique ***Use Autobaud*** :

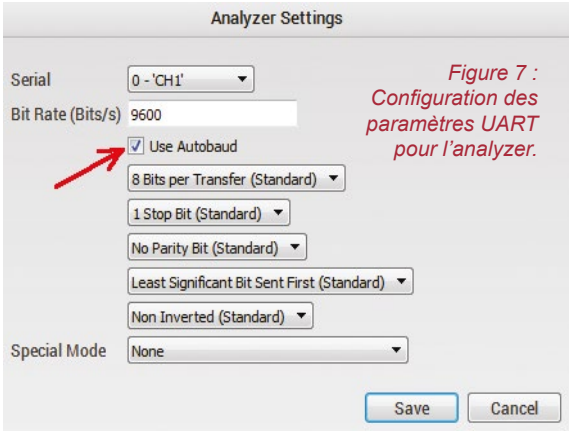


Figure 7 : Configuration des paramètres UART pour l'analyzer.

Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)

Une fois la configuration validée, on constate que le signal capturé ne correspond pas vraiment à l'*analyzer* configuré. La présence d'erreurs du type « framing error » indique que le format de la trame n'est pas valide :

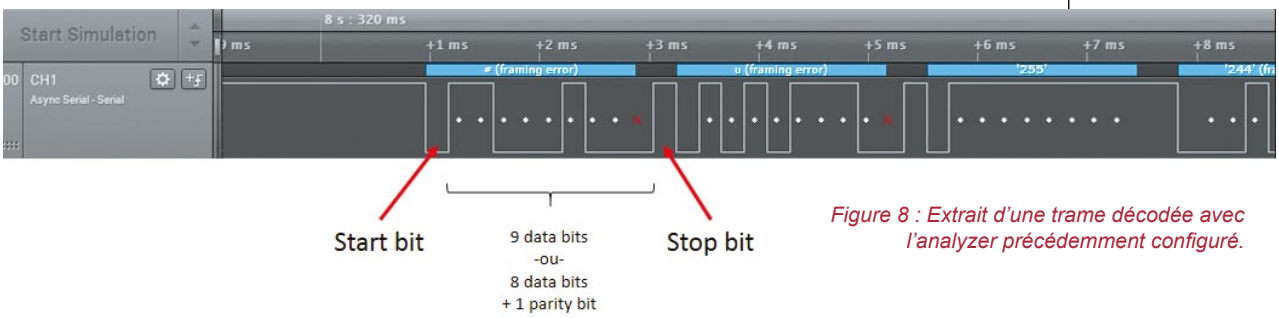


Figure 8 : Extrait d'une trame décodée avec l'analyzer précédemment configuré.

L'*analyzer* repère bien le « *start bit* » et le « *stop bit* », mais entre ces deux repères se trouvent 9 bits. Il peut donc s'agir soit de 9 bits de données, soit de 8 bits de données suivis d'un bit de parité. Cette dernière possibilité semble plus plausible.

Nous retournons donc sur la configuration de notre *analyzer*, afin d'y ajouter le bit de parité. Et là, bingo ! Avec une parité impaire *(odd parity)*, toutes les trames capturées sont valides. Il n'y a donc vraisemblablement aucune chance qu'il s'agisse de transferts avec 9 bits de données.

Voici ce que nous pouvons désormais en déduire sur les caractéristiques du bus :

• vitesse de transmission : 4800 bits/seconde ;

• 8 bits de données ;

• 1 bit de parité impaire ;

• 1 bit de fin.

Il nous reste maintenant à identifier le sens des données transmises dans les trames. Il s'agit notamment de déterminer si le bit 1 correspond au bit de poids faible (LSB) ou au bit de poids fort (MSB).

Afin d'avancer sur l'analyse, je vais lancer une capture « naïve », et incrémenter la consigne de chauffage au fur et à mesure de 17°C à 22°C avec un pas de 0,5°C, puis revenir à 17°C. Avec un peu de chance, on pourra identifier des valeurs qui s'incrémentent, puis décrémentent.

Cette séquence correspond donc à l'affectation successive de 20 valeurs de consigne (la première valeur modifiée est 17,5°C).

Depuis le logiciel Logic, une longue capture de 2min30s permet d'enregistrer toutes ces actions. Grâce à l'*analyzer* configuré, il est possible de directement générer un fichier CSV avec l'ensemble des valeurs hexadécimales récupérées.

En observant les séquences, on remarque qu'elles commencent toutes par « 23 75 », « 23 7F » ou « 23 79 ». La commande suivante permet de retrouver une séquence sur chaque ligne :

```
$> cat raw_temp.csv | cut -d "," -f 3 | sed "s/0x//g" | tr '\n' ' ' | sed -e
"s/ 23 75/\n23 75/g" -e "s/ 23 79/\n23 79/g" -e "s/ 23 7F/\n23 7F/g" > raw
```

Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)

Étant donné l'approche, qui consiste à partir d'une consigne de 17°C jusqu'à 22°C, puis à revenir à 17°C, chaque valeur sera donc définie deux fois, à l'exception de 22°C et 17°C. Observons donc les séquences capturées une ou deux fois :

```
$> cat raw | sort | uniq -c | grep -E "\ (1|2)\ "
```

Ces séquences se ressemblent beaucoup, on peut notamment remarquer :

- 20 séquences commençant par « 23 75 FF F1 FC C2 D2 FA 71 » ;
- 20 séquences commençant par « 23 7F F5 F1 F8 D2 C2 FA 71 ».

Il y a donc très probablement un lien avec les 20 valeurs de consignes différentes que nous venons de réaliser.

Essayons d'analyser les 20 séquences qui commencent par « 23 75 FF F1 FC C2 D2 FA 71 ».

```
23 75 FF F1 FC C2 D2 FA 71 FE FB 9F 9A 0C → 17,5°C ?
23 75 FF F1 FC C2 D2 FA 71 FE FB 7F 67 22 → 18,0°C ?
23 75 FF F1 FC C2 D2 FA 71 FE FB 5F 43 40 → 18,5°C ?
23 75 FF F1 FC C2 D2 FA 71 FE FB 3F 2F E6 → 19,0°C ?
23 75 FF F1 FC C2 D2 FA 71 FE FB 1F 0B 84 → 19,5°C ?
23 75 FF F1 FC C2 D2 FA 71 FE FA FF C5 9B → 20,0°C ?
23 75 FF F1 FC C2 D2 FA 71 FE FA DF E1 F9 → 20,5°C ?
23 75 FF F1 FC C2 D2 FA 71 FE FA BF 8D 5F → 21,0°C ?
23 75 FF F1 FC C2 D2 FA 71 FE FA 9F A9 3D → 21,5°C ?
23 75 FF F1 FC C2 D2 FA 71 FE FA 7F 54 13 → 22,0°C ?
23 75 FF F1 FC C2 D2 FA 71 FE FA 9F A9 3D → 21,5°C ?
23 75 FF F1 FC C2 D2 FA 71 FE FA BF 8D 5F → 21,0°C ?
23 75 FF F1 FC C2 D2 FA 71 FE FA DF E1 F9 → 20,5°C ?
23 75 FF F1 FC C2 D2 FA 71 FE FA FF C5 9B → 20,0°C ?
23 75 FF F1 FC C2 D2 FA 71 FE FB 1F 0B 84 → 19,5°C ?
23 75 FF F1 FC C2 D2 FA 71 FE FB 3F 2F E6 → 19,0°C ?
23 75 FF F1 FC C2 D2 FA 71 FE FB 5F 43 40 → 18,5°C ?
23 75 FF F1 FC C2 D2 FA 71 FE FB 7F 67 22 → 18,0°C ?
23 75 FF F1 FC C2 D2 FA 71 FE FB 9F 9A 0C → 17,5°C ?
23 75 FF F1 FC C2 D2 FA 71 FE FB BF BE 6E → 17,0°C ?
```

Figure 9 : Séquences capturées, codées en hexadécimal.

