# balise GPS **LoRa**

## Matériel et logiciel libres et ouverts


Quel bel instrument qu'une balise GPS ! Elle donne sa position à intervalles réguliers dont vous réglez la durée à votre guise ! Avec ce projet proposé par Elektor, la position est communiquée par Node-RED, en passant par le réseau LoRa, à un PC, un RPi, un téléphone portable, une tablette ou un autre appareil. Celui-ci affiche la position instantanée de l'instrument sur une carte ouverte dans un navigateur. Magellan aurait apprécié.

### INFOS SUR LE PROJET

**Mots clés**
LoRa, RFM95, STM32, Raspberry Pi, Arduino, GPS, enseignement, OpenHardware, OpenStreetMap

**Niveau**
débutant – connaisseur – expert

**Temps**
1 h environ pour l'assemblage complet (coffret et antenne compris)

**Outils**
fer à souder, tournevis, perceuse

**Coût**
90 à 100 € environ

Beaucoup d'entre nous aiment savoir si un objet se trouve toujours à l'endroit où on l'a laissé : la voiture par exemple, des engins de chantier, ou un animal domestique. On s'est donc mis à équiper ces objets (pardon pour les animaux…) de balises GSM-GPS telles que ElekTrack [1]. Leur propriétaire peut ainsi s'assurer de leur position ou être prévenu si celle-ci changeait. Pour cela il fallait un module de téléphone mobile et une couverture réseau pas forcément disponible. Un tel suivi n'est pas gratuit. Les redevances dépassent quelquefois le raisonnable, comme certains chercheurs russes en ont fait l'amère expérience [2]. Comme alternative, il y a le LoRaWAN, un réseau dans la bande ISM des 868 MHz, sans licence, et dont le point fort est une grande portée pour une faible puissance d'émission.

L'un de ces réseaux est *The Things Network*. Les passerelles, c'est-à-dire les points d'accès, sont gérées par une communauté à laquelle chacun peut se joindre avec sa propre passerelle et contribuer ainsi à étendre le réseau. Le transport de données dans ce réseau est gratuit, l'idéal pour entrer dans le monde des capteurs en réseau. Si, dans votre environnement, il ne se trouve pas de passerelle, vous pouvez installer la vôtre à peu de frais.

### Nœud LoRa

La liste des ingrédients pour une balise est assez courte. Comme pour la bonne cuisine, tout est dans l'art de bien les accommoder. Dans ce cas, il aura fallu quelques semaines de sang et de larmes. Au fil de la conception, nous sommes passés de la simple

balise LoRa-GPS initiale au nœud universel LoRa-Elektor (**fig. 1**) aux utilisations multiples, dont la balise n'en est qu'une parmi d'autres. Le nœud LoRa-Elektor et tous ses avatars passés, présents et futurs sont libres et ouverts aussi bien pour le matériel que pour le logiciel.

Comme le nœud LoRa-Elektor a été décrit en détail dans le numéro de mars/avril [3], de même que l'accès à *The Things Network* [4], il vaudrait mieux avoir ce numéro à portée de main. Dans la description du matériel (**fig. 2**), nous pourrons ainsi nous concentrer sur l'essentiel et utiliser la place disponible pour nous occuper du module GPS, du logiciel GPS, de l'intégration dans *The Things Network*, du transport de données à travers