Il y a 4 octets qui varient sur chaque ligne. Sur la figure 9, ceux identifiés en rouge semblent évoluer de façon régulière, et pourraient correspondre au codage de la température. Les octets identifiés en mauve ne semblent correspondre à aucune logique visible.

Dans la suite de données identifiée en rouge, si on se focalise sur le second octet (celui de « droite »), on constate que plus la température augmente, plus la valeur décodée diminue, et inversement. Si le bit de poids fort se trouvait au début de la trame de 8 bits, on obtiendrait les valeurs suivantes :

- **0x9F** devient **0xF9** (17,5°C) ;
- **0x7F** devient **0xFE** (18,0°C) ;
- **0x5F** devient **0xFA** (18,5°C) ;
- **0x3F** devient **0xFC** (19,0°C) ;
- **0x1F** devient **0xF8** (19,5°C) ;
- **0xFF** devient **0xFF** (20,0°C) ;
- etc.

La suite **0xF9**, **0xFE**, **0xFA**, **0xFC**, **0xF8**, **0xFF** n'est ni croissante, ni décroissante. Nous pouvons continuer l'analyse en partant du principe que le bit de poids fort se trouve bien à la fin de la trame de 8 bits.

Depuis le début, nous avons pris l'hypothèse que le niveau haut correspondait à un « 1 », et le niveau bas à un « 0 ». Et si c'était l'inverse ? Calculons le complément à 1, ce qui correspond à une permutation des valeurs 0 et 1, pour les octets repérés en rouge dans la figure 9 :

- **0xFB 0x9F** devient **0x04 0x60** = 1120 (17,5°C) ;
- **0xFB 0x7F** devient **0x04 0x80** = 1152 (18,0°C) ;
- **0xFB 0x5F** devient **0x04 0xA0** = 1184 (18,5°C) ;
- **0xFB 0x3F** devient **0x04 0xC0** = 1216 (19,0°C) ;
- **0xFB 0x1F** devient **0x04 0xE0** = 1248 (19,5°C) ;
- **0xFA 0xFF** devient **0x05 0x00** = 1280 (20,0°C) ;

Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)

- `0xFA 0xDF` devient `0x05 0x20` = 1312 (20,5°C) ;
- etc.

Cette fois-ci nous retrouvons bien une valeur croissante codée sur 2 octets. En recherchant la valeur décimale que représentent ces 2 octets, on constate qu'il s'agit de la température en Celsius multipliée par 64.

Nous savons désormais que les données binaires transmises sur le bus sont codées avec un niveau haut pour le « 0 » et un niveau bas pour le « 1 ».

Revenons maintenant sur les valeurs qui étaient repérées dans la zone mauve, après avoir calculé le complément à 1. Sur la première ligne que nous avions extraite, `23 75 FF F1 FC C2 D2 FA 71 FE FB 9F 9A 0C` devient `DC 8A 00 0E 03 3D 2D 05 8E 01 04 60 65 F3`.

Il semble plausible que les 2 derniers octets représentent un CRC (ou une somme de contrôle). En utilisant l'outil en ligne **[5]**, il est facile de lister les CRC obtenus avec les principales méthodes de calcul :

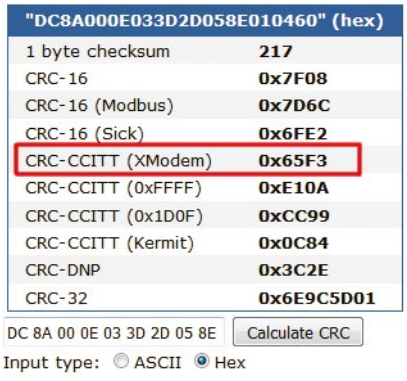


Figure 10 : Principales valeurs de CRC obtenues depuis l'outil en ligne sur www.lammertbies.nl.

Il existe donc une valeur standard de CRC qui vaut `0x65 0xF3`, ce qui correspond bien aux 2 derniers octets de la chaîne analysée, après le complément à 1. En faisant la même opération sur d'autres séquences, le `CRC-CITT XModem` correspond bien aux 2 derniers octets.

CONCLUSION

En partant de moyens relativement simples, nous avons démontré qu'il était possible de mener une rétro-ingénierie sur un bus « fermé ». Nous sommes désormais capables d'isoler les messages qui transitent et de vérifier leur intégrité. En faisant l'analogie avec le modèle OSI **[6]**, nous pouvons dire que nous avons décodé la couche 1 (couche *Physique*), et une partie de la couche 2 (couche *Liaison*).

Il reste cependant du chemin à parcourir afin d'obtenir la maîtrise globale du système. Il est temps de laisser l'analyseur logique de côté pour pouvoir directement accéder au bus depuis une Raspberry Pi. De cette manière, il sera possible d'analyser plus précisément le contenu des messages qui transitent sur le bus afin de continuer à décoder son fonctionnement. **EL**

RÉFÉRENCES

[1] http://www.chaleurterre.com/forum/viewtopic.php?f=59&t=15784

[2] https://fr.wikipedia.org/wiki/Th%C3%A9or%C3%A8me_d'%C3%A9chantillonnage

[3] https://www.saleae.com/#DatasheetTile et https://www.saleae.com/originallogic/specs

[4] https://fr.wikipedia.org/wiki/UART

[5] https://www.lammertbies.nl/comm/info/crc-calculation.html

[6] https://fr.wikipedia.org/wiki/Mod%C3%A8le_OSI

Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)

PILOTEZ VOTRE POMPE À CHALEUR ATLANTIC EN UTILISANT LE BUS SIEMENS BSB

Erwan Loaëc

L'article précédent a montré qu'en partant de l'analyse du signal brut entre la sonde d'ambiance et la PAC, nous sommes désormais capables de décoder les trames échangées. Maintenant que les couches « basses » sont en partie traitées, nous allons nous pencher sur le décodage des messages. À l'issue de cet article, nous serons capables de lire les informations échangées sur le bus depuis une Raspberry Pi, mais également de transmettre des données afin d'agir sur le fonctionnement de la PAC.

Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)

L'étude du signal qui transite sur le bus a été réalisée à partir d'un analyseur logique. Maintenant que nous connaissons les caractéristiques de la liaison, nous allons implémenter une solution permettant de relier le bus à une plateforme afin de lire, puis de transmettre les messages.

La lecture des données « brutes » directement depuis notre code facilitera grandement le décodage des messages. En effet, il reste encore du chemin à parcourir avant de maîtriser le fonctionnement du système. Nous allons notamment étudier comment fonctionne l'adressage, mais également les différents types de messages : action sur la PAC, interrogation sur une valeur, etc. Enfin, une fois le format connu, nous allons transmettre nos propres messages. De cette façon il sera possible d'agir sur la PAC, permettant ainsi l'intégration du chauffage dans votre domotique.

1. CONNEXION AVEC UNE RASPBERRY PI

1.1 Attention

Le montage proposé dans cet article sera directement connecté à la pompe à chaleur. Il s'agit d'un montage expérimental qui n'est absolument pas supporté par le fabricant. Si vous souhaitez mettre en œuvre un montage similaire à celui décrit dans l'article, vous devrez assumer les dégâts éventuels qui pourraient être causés sur votre installation de chauffage.

1.2 Réception des trames

La Raspberry Pi permet d'utiliser une connexion UART en câblant un port de réception (RX), et un port de transmission (TX). La liaison UART utilise une tension de 0-3,3V, contrairement à la liaison que nous étudions qui fonctionne avec les valeurs 0-12V.

Afin de ne pas perturber le fonctionnement du bus, nous allons concevoir un montage permettant d'isoler électriquement la carte Raspberry Pi de la carte Siemens de la pompe à chaleur, principalement au moyen d'optocoupleur.

Intéressons-nous dans un premier temps à la lecture du signal sur le bus. Lorsque la tension du bus sera de 12V, le courant passant dans la led de l'optocoupleur à travers **RX-R1** sera d'environ 5mA. Le phototransistor sera passant. Ainsi, l'émetteur relié à l'entrée RX de la Raspberry Pi aura une tension proche de celle de l'alimentation c'est-à-dire 3,3v. Lors d'une transmission d'un signal sur le bus Siemens, une tension nulle aura pour effet de bloquer le phototransistor de l'optocoupleur. La tension RX sera donc reliée à la masse à travers la résistance de *Pull-Down* **RX-R2**.

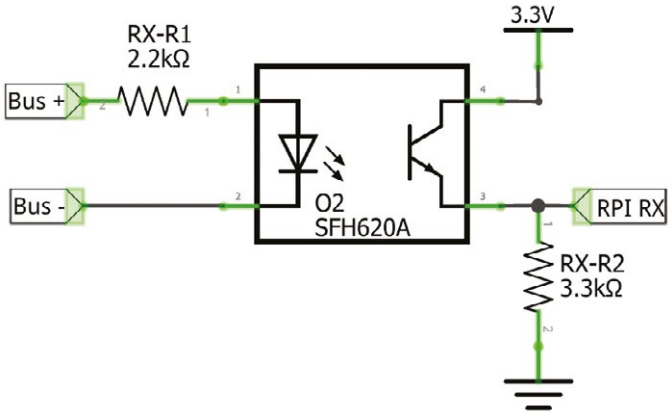


Figure 1 : Schéma du montage électronique utilisé pour la lecture sur le bus.

On notera que ce montage va tirer un courant permanent sur le bus de la PAC d'environ 5mA, limité par la résistance **RX-R1**.

1.3 Transmission de données

Lors de mon étude, je me suis dans un premier temps penché plus longuement sur la lecture des données qui transitent sur le bus depuis la Raspberry Pi, avant d'avancer sur la transmission de données sur le bus. Afin de garder de la

Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)

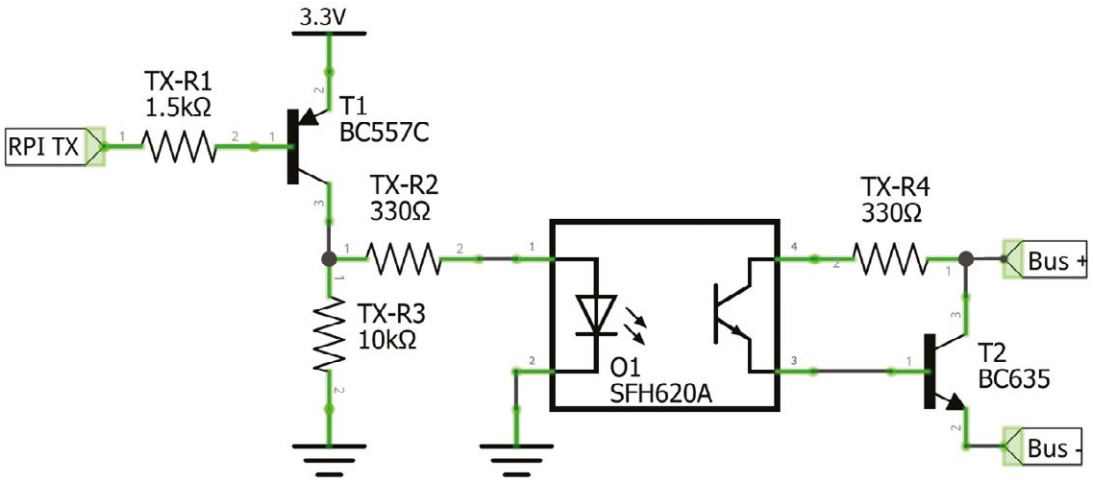


Figure 2 : Schéma du montage électronique utilisé pour la transmission sur le bus, légèrement plus complexe que celui utilisé pour la lecture des données.

cohérence dans l'article, je vous propose d'aborder directement le montage qui va nous permettre de transmettre un signal sur le bus.

Nous allons utiliser de nouveau un optocoupleur pour isoler électriquement la Raspberry Pi du bus sur lequel nous voulons agir.

Lorsqu'aucune donnée n'est transmise, la tension sur le bus est de 12V. La transmission d'un état bas correspond à la mise en court-circuit du bus de données. Cela peut correspondre au bit de start, ou à un bit de données. À 4800 bits/seconde, chaque occurrence ne dure que 208ns (potentiellement répétée plusieurs fois de suite).

Un transistor PNP installé derrière la sortie TX permet d'inverser le signal, et de n'alimenter la led de l'optocoupleur que lorsque TX passe à l'état bas. De cette façon, le courant tiré depuis TX ne dépasse pas 3,3V/1500 Ohms, c'est-à-dire 2,2mA. Ce courant est tout à fait acceptable pour la Raspberry Pi.

Concernant la connexion sur le bus, un second transistor installé derrière l'optocoupleur permet d'encaisser la forte charge de courant pendant le laps de temps très bref de transmission d'un état bas, qui correspond ici à la mise en court-circuit du bus. Il est probable que la commutation aurait pu être

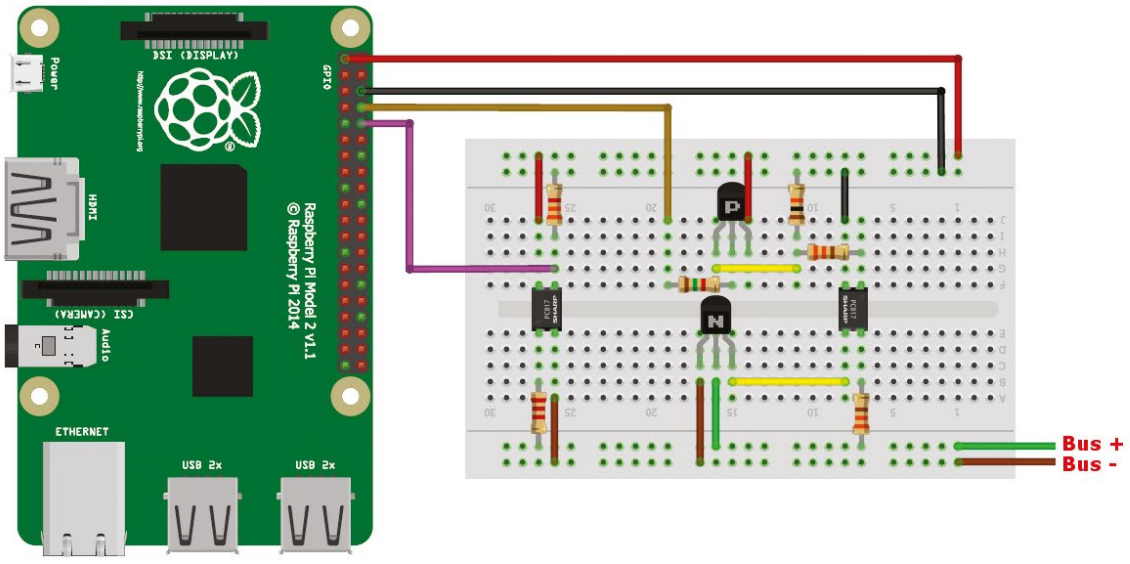


Figure 3 : Le câblage des quelques composants est réalisé sur une platine d'expérimentation à côté de la Raspberry Pi.

Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)

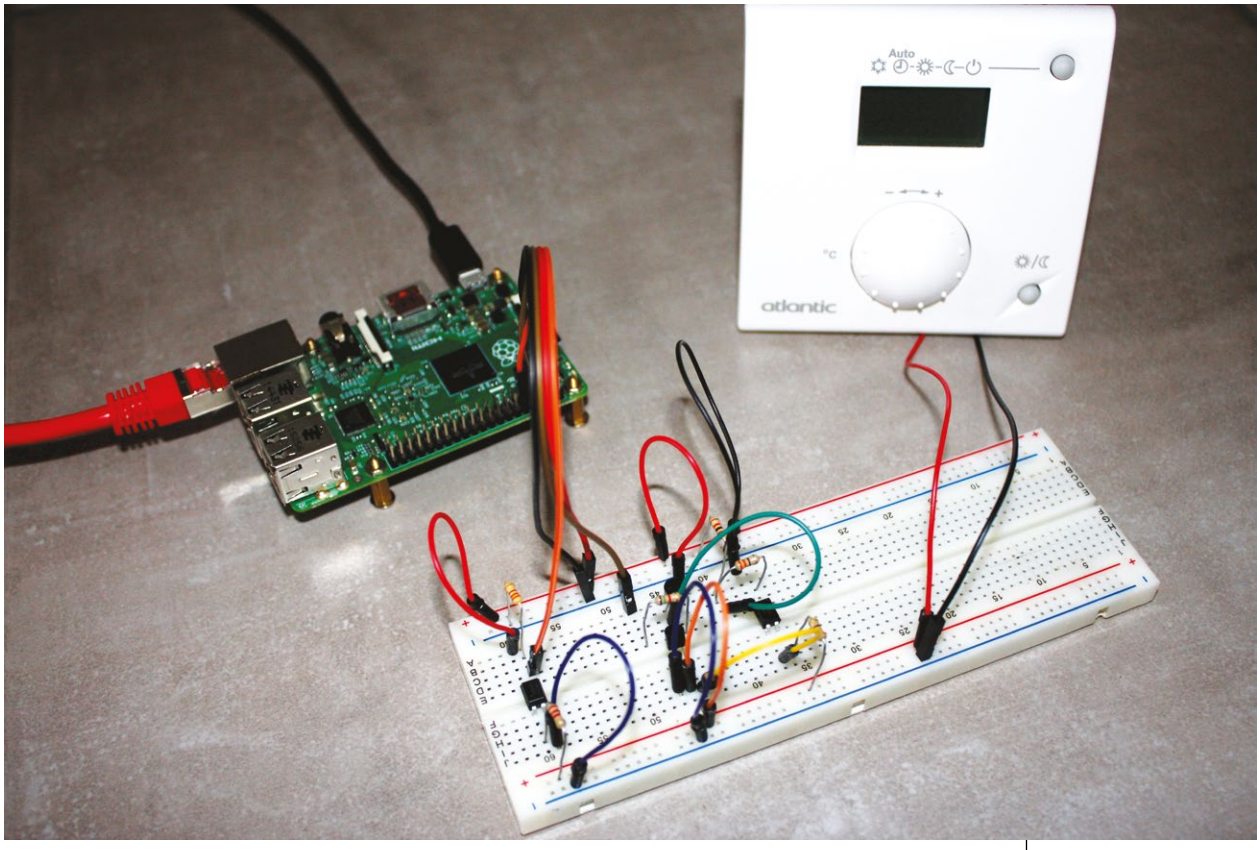


Figure 4 : Photo du montage réalisé avec la Raspberry Pi et la platine à essai avec les résistances, transistors et optocoupleurs nécessaires. La platine à essai est connectée au bus en passant par la sonde d'ambiance. Il est possible de la connecter directement sur le bornier de la carte Siemens situé dans le coffret électrique de la pompe à chaleur.

directement prise en charge par l'optocoupleur, étant donné que le courant n'y transite que pendant un temps très court. J'avais dans mes cartons un transistor NPN capable de traiter un courant jusqu'à 500mA, autant en profiter.

Tant que l'on ne transmet pas de données, ce montage ne tire aucun courant sur le bus.

Nous sommes ici dans une situation particulière où l'UART permet une communication full-duplex, c'est-à-dire que la transmission et la réception de données sont toutes les deux indépendantes. Chaque signal est câblé sur une broche spécifique, ce qui n'est pas le cas du bus de la pompe à chaleur sur lequel nous allons nous câbler.

En effet, si l'on décide d'envoyer des données, alors qu'au même moment un autre élément connecté est déjà en train de transmettre sur le bus, il y aura ce qu'on appelle une collision. Plus le bus est « occupé », plus le risque de collision devient élevé. Le contrôle de parité directement au niveau de l'UART permettra de ne pas recevoir les trames perturbées. De même, le contrôle de la valeur du CRC, mise en évidence précédemment, permettra de valider l'intégrité des messages, ignorant ainsi les données récupérées lors des collisions.

2. CONFIGURATION DE L'UART SUR LA RASPBERRY PI

2.1 Préparation du système

Nous allons utiliser ici une carte Raspberry Pi 2, mais le montage doit pouvoir fonctionner sur toutes les versions. Pour l'OS, nous allons partir sur la dernière version de Raspbian disponible à l'heure où je rédige cet article, **Raspbian Jessie Lite**. Par défaut, la liaison UART est utilisée comme console de l'OS. Afin de pouvoir l'utiliser en dehors de ce cadre, il faut commencer par désactiver la console sur l'UART.

Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)

En premier lieu, nous éditons le fichier **/boot/cmdline.txt** afin de supprimer les paramètres **console=serial0,115200**. Pour prendre en compte ces changements, il faudra veiller à redémarrer le système.

Ensuite, en dehors des composants logiciels déjà présents dans la distribution, nous aurons besoin de deux bibliothèques Python supplémentaires, *serial* et *crcmod* :

```
$> apt-get install python-serial python-crcmod
```

Sur la Raspberry Pi, l'UART correspond au périphérique **/dev/AMA0**. Le paramétrage de l'UART se fait en utilisant la commande **stty**. Cette commande n'est pas persistante, il faudra donc veiller à la relancer après un redémarrage ou l'inscrire par exemple dans le fichier **/etc/rc.local**.

```
$> stty -F /dev/ttyAMA0 speed 4800 parenb parodd cs8 -cstopb -echo
```

- **speed 4800** : vitesse configurée à 4800 bps ;
- **parenb parodd** : utilisation d'un bit de parité impaire ;
- **cs8** : chaque trame contient 8 bits de données ;
- **-cstopb** : 1 bit de stop est envoyé à la fin de chaque trame ;
- **-echo** : désactivation de l'*echo* des données transmises. Tel qu'est conçu le montage, les données transmises seront également lues. Il est donc important de désactiver l'*echo*, sous peine de se retrouver avec une transmission en boucle des données.

2.2 Lecture des données

Depuis quelques lignes de Python, nous allons lire les données reçues sur l'UART du Raspberry Pi :

```
 1: #!/usr/bin/python
 2: #! -*- coding=utf-8 -*-
 3:
 4: import serial # Module qui va nous permettre de manipuler l'UART
 5: import sys
 6:
 7: ser = serial.Serial('/dev/ttyAMA0', 4800, timeout=1, bytesize=8,
 8:                     stopbits=1, parity=serial.PARITY_ODD)
 9:
10: while True: # Boucle infinie
11:
12:     data = ser.read()
13:     datalen = len(data)
14:
15:     if datalen > 0:
16:
17:         for idx in range(datalen):
18:
19:             byteVal = 255 - ord(data[idx]) # Inversion des bits
20:
21:             sys.stdout.write("%0.2X " % (byteVal)) # Affiche les valeurs
hexadécimales lues
22:             sys.stdout.flush()
```

Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)

Nous commençons par instancier un objet **Serial**, en déclarant tous les paramètres de la liaison (vitesse, parité, nombre de bits de données...). Puis, nous allons lire en boucle les données de l'UART, en inversant les bits de chaque octet lu. L'inversion des bits d'un octet se fait facilement en soustrayant la valeur lue à 255 (ligne 19).

Afin d'observer les données qui transitent, nous lançons ce programme en même temps que notre analyseur logique pendant 60 secondes. De cette façon, il sera possible de découper manuellement chaque séquence, matérialisée ici par un retour à la ligne. En effet, l'UART utilise un tampon qui ne nous permet pas de savoir s'il s'agit de données issues d'une même séquence de transmission, ou de séquences proches.