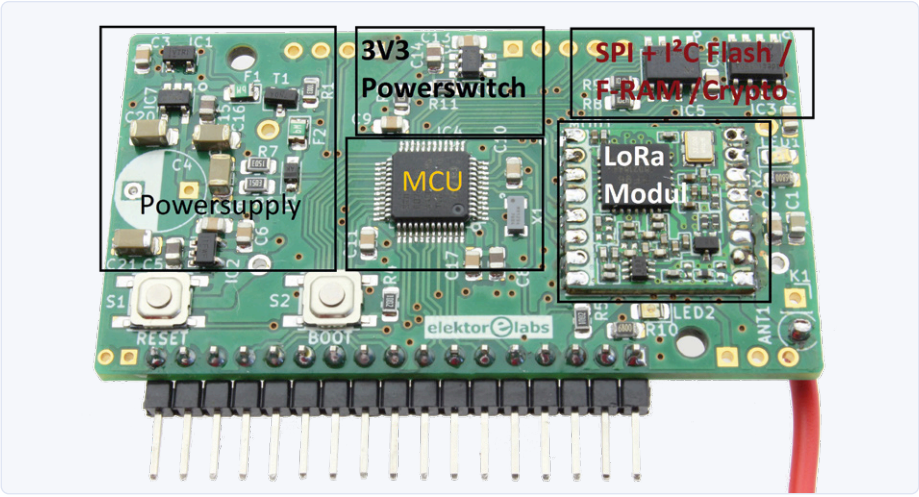



*Figure. 1. Le circuit équipé du nœud LoRa Elektor*

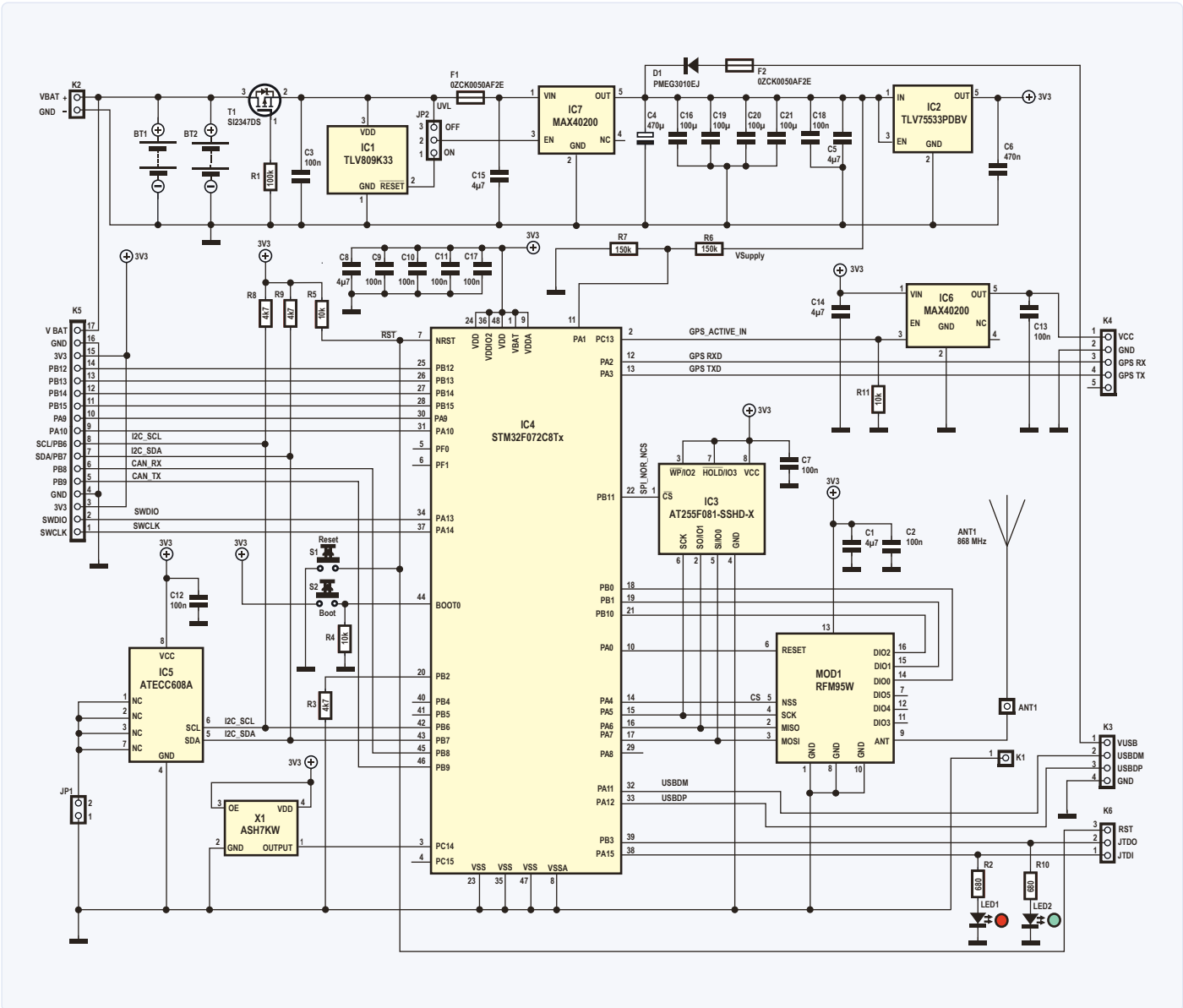



*Figure. 2. Schéma du nœud LoRa.*

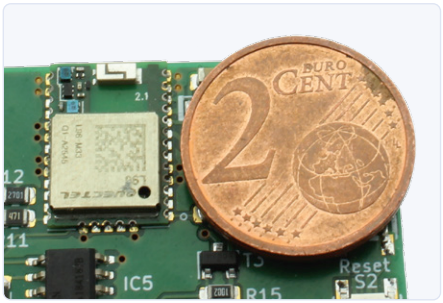



*Figure. 3. Le minuscule module GPS L96.*

ce réseau, ainsi que de MQTT, de Node-RED et de l'affichage sur une carte géographique.