Ainsi, après une minute de capture, nous obtenons le résultat suivant :

```
DC 8A 00 0B 06 3D 05 17 DC 0A D5
DC 80 0A 0E 07 05 3D 17 DC 00 00 33 65 2E

DC 8A 00 0B 06 3D 05 17 DC 0A D5
DC 80 0A 0E 07 05 3D 17 DC 00 00 33 65 2E

DC 8A 00 0B 06 3D 05 17 DC 0A D5
DC 80 0A 0E 07 05 3D 17 DC 00 00 33 65 2E

DC 8A 00 0B 06 3D 05 17 DC 0A D5
DC 80 0A 0E 07 05 3D 17 DC 00 00 33 65 2E

DC 86 00 0E 02 3D 2D 02 15 04 F0 00 36 EE

DC 8A 00 0B 06 3D 05 17 DC 0A D5
DC 80 0A 0E 07 05 3D 17 DC 00 00 33 65 2E

DC 8A 00 0B 06 3D 05 17 DC 0A D5
DC 80 0A 0E 07 05 3D 17 DC 00 00 33 65 2E
```

Nous savons déjà que les 2 derniers octets correspondent au CRC du message. Il semble aussi que chaque message commence par **0xDC**.

Lors de la capture, nous constatons que les messages sont presque tous reçus par paire. Un protocole de communication intègre généralement les composantes suivantes :

- Taille du message (sauf si la taille est fixe, ce qui n'est clairement pas le cas ici) ;
- Adresse de la source ;
- Adresse de destination ;
- Contrôle d'intégrité.

Le contrôle d'intégrité correspond au CRC identifié précédemment. Penchons-nous sur la taille du message. Pendant notre capture, nous observons trois messages distincts :

- **DC 8A 00 0B 06 3D 05 17 DC 0A D5** : 11 octets (0x0B)
- **DC 80 0A 0E 07 05 3D 17 DC 00 00 33 65 2E** : 14 octets (0x0E)
- **DC 86 00 0E 02 3D 2D 02 15 04 F0 00 36 EE** : 14 octets (0x0E)

Sur ces 3 exemples, la longueur totale du message semble inscrite dans le quatrième octet. Il s'agit d'un élément extrêmement intéressant, puisque nous allons maintenant pouvoir identifier

Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)

chaque message en lisant le flux continu de données sur l'UART, sans devoir faire la corrélation avec notre analyseur logique. Le code Python suivant permet de récupérer distinctement les messages lus sur le bus.

```
 1: #!/usr/bin/python
 2: #! -*- coding=utf-8 -*-
 3:
 4: import serial
 5: import sys
 6:
 7: ser = serial.Serial('/dev/ttyAMA0', 4800, timeout=1, bytesize=8,
 8:                     stopbits=1, parity=serial.PARITY_ODD)
 9:
10: message = ""
11: messageStart = False
12: messageIndex = 0
13: messageSize = 0
14:
15: while True:
16:
17:     data = ser.read()
18:     datalen = len(data)
19:
20:     if datalen > 0:
21:
22:         for idx in range(datalen):
23:             byteVal = 255 - ord(data[idx]) # Inversion les bits
24:             message += "%0.2X " % (byteVal)
25:             messageIndex += 1
26:
27:         if not messageStart:
28:
29:             # Détection du début d'un message
30:             if byteVal == 0xDC:
31:                 messageStart = True
32:                 messageIndex = 0
33:
34:         else:
35:
36:             # Récupération de la taille du message sur
37:             # le quatrième octet (index=3)
38:             if messageIndex == 3:
39:                 messageSize = byteVal
40:
41:             # Lecture du dernier octet du message
42:             if messageIndex == messageSize - 1:
43:                 messageStart = False
44:                 messageIndex = 1
45:                 print message
46:                 message = ""
```

Continuons l'analyse en manipulant dans un premier temps les menus directement sur la pompe à chaleur, à partir de quelques appuis successifs sur le bouton **Info**, puis en manipulant la sonde intérieure. Le but ici est de générer des données sur le bus afin de les analyser.

Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)

Voici une capture réalisée pendant les manipulations de l'interface de la pompe à chaleur :

```
DC 8A 00 0B 06 3D 21 06 67 76 21
DC 80 0A 0E 07 21 3D 06 67 01 02 00 DC B9
DC 8A 00 0B 06 3D 21 05 18 AC 0A
DC 80 0A 0E 07 21 3D 05 18 01 00 00 C2 92
DC 8A 00 0B 06 3D 21 06 67 76 21
DC 80 0A 0E 07 21 3D 06 67 01 02 00 DC B9
DC 8A 00 0B 06 3D 21 05 18 AC 0A
DC 80 0A 0E 07 21 3D 05 18 01 00 00 C2 92
DC 8A 00 0B 06 3D 21 06 67 76 21
DC 80 0A 0E 07 21 3D 06 67 01 02 00 DC B9
```

Capture pendant les manipulations de la sonde intérieure :

```
DC 86 00 0B 06 3D 2D 05 8E 2B A3
DC 80 06 0E 07 2D 3D 05 8E 00 04 C0 B4 39
DC 86 00 0B 06 3D 2D 05 90 D8 5C
DC 80 06 0E 07 2D 3D 05 90 00 04 40 9C 4C
DC 86 00 0B 06 3D 2D 05 A5 BE AA
DC 80 06 0E 07 2D 3D 05 A5 00 07 00 11 77
```

Nous avions déjà constaté que les trames étaient capturées par séquence de deux messages. Cela ressemble fortement à un fonctionnement de type « question/réponse ». Nous allons avancer avec cette hypothèse :

- **Question de l'interface de la PAC :** `DC 8A 00 0B 06 3D 21 06 67 76 21`
- **Réponse :** `DC 80 0A 0E 07 21 3D 06 67 01 02 00 DC B9`
- **Question de la sonde intérieure :** `DC 86 00 0B 06 3D 2D 05 8E 2B A3`
- **Réponse :** `DC 80 06 0E 07 2D 3D 05 8E 00 04 C0 B4 39`

Il semblerait que les deuxièmes et troisièmes octets correspondent à l'adressage. Si l'on regarde de plus près ces deux octets :

- **Question de l'interface de la PAC :** `0x8A 0x00`
- **Réponse :** `0x80 0x0A`
- **Question de la sonde intérieure :** `0x86 0x00`
- **Réponse :** `0x80 0x06`

La première hypothèse est que l'adresse source est contenue dans le second octet du message. Le premier quartet serait `0x8` (valeur binaire : 1000) et l'adresse serait la valeur du second quartet. De la même manière, l'adresse de destination d'un message serait sur le troisième octet, avec le premier quartet qui vaudrait `0x0`, et l'adresse de destination sur le second quartet.

Cette hypothèse semblait cohérente jusqu'à ce que je capture les deux messages suivants :

```
DC 8A 7F 14 02 05 00 00 6C 00 75 03 05 07 10 34 24 00 2A 1F
DC 80 7F 17 02 05 00 02 19 03 45 00 00 FF FF FF FF FF FF 00 0F 7D C1
```

Ces messages sont isolés, et ne semblent pas correspondre au schéma « question/réponse ».

La valeur du second octet semble indiquer que la source du premier message est l'interface de la PAC. Le troisième octet, `0x7F`, est particulier. Son écriture binaire est `0111 1111`, ce qui ressemble étrangement à une adresse de broadcast **[1]** si on se focalise sur les 7 bits de poids faible.

Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)

L'hypothèse la plus probable est donc que les adresses soient codées sur 7 bits, et non pas sur 4.

Voici les 3 adresses identifiées sur le bus :

- **0x0A** : IHM de la pompe à chaleur ;
- **0x06** : sonde intérieure ;
- **0x00** : système central, qui semble être le cœur du système.

Maintenant que nous avons identifié le rôle des 4 premiers octets, focalisons-nous sur la suite des messages.

D'après les captures réalisées, l'interface de la pompe à chaleur semble questionner en permanence le système sur la valeur à afficher sur l'écran toutes les 10 secondes environ. Par exemple, sur l'écran principal, l'état de la PAC est affiché. Nous observons toutes les 10 secondes les messages suivants :

```
DC 8A 00 0B 06 3D 05 17 DC 0A D5
DC 80 0A 0E 07 05 3D 17 DC 00 00 33 65 2E
```

En affichant l'écran suivant, qui correspond à l'état du ballon d'eau chaude, les messages observés sont :

```
DC 8A 00 0B 06 3D 05 17 CE 38 A6
DC 80 0A 0E 07 05 3D 17 CE 00 00 4B 6C 7E
```

Le cinquième octet semble donc être le type de message. Nous pouvons dire que **0x06** semble identifier une requête sur une valeur ou un état, et **0x07** une réponse avec la valeur attendue.

La suite du message correspond probablement à l'élément sur lequel porte la requête :

- **3D 05 17 DC** : Quel est l'état de la PAC ?
- **05 3D 17 DC 00 00 33** : L'état de la PAC est « pas de demande ».

La requête est codée sur 4 octets. La réponse reprend ces 4 octets, puis on observe 3 octets qui semblent correspondre à la valeur du paramètre retourné (**0x000033** = Pas de demande). On peut néanmoins constater une subtilité, les deux premiers octets de l'identifiant de la requête sont inversés entre la requête et la réponse. La distinction entre les deux types de messages est déjà codée sur le cinquième octet (**0x06** ou **0x07**). Je n'ai pas trouvé d'explication logique pour expliquer le rôle donné à l'inversion de ces deux octets.

Nous venons d'analyser la transmission d'un état, essayons maintenant d'observer la récupération d'une valeur de température. Après avoir fait défiler les écrans de la PAC jusqu'à l'affichage de la température extérieure, voici les messages capturés :

```
DC 8A 00 0B 06 3D 05 05 21 51 76
DC 80 0A 0E 07 05 3D 05 21 00 03 06 B0 99
```

Sur le même principe que l'analyse précédente, on peut en déduire le sens des messages :

- **3D 05 05 21** : Quelle est la température extérieure ?
- **05 3D 05 21 00 03 06** : La température extérieure est de 12,1°C.

La valeur de température est donc codée avec **0x000306** = 774 en valeur décimale. En début d'article, nous avions utilisé les variations de température de consigne pour déduire le codage des messages. Nous avions identifié que la valeur décimale correspondait à 1/64°C, nous retrouvons bien 774/64=12,1 arrondi au dixième de degré.

Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)

Maintenant que nous avons trouvé le moyen de décrire l'état du système, nous pouvons nous intéresser aux commandes afin de comprendre ce qui se passe lorsque l'on allume, éteint, force le régime réduit, etc.

La sonde intérieure dispose d'un bouton permettant de basculer sur chacun de ces modes. Pour commencer, nous nous positionnons sur le mode arrêt. Nous capturons les trois messages suivants :

- Message 1 : `DC 86 00 0D 03 3D 2D 05 74 01 00 B1 5B`
  → Source : `0x06` *(sonde ambiance)*
  → Destination : `0x00` *(système central)*
  → Signification : **commande arrêt ?**
- Message 2 : `DC 80 06 0B 04 2D 3D 05 74 58 5F`
  → Source : `0x00` *(système central)*
  → Destination : `0x06` *(sonde ambiance)*
  → Signification : **acquittement de la commande d'arrêt ?**
- Message 3 : `DC 80 7F 15 02 2D 00 02 11 00 00 00 90 FF FF FF FF 00 01 C8 CF`
  → Source : `0x00` *(système central)*
  → Destination : `0x7F` *(broadcast)*
  → Signification : **diffusion du nouvel état du système ?**

Nous venons donc de mettre en évidence un nouveau type de message. Le cinquième octet `0x03` pourrait indiquer qu'il s'agit d'une commande pour modifier un état, et la valeur `0x04` serait l'acquittement d'une commande. En analysant les deux premiers messages, on peut déduire le format de la commande.

À partir du premier message :

- `3D 2D 05 74` : action sur le mode de fonctionnement du chauffage, codé sur 4 octets ;
- `01 00` : valeur de l'état : « arrêt », codé sur 2 octets.

À partir du second message :

- `2D 3D 05 74` : acquittement du mode de fonctionnement demandé.

Comme pour les requêtes sur les valeurs du système, les deux premiers octets qui décrivent le message sont inversés entre la commande et l'acquittement (ici `0x3D` et `0x2D`).

Le message diffusé en broadcast, c'est-à-dire à l'ensemble des éléments connectés sur le bus, contient l'état du système. C'est vraisemblablement ce message qui va entraîner la modification des écrans affichant le mode de fonctionnement du chauffage (à l'arrêt, régime confort, régime réduit...).

En reprenant notre méthodologie, nous pouvons déduire le codage de l'ensemble des messages et de leurs valeurs. Voici quelques exemples :

Type de message (5ème octet) :

- `0x03` : Commande sur la PAC
- `0x04` : Acquittement de la commande (précédé par un message de type 0x03)
- `0x06` : Requête sur un paramètre
- `0x07` : Réponse contenant la valeur d'un paramètre (précédée par un message de type 0x06)

Commandes sur la PAC :

- Mode de fonctionnement du chauffage : `3D 2D 05 74`

Valeurs possibles :

- `0x0100` : à l'arrêt
- `0x0101` : automatique
- `0x0102` : réduit
- `0x0103` : confort
- Mode de fonctionnement du ballon d'eau chaude : `3D 31 05 73`

Valeurs possibles :

- `0x0100` : charge forcée

Paramètres de la PAC :

- État PAC : `3D 05 17 DC`

Valeurs possibles :

- `0x000033` : Pas de demande
- `0x00002E` : Compresseur en fonction

Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)

• **0x000011** : Temporisation arrêt actif
• **0x00007D** : Dégivrage actif
• **0x00002C** : Compresseur et résistance électrique en marche
• État ECS : **3D 05 17 CE**

Valeurs possibles :

• **0x00004B** : Chargé
• **0x000045** : Charge active
• **0x00005E** : Charge accélérée active

Température extérieure : **3D 05 05 21**

Température d'ambiance : **3D 2D 05 1E**

Consigne de chauffage : **3D 2D 05 8E**

Consigne d'ambiance : **3D 2D 05 93**

Toutes les valeurs de températures sont des entiers signés, codés sur 2 octets, et correspondent à 1/64ème degré Celsius. La valeur est précédée d'un octet nul. Il s'agit d'un *padding* pour arriver à 3 octets.

2.3 Transmission sur le BUS

Nos recherches permettent maintenant de lire correctement les données qui transitent sur le bus. La récupération des paramètres ne fonctionne que si un des éléments connectés au bus émet une requête pour en connaître sa valeur, ce qui n'est pas fiable, puisque cela dépend par exemple de la page affichée sur l'écran de la PAC.

Il faudrait donc être capable d'émettre ses propres requêtes. Nous avons jusqu'à présent identifié 4 adresses : le système central **0x00**, l'écran de configuration **0x0A**, la sonde intérieure **0x06** et l'adresse de diffusion **0x7F**.

Le code suivant permet d'envoyer une requête sur le bus :

```
 1: #!/usr/bin/python
 2: #! -*- coding=utf-8 -*-
 3:
 4: import serial
 5: import sys
 6: import crcmod
 7:
 8: ser = serial.Serial('/dev/ttyAMA0', 4800, timeout=1, bytesize=8,
 9:                     stopbits=1, parity=serial.PARITY_ODD)
10:
11: # Température intérieure ?
12: msgData = "\x06\x3D\x05\x05\x21"
13:
14: srcAddr = 0x07 # Adresse source
15: dstAddr = 0x00
16:
17: lenMsg = len(msgData) + 6
18:
```

Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)

```
19: msg = "\xDC" # Préfixe de chaque message
20: msg += chr(srcAddr | 0x80 ) # Adresse source avec le premier bit à 1
21: msg += chr(dstAddr) # Adresse de destination
22: msg += chr(lenMsg) # Longueur du message
23: msg += msgData # Contenu du message
24:
25: # Calcul et ajout du CRC
26: crc16 = crcmod.predefined.Crc('xmodem')
27: crc16.update(msg)
28: # Récupération de la valeur du CRC sur 16 bits, découpé en 2 octets distincts
29: msg += chr((crc16.crcValue & 0xFF00) >> 8) + chr(crc16.crcValue & 0xFF)
30:
31: msgToSend = ""
32: for idx in range(len(msg)):
33:     # Inversion des bits avant la transmission sur le bus
34:     msgToSend += chr(255-ord(msg[idx]))
35:
36: ser.write(msgToSend)
```

Arbitrairement, nous allons affecter à notre montage l'adresse **0x07** (ligne 14). Cette valeur ne semble pas être utilisée. La bibliothèque **crcmod** permet de calculer le CRC. La valeur de 16 bits obtenue est scindée en 2 octets distincts grâce à une manipulation binaire (ligne 29).