## Coffret

Pour le coffret, qui doit contenir la batterie, la carte mère et un module GPS, nous avons retenu un modèle de la série 1551 de Hammond, avec l'indice de protection IP54 (étanche à la poussière et aux projections d'eau), propre à l'usage en extérieur.

### Microcontrôleur

Le microcontrôleur utilisé est un STM32F072C8T avec 64 Ko de mémoire flash et 16 Ko de mémoire RAM. La famille des STM32 comporte beaucoup de puces en boîtier LQFP48 aux broches compatibles, permettant d'utiliser avec la même carte un microcontrôleur avec davantage de mémoire flash ou à plus faible consommation. Le développement du micrologiciel peut se faire classiquement en C/C++ ou avec l'EDI Arduino (grâce au projet STM32duino). Grâce au chargeur (non réinscriptible) de la puce, il suffit d'un simple convertisseur USB-série pour télécharger le micrologiciel.

## Le transmetteur LoRa

Sur la carte se trouve le transmetteur LoRa, un RFM95W, qui travaille dans la bande ISM 868 MHz.

## Alimentation

L'alimentation est fournie par une batterie Li-ion bon marché, rechargeable, utilisée sur les drones. Des cellules AAA au lithium pourraient convenir mais exigeraient un coffret plus grand.

Nous avons délibérément renoncé à un circuit de charge, la batterie doit donc être rechargée par un chargeur externe approprié. Nous n'avons pas renoncé à une détection de tension basse pour protéger la batterie d'une décharge profonde ni à une protection contre une inversion de courant intempestive, sous la forme d'une «diode idéale» IC7, pour que la batterie ne risque pas d'être surchargée, de fuir, d'exploser ou de prendre feu. Le circuit de protection est protégé directement au niveau de la batterie par un fusible de 500 mA pour éviter une destruction des pistes du PCB en cas de court-circuit. Ce circuit a l'avantage de permettre, moyennant une diode de plus, l'alimentation directe sous 5 V, avec la batterie en secours.

## Mémoire flash et module de cryptage I²C/SPI

La carte peut recevoir en option une mémoire flash (IC3) et un module de cryptage (IC5) connectés par I²C/SPI au contrôleur. Ces deux puces ne sont pas nécessaires pour la balise GPS, on peut donc les ignorer.

## Régulateur 3,3 V

Le régulateur 3,3 V IC2 est assez puissant pour alimenter à la fois le nœud LoRa et le module GPS.

## Module GPS

Il nous a fallu un peu de temps pour choisir et mettre en œuvre le module GPS. Au départ, nous avions prévu le Quectel L96, un module minuscule (**fig. 3**), à monter directement sur la carte mère. Dans une note d'application du L96, il était question d'un plan de masse approprié sur lequel le module devait être correctement positionné. Dans la première version du circuit imprimé, cette note avait été «oubliée», mais même dans la deuxième version, où elle avait été scrupuleusement respectée, la réception GPS par la minuscule antenne du module (le composant au-dessus de l'inscription 2.1) était si médiocre qu'il fallait un temps considérable pour obtenir la localisation, avec un effet néfaste sur la consommation d'énergie du module.

C'est ainsi que nous en sommes arrivés à la version définitive du circuit, dans laquelle le récepteur GPS n'est plus intégré à la carte mère, mais est un module séparé placé au-dessus. Pour ce module GPS, nous avons finalement choisi le GT-7U 1728 d'Open-Smart (**fig. 4**), l'un de ces composants chinois étonnamment bon marché et disponible dans notre boutique en ligne [3]. Ce mode de montage facilite l'emploi du module. Pour améliorer la réception, il est possible d'y connecter une antenne externe.