La variable **msgData** est définie en ligne 12 avec la valeur précédemment relevée qui correspond à une requête sur la température extérieure.

En lançant en parallèle le code qui permet de lire les données, nous récupérons bien la requête, ainsi que la réponse avec la valeur de la température extérieure :

- **DC 87 00 0B 06 3D 05 05 21 07 12** : Quelle est la température extérieure ?
- **DC 80 07 0E 07 05 3D 05 21 00 02 FA 25 00** : La température extérieure est 11,9°C : (0x0002FA = 762, donc 762/64 = 11,9)

En adaptant le code, il est désormais tout à fait possible de modifier le mode de chauffage. Ainsi le programme suivant mettra le circuit de chauffage à l'arrêt :

```
 1: #!/usr/bin/python
 2: #! -*- coding=utf-8 -*-
 3:
 4: import serial
 5: import sys
 6: import crcmod
 7:
 8: ser = serial.Serial('/dev/ttyAMA0', 4800, timeout=1, bytesize=8,
 9:                     stopbits=1, parity=serial.PARITY_ODD)
10:
11: # Calcul et ajout du CRC
12: def processCRC(msg):
13:
14:     crc16 = crcmod.predefined.Crc('xmodem')
15:     crc16.update(msg)
16:
```

Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)

```
17:     # Les 16 bits du CRC
18:     return chr((crc16.crcValue & 0xFF00) >> 8) + chr(crc16.crcValue & 0xFF)
19:
20: srcAddr = 0x07
21: dstAddr = 0x00
22:
23: # Mode du chauffage
24: msgData = "\x03\x3D\x2D\x05\x74\x01\x03"
25: # Mode arrêt
26: msgData += "\x01\x00"
27:
28: lenMsg = len(msgData) + 6 # 6 octets d'en tête + crc
29:
30: msg = "\xDC" # Préfixe de chaque message
31: msg += chr(srcAddr | 0x80 ) # Adresse source avec le premier bit à 1
32: msg += chr(dstAddr) # Adresse de destination
33: msg += chr(lenMsg) # Longueur du message
34: msg += msgData # Contenu du message
35: msg += processCRC(msg)
36:
37: # Inversion des bits avant la transmission sur le bus
38: msgToSend = ""
39: for idx in range(len(msg)):
40:     msgToSend += chr(255-ord(msg[idx]))
41:
42: ser.write(msgToSend)
```

Il faudrait améliorer le code pour vérifier que l'on reçoit bien l'acquittement pour valider le nouvel état. De la même façon, il est tout à fait possible de commander la production d'eau chaude, ou de modifier le paramétrage de certaines valeurs comme la consigne du chauffage.

3. TABLEAU DE BORD

Pour terminer notre cheminement, nous allons nous pencher sur la génération de graphes pour suivre quelques-uns des indicateurs que nous avons identifiés.

Pendant longtemps, pour générer des graphes qui représentent des valeurs dans le temps, la référence a été d'utiliser la suite **RRDtools** qui permet notamment de générer des graphes sous forme de fichiers image (PNG ou JPEG). Désormais, la mode est d'utiliser des interfaces comme **Grafana**, beaucoup plus flexibles que RRD puisque la génération de graphes se fait directement sur le navigateur.

Le stockage des données se fait en utilisant une base de données orientée « Time Series » **[2]**, comme OpenTSDB, Graphite ou InfluxDB. La figure 5, page suivante, présente le genre de tableau de bord qu'il est possible de mettre en place.

Sur la première partie du graphe, on peut distinguer un cycle de production d'eau chaude, avec la température de départ de la PAC qui dépasse les 55°C. Ensuite, un peu avant 4 heures du matin, la température est passée sous les 19°C, qui est la valeur de consigne, et a déclenché un cycle de chauffage, jusqu'à ce que la température d'ambiance atteigne les 19,5°C (consigne + 0,5°C).

Je n'aborderai pas en détail la création de dashboard Grafana. Cet aspect pourrait à lui seul faire l'objet d'un article spécifique. Il existe beaucoup de documentation sur Internet à ce sujet.

Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)

CONNECT ÉVOLUE !

LISEZ CE NUMÉRO ET PLUS DE 20 AUTRES EN LIGNE !

ACTUELLEMENT SUR CONNECT :

CE NUMÉRO
et + de 15 autres numéros de Hackable

+

2 numéro Hors-Série de Hackable

TOUT CELA À PARTIR DE 149 € TTC*/AN !

* Tarif France Métropolitaine

OFFRE DÉCOUVERTE CONNECT

1 MOIS GRATUIT, RÉSERVÉE AUX PROFESSIONNELS

Appelez le 03 67 10 00 28 et donnez le code « HK19 » pour découvrir Connect gratuitement pendant 1 mois !

Pour tous renseignements complémentaires, contactez-nous via notre site internet : **www.ed-diamond.com**, par téléphone : **03 67 10 00 28** ou envoyez-nous un mail à **connect@ed-diamond.com** !

Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)

Figure 5 : Exemple de graphe obtenu avec Grafana.

CONCLUSION

Cet article balaie les principales étapes incontournables lorsque l'on souhaite s'aventurer dans l'exploration d'un système fermé. Heureusement pour nous, le protocole utilisé par les composants installés dans ma pompe à chaleur reste accessible pour peu que l'on prenne les bonnes directions.

Maintenant qu'il est possible de se connecter au bus de communication, les possibilités sont tout de suite très nombreuses. Par exemple, il devient envisageable de construire sa propre sonde d'ambiance pour remplacer celle déjà installée. Pourquoi ne pas utiliser un ensemble de capteurs de température répartis dans l'habitation, et choisir la température d'ambiance annoncée à la PAC (température moyenne de l'habitation, température minimale, etc.) au lieu de se contenter d'une seule mesure, bien souvent captée au beau milieu du salon ?

Il devient même possible d'aller plus loin, en assurant une partie du travail de l'automate. Imaginons que l'on démarre la production d'eau chaude en ne se basant pas uniquement sur un seuil de température et un créneau horaire (probablement réglé pour fonctionner sur le tarif réduit de votre fournisseur d'électricité), mais en se basant sur les habitudes, les prévisions météorologiques, dans le but de ne pas se retrouver avec de l'eau froide, tout en réduisant au maximum les coûts en électricité.