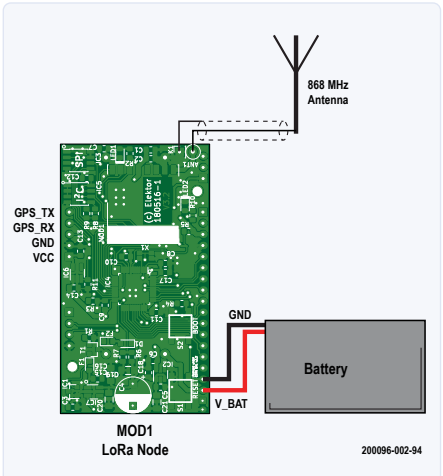



*Figure. 4. Le module GPS installé sur le nœud LoRa.*

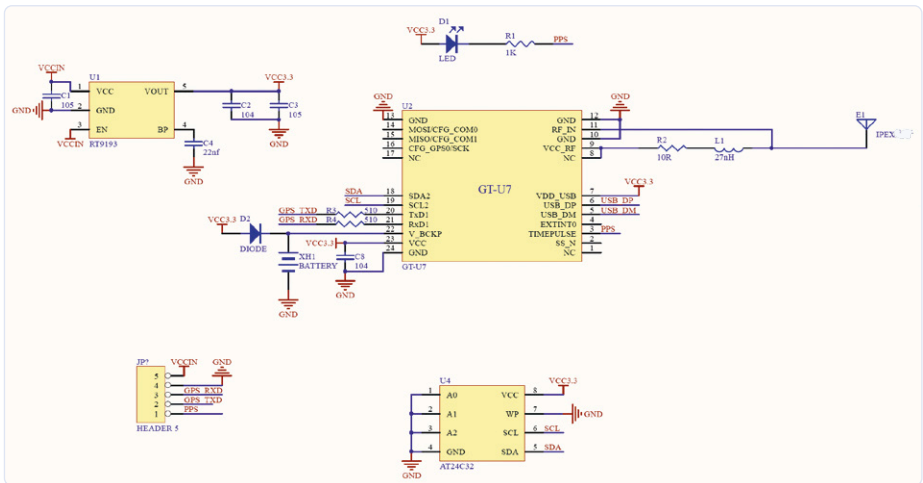



*Figure. 5. Schéma interne du module GPS.*

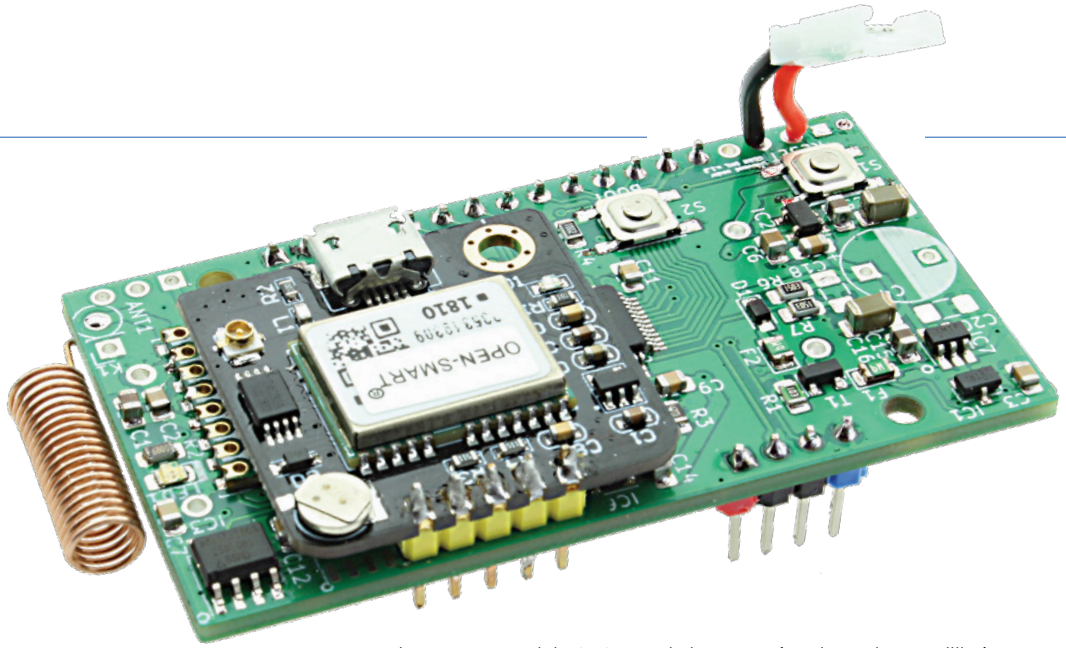



*Figure. 6. Le module GPS survole la carte mère de quelques millimètres.*

De plus, la carte mère peut ainsi se limiter aux fonctions nécessaires à l'ensemble des applications prévues.

Dans notre boutique en ligne, vous trouverez quelques données de base du module GPS et, caché sous *For more information click here*, un lien vers une archive avec des pilotes, un manuel utilisateur, des programmes pour le µC et l'Arduino, des outils de test et un schéma du circuit (**fig. 5**), qui montre que quand le module fonctionne avec le port USB, son alimentation passe aussi par ce port. La ligne VCC de l'USB est connectée directement à la broche VCC du module (qui peut supporter jusqu'à 5 V). Sans précaution spéciale, on pourrait donc avoir une connexion directe du 5 V de l'USB avec l'alimentation 3,3 V de la carte mère, ce qui endommagerait la balise en cas de connexion directe à un port USB 5 V. Même si l'USB respectait la plage de tension permise (3,0 à 3,6 V), il y aurait un conflit si le module GPS était alimenté par le nœud LoRa. Une seconde «diode idéale» IC6 permet de l'éviter. Si la tension à la sortie du MAX40200 dépasse la tension à son entrée, la «diode idéale» bloque, mais si le module est alimenté avec les 3,3 V de la carte mère, la faible tension de chute directe de cette diode (85 mV seulement) assurera au module une tension d'alimentation correcte. De plus, le µC a la possibilité, par la broche *Enable*, de couper l'alimentation du module. Dans notre application, le mieux serait d'oublier complètement ce port USB du module GPS !

Pour faire entrer le module GPS dans le coffret, il a été embroché par-dessus la carte mère (**fig. 6**) et survole de quelques millimètres les composants du nœud LoRa. C'est ainsi que la batterie, la carte mère et le module GPS se partagent au mieux la place dans le coffret. Outre l'antenne GPS, couchée dans le coffret, il faut aussi tenir compte de l'antenne LoRa. Une antenne hélicoïdale miniature serait assez petite pour entrer dans le coffret, mais cela réduirait la puissance émise de 30 dBm, ce qui rendrait l'émission d'un signal vers une passerelle LoRaWAN presque impossible. C'est pourquoi une antenne extérieure est préférable, qu'il s'agisse d'un simple fil ou d'un dispositif esthétique et étanche [2].

## Micrologiciel

Le micrologiciel se compose essentiellement de quatre parties (**fig. 7**), un décodeur pour les données GPS, la bibliothèque Arduino LMIC comme pile LoRaWAN, une bibliothèque pour la basse consommation et une interface utilisateur pour le paramétrage. Le décodage des données GPS est effectué par la bibliothèque TinyGPS++, que nos lecteurs connaissent depuis son usage dans le serveur de temps ESP32-NTP [5]. Pour que la bibliothèque puisse effectuer son travail, les données entrantes doivent être lues sur l'interface série et passées à la bibliothèque pour décodage. Pour éviter une collision entre le module GPS et l'interface série utilisée pour le téléchargement du micrologiciel, on utilise le deuxième UART qu'on initialise dans le croquis avec **HardwareSerial Serial X(USART2) ;**. Dans la description de la carte, on a déjà défini les broches utilisées pour la commande du module GPS. Le µC doit alimenter la bibliothèque avec les données entrantes qu'il prélève dans le tampon de l'UART. Pour s'apercevoir que le module GPS a relevé une nouvelle position, la bibliothèque lit périodiquement la position courante et traite la réponse.

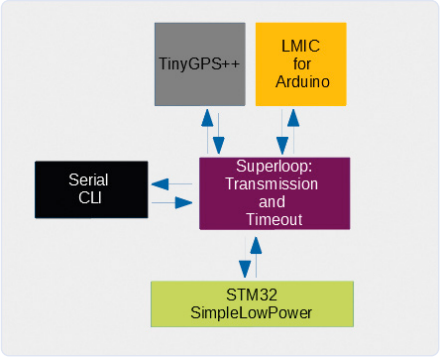


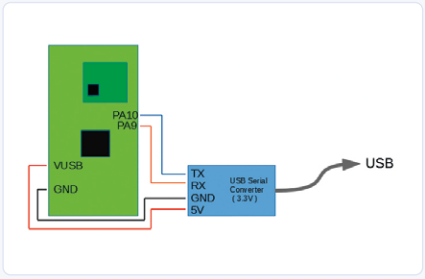

*Figure. 7. Structure du micrologiciel.*

Pour la communication LoRaWAN, on utilise la même bibliothèque LMIC qui a servi de pile LoRaWAN dans le LoRaSwitch et un grand nombre d'autres projets Arduino-LoRaWAN. En utilisant cette bibliothèque, on doit veiller à ce que la fonction **os_runloop_once** soit appelée aussi souvent que possible et qu'il se produise le moins possible d'interruptions de longue durée. C'est aussi l'une des raisons pour lesquelles il est si difficile de faire tourner la LMIC sur un ESP32 et de respecter les contraintes de temps.

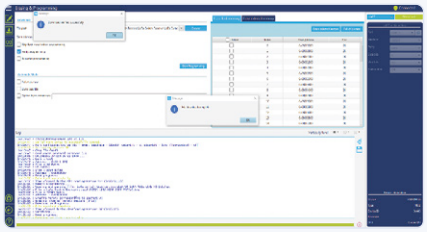
Pour le transport de données, on passe une charge utile à la bibliothèque et on attend la confirmation que les données ont bien été envoyées. Dans la plupart des croquis, on définit la configuration de la bibliothèque LoRaWAN et la plupart de ses données d'accès à la compilation ; ici elle peut être modifiée en cours d'exécution. Cette configuration est effectuée au moyen d'une ligne de commande sérielle qui peut être saisie dans la fenêtre terminal de l'EDI Arduino (**fig. 10**). Les éléments de base de cette ligne

## TÉLÉCHARGEMENT DU MICROLOGICIEL

Tout ce matériel ne fait pas une balise LoRa. Pour cela, il faut encore «un peu» de logiciel, disponible sur la page web du projet [9] sous forme de fichier hexadécimal. Pour télécharger le logiciel, il suffit d'un convertisseur USB-série [voir l'encadré produits] et quatre fils Dupont (femelle-femelle).

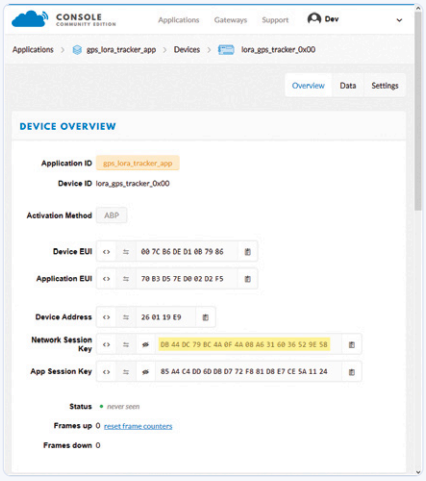



Le fichier hexadécimal peut être programmé avec le programmateur STM32-Cube. Pour cela, il faut appuyer simultanément pendant un petit moment sur les boutons *Boot* et *Reset* pour que la puce passe dans le mode *bootloader*. On peut alors choisir la bonne interface série et établir la connexion en appuyant sur *Connect*. Si aucun message d'erreur n'apparaît, la programmation du micrologiciel peut commencer.



Une fois la programmation réussie, on sort la puce du mode *bootloader* en appuyant sur *Reset*. Elle commence alors immédiatement à exécuter le code.

Si l'on ne désire pas programmer la puce avec le micrologiciel tel quel, on peut l'adapter à ses propres besoins en utilisant l'EDI Arduino, le recompiler et le télécharger. L'EDI Arduino supporte la carte LoRa Elektor avec la version 1.8.0 du projet STM32duino, sans nécessiter de modification d'un fichier quelconque. Mais attention : la version courante 1.8.0 installable au moyen du gestionnaire de cartes Arduino comporte quelques bogues qui peuvent provoquer le plantage du micrologiciel. Dans la version courante disponible sur Github du projet STM32duino, ces bogues sont corrigés.




de commande ont été repris du projet de la station de soudage à température contrôlée [6], où elle a été utilisée pour le réglage de quelques paramètres. Pour la balise LoRa-GPS, on a ajouté quelques fonctions telles que l'analyse de chaînes hexadécimales et quelques petites modifications pour faciliter l'addition de nouvelles commandes. Le code est utilisable non seulement sur le STM32, mais aussi, moyennant quelques modifications, sur les µC ESP32 et AVR.

L'interface elle-même utilise le premier UART et on peut y accéder par les broches PA9 (RX MCU) et PA10 (TX MCU). Elle est paramétrée pour une vitesse de 115200 bauds, 8 bits de données, sans parité et un bit d'arrêt (115200Baud 8N1). Pour la fenêtre terminal de l'EDI Arduino, il suffit de régler la vitesse à 115200 bauds.

La quatrième partie est une bibliothèque minimale de gestion de la basse consommation, qui fait en sorte que le STM32 passe en sommeil au bout d'un temps prédéfini et puis se réveille. Dans ce mode, le contenu de la RAM est maintenu et l'horloge en temps réel continue de fonctionner. Le réveil est provoqué par une alarme gérée par l'horloge en temps réel. Il existe une bibliothèque bien plus riche, mais qui consomme 12 Ko de mémoire flash.

| IEEE 754 float (4Byte ) | IEEE 754 float (4Byte) | 4 Byte Status |
|---|---|---|

*Figure. 8. Paquet de données pour une position GPS.*

| Reserved Bit 31 - 16 | VBat/2 Bit 15 - 8 | Time untill GPS-Fix Bit 7 - 1 | GPS Invalid Bit 0 |
|---|---|---|---|

*Figure. 9. Signification de l'octet d'état*

Dans la description des diverses composantes, il ne faut pas oublier que pour émettre une information avec la bibliothèque LMIC, on ne dispose que de 51 octets. Plus l'information est courte, plus on dispose de temps d'activité et moins le LoRaWAN et la bande de fréquence 868 MHz sont encombrés. Dans la version courante du micrologiciel on émet 12 octets pour une position (**fig. 8**), ce qui n'est pas optimal et pourrait être réduit à 7 octets. La longitude est contenue dans la première valeur flottante au format IEEE754, la latitude dans la seconde. Les 4 octets suivants (**fig. 9**) contiennent diverses informations de statut (les données de position sont-elles valides, au bout de combien de temps la position GPS a-t-elle été déterminée, état de la batterie). On peut utiliser ces données comme informations complémentaires. Le code étant ouvert, on peut y ajouter ses propres valeurs.

## Optimisation de la consommation poussée à ses limites

L'énergie fournie par la batterie étant finie, on doit porter une grande attention à l'optimisation de son utilisation, et donc couper ou mettre en sommeil tous les consommateurs inutiles. La gestion du module GPS est simple : une broche GPIO du contrôleur fait changer d'état la broche de validation de la diode idéale IC6, ce qui commute l'alimentation du module. Le module LoRa se met de lui-même en sommeil après l'émission. La feuille de caractéristiques du STM32 laisse supposer que le courant consommé serait de l'ordre de 68 µA, mais si l'on mesure le courant du circuit sans précaution spéciale, on trouve 0,5 à 0,7 mA, dont une grande partie est consommée par le microcontrôleur. Où passe ce courant ? La réponse se trouve chez les broches GPIO non utilisées. Chaque broche non explicitement définie dans le code travaille en mode entrée flottante par défaut (voir l'encadré *Consommation électrique des broches GPIO non utilisées*). Comme mesure

correctrice, on devrait toujours activer leur résistance de rappel interne pour que ces entrées soient à un potentiel déterminé. On peut encore économiser un peu d'énergie en mettant les broches de l'UART communiquant avec le module GPS en mode entrée quand ce module est coupé, sinon la broche TX de l'ESP32 pourrait appliquer 3,3 V à la broche E/S du module GPS.
Outre le STM32, il y a d'autres composants consommateurs d'énergie, par ex. le régulateur de tension, les diodes idéales, le module LoRa et les condensateurs.Pour ne pas être plus royalistes que le roi, nous laisserons les choses en l'état. La consommation de la balise LoRa-GPS au repos s'établit alors à 150 µA à une température ambiante de 25 °C.
Quand la balise se réveille, la consommation augmente sensiblement. La plus grande part en revient au module GPS, qui a besoin de 60 mA et qui détermine l'autonomie de la balise. On comprend pourquoi il est important qu'il évalue la position de la balise le plus rapidement possible ! Le module LoRa requiert 100 mA en pointe pendant l'émission, mais celle-ci dure moins d'une seconde. À une fréquence d'horloge de 48 MHz, le µC consomme 2,5 mA et dispose d'un temps suffisant pour le traitement des données de position et leur envoi vers le LoRaWAN. Si la puissance de calcul maximum n'est pas nécessaire, on devrait réduire autant que possible la fréquence de l'horloge. L'autonomie maximale de la batterie dépend donc essentiellement de l'intervalle entre deux relevés de position et la durée (*timeout*) au bout de laquelle la tentative de relevé de la position par le module GPS est abandonnée en cas de mauvaise réception.

## Réglage des paramètres

Pour qu'il ne soit pas nécessaire de télécharger un nouveau micrologiciel à chaque modification d'un paramètre, l'interface sérielle utilisateur (à utiliser par ex. avec la fenêtre terminal de l'EDI Arduino) permet quelques réglages de base :

- intervalle entre deux émissions de la position GPS
- temps maximum d'une tentative de relevé de la position
- vitesse d'émission LoRa

Ces trois paramètres ont une influence sur la consommation d'énergie de la balise. L'intervalle entre deux émissions est réglé avec **set INTERVALL** et peut être lu avec **get INTERVALL**. Il en est de même avec la position GPS, réglée par **set GPSTIMEOUT** et lue par **get GPSTIMEOUT**. Les valeurs initiales sont 15 mn pour l'intervalle d'émission et 10 mn avant l'abandon en cas de réception GPS mauvaise ou absente.

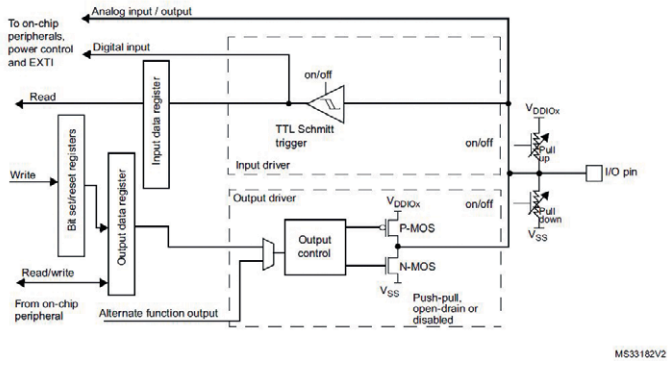
La vitesse de transfert de données LoRa peut être réglée dans le domaine SF7…SF10, où SF7 est la vitesse maximum et donc le temps d'émission le plus court, et SF10 une vitesse lente où la modulation a besoin de plus d'énergie par unité de temps. SF7 est préréglée pour une transmission rapide, mais aussi une portée limitée. Lorsqu'on augmente la valeur de SF7, cela accroît la portée, mais peut conduire à des perturbations de la transmission si la balise est en mouvement : une lame à deux tranchants ! Cela réduit aussi le nombre de positions transmissibles par jour. *The Things Network* prévoit un temps d'émission quotidien de 30 secondes ; avec SF7, il faut compter une durée d'émission de 62 ms et 412 ms avec SF10, ce qui représente par jour 450 envois pour SF7 et 70 envois pour SF10.
Pour pouvoir transmettre des données par LoRaWAN, il faut inclure les clés du destinataire final. La balise LoRa-GPS utilise le mode ABP. Nous en arrivons ainsi à la configuration dans *The Things Network*.

## Configuration dans The Things Network

L'accès au réseau *The Things Network* a déjà été traité en détail dans le numéro de mars/avril [4], nous pouvons donc nous contenter de l'effleurer ici. Après avoir créé un compte utilisateur et s'être connecté, il faut commencer par cliquer sur *Applications* puis sur *Add application* pour créer sur la page suivante une application à associer à la balise. Dans le dialogue suivant, il faut saisir un nom unique (*gps_lora_tracker_app*) et une description (*Test Sensor Nodes for Developement*) de l'application.

Les appareils LoRa sont toujours associés à une application pour que les données incidentes puissent être immédiatement décodées et traitées de manière appropriée dans *The Things Network*. Pour la balise LoRa-GPS ceci ne joue tout d'abord aucun rôle, le traitement des données étant effectué par un serveur (un RPi) avec Node-RED. L'application créée doit encore être associée à un appareil, en l'occurrence, notre balise LoRa-GPS. Avec *Register device*, on accède à un dialogue qui permet de saisir les données de la balise. Pour chaque balise, il faut indiquer un nom unique et terminer le dialogue avec *Register*. Dans l'appareil ainsi créé, sous *Settings*, il faut changer la méthode d'activa-

## QUE CONSOMMENT LES BROCHES E/S INUTILISÉES ?




Dans le *Reference Manual RM0091* on trouve une description de la logique d'une broche GPIO. La bascule de Schmitt intégrée assure que lorsque qu'elle fonctionne comme entrée logique, la broche fournit toujours un niveau déterminé, haut ou bas selon le résultat de la comparaison de la tension lue avec des seuils fixés. Si la broche est laissée flottante, elle se comporte comme une antenne, si bien qu'il arrive une tension fluctuante sur la bascule de Schmitt, ce qui la fait fréquemment commuter entre haut et bas et entraîne une (faible) consommation de courant. Les broches non utilisées sur la carte totalisent ainsi une consommation de courant évaluée à 0,5 à 0,7 mA.

## BROCHES À FONCTIONS VARIABLES : QUE DU PLAISIR !

Les broches à fonctions variables *(Alternate Pin Functions)* ne sont apparues que tout récemment dans le monde des AVR, alors que chez le STM32, on les connaît depuis pas mal de temps. Si pour l'Atmega128, on devait encore décider si une broche était une GPIO ou remplissait une fonction particulière telle que SPI, chez le STM32 une broche peut changer de fonction. On peut ainsi utiliser pour l'UART0 les broches PA8 et PA9 ou bien les broches PB6 et PB7. Pour toutes les broches à fonctions variables, on devrait décider très tôt, même si elles ne sont pas utilisées tout de suite, de les rendre accessibles en les routant vers l'extérieur. Il serait bien fâcheux, une fois le projet terminé, de constater au vu de la feuille de caractéristiques, qu'une broche non routée gère le signal d'horloge du deuxième port SPI qui serait bien utile. C'est pourquoi, sur entente entre les concepteurs du matériel et du logiciel, l'étude du circuit imprimé devrait rendre disponibles pour des utilisations ultérieures certaines des *Alternate Pin Functions*.

tion de OTAA à ABP pour pouvoir accéder aux données qui doivent être intégrées au micrologiciel de la balise. Après *Save*, on peut prélever dans la vue d'ensemble les informations de réglage (*Network Session Key*, *Application Session Key* et *Device Address*) à envoyer à la balise par l'interface sérielle.

Pour éviter la saisie manuelle des trois clés, utilisez le presse-papiers pour les copier depuis le *TheThingsNetwork* (*Device Overview*) dans le moniteur sériel. Lors de la transmission, les clés sont converties en interne et stockées dans