De la même façon, le circuit de chauffage doit pouvoir être amélioré. Selon l'exposition et l'isolation de votre habitation, il n'est peut-être pas utile de démarrer un plancher chauffant en fin de nuit si on sait que le ciel sera dégagé dès le lever du soleil, et que son rayonnement produira assez de chaleur pour retrouver le seuil minimal que l'on s'est fixé. Ce genre de configuration n'est possible que si nous avons le contrôle complet du système de chauffage. **EL**

RÉFÉRENCES

[1] https://fr.wikipedia.org/wiki/Broadcast_(informatique)

[2] https://en.wikipedia.org/wiki/Time_series_database

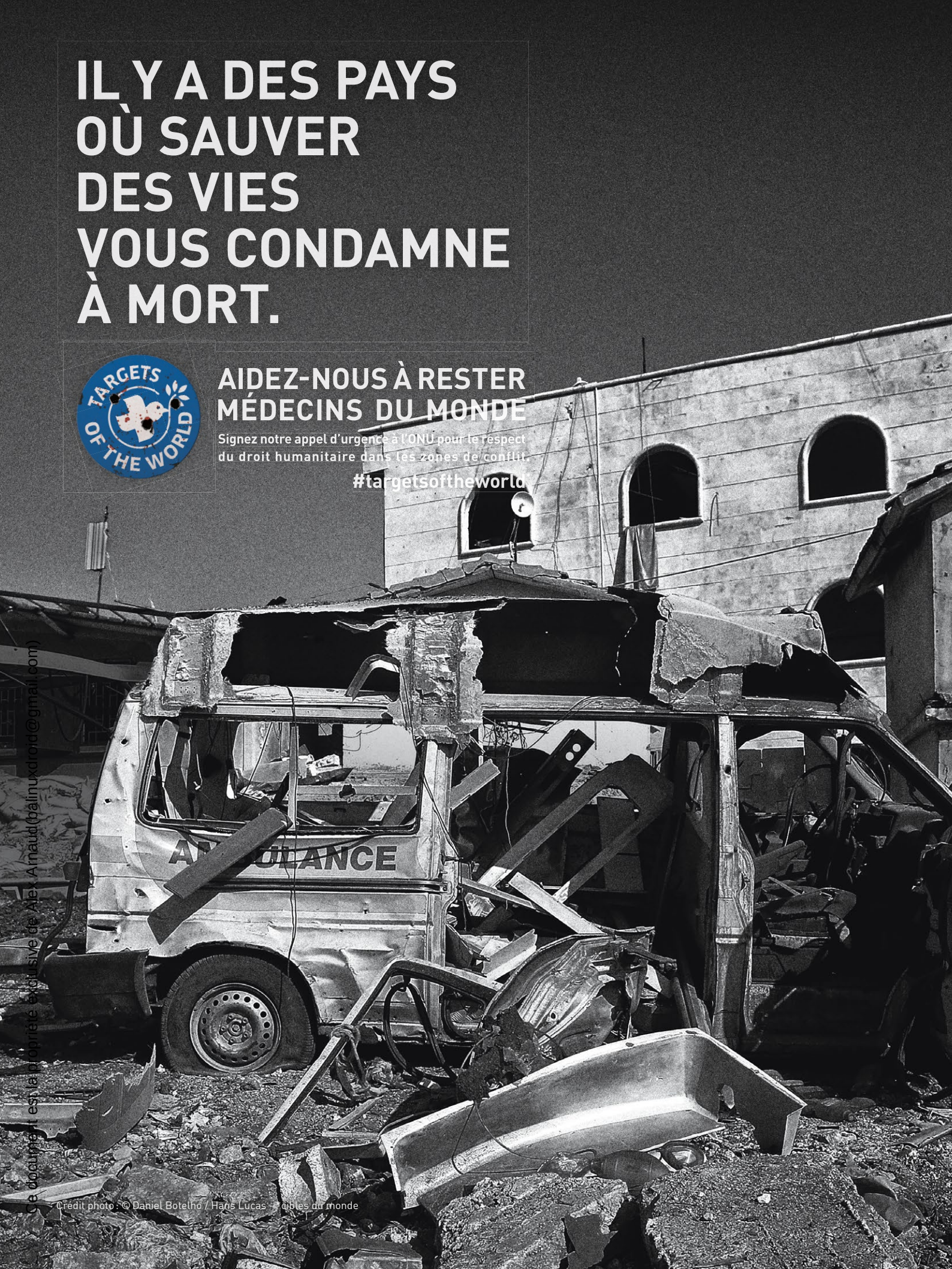
Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)

IL Y A DES PAYS OÙ SAUVER DES VIES VOUS CONDAMNE À MORT.

AIDEZ-NOUS À RESTER MÉDECINS DU MONDE

Signez notre appel d'urgence à l'ONU pour le respect du droit humanitaire dans les zones de conflit.

#targetsoftheworld



Ce document est la propriété exclusive de Alex Arnaud (alinuxdrod@gmail.com)

Crédit photo : © Daniel Botelho / Hans Lucas – cibles du monde

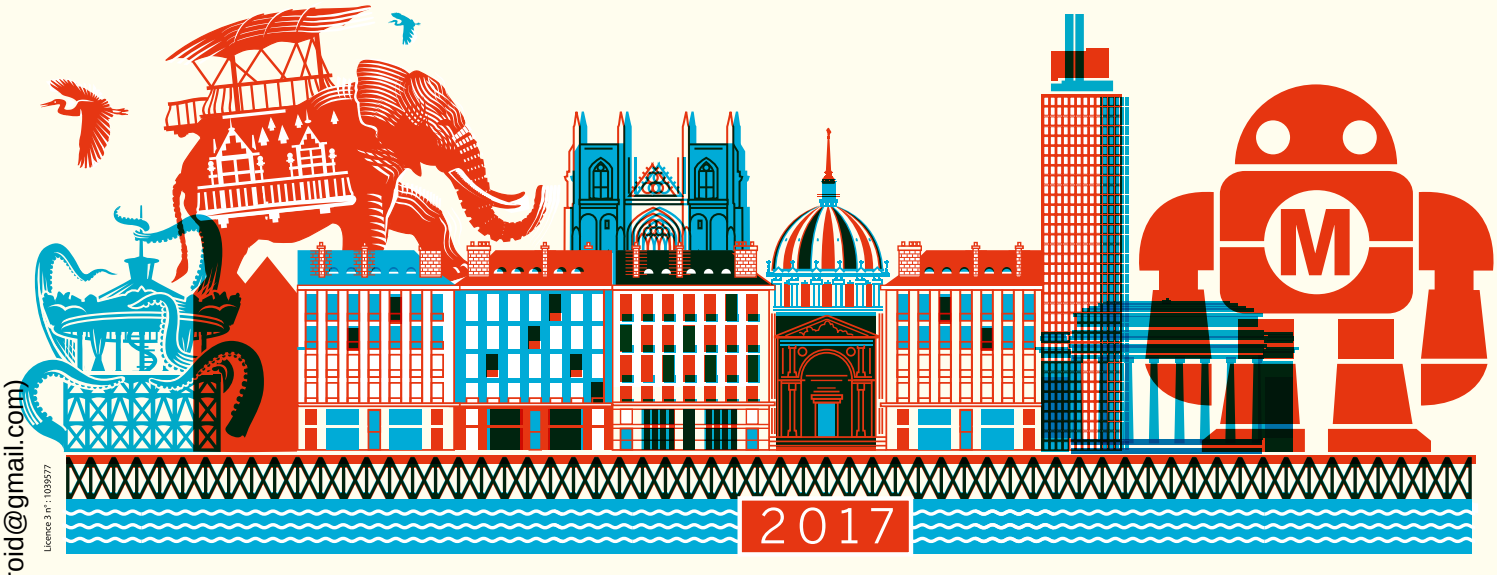
Maker Faire® Nantes

Présenté par Leroy Merlin

BIENVENUE DANS LE FUTUR !

7-8-9 JUILLET

Les Machines de l'île de Nantes



PROGRAMME
nantes.makerfaire.com

BILLETTERIE
lesmachines-nantes.fr

400 Makers ★ 50 Ateliers ★ 20 Animations

Bricoleurs / **Artistes** / Hackeurs / **Gamers**
Youtubers / Chercheurs / **Performeurs**
Créateurs / **Entrepreneurs** / Rêveurs...

Présenté par LEROY MERLIN

Make: makezine.com · Nantes Métropole · Région Pays de la Loire · Le voyage à Nantes · AIRFRANCE · CCI Nantes St-Nazaire · ulule · #MFN17

tan · APPART'CITY · télénantes loire-atlantique · Presse Océan · ouest france · UGC · France bleu loire océan

Licence 3 n° 1039577

Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)

Ce document est la propriété exclusive de Alex Arnaud(balinuxdroid@gmail.com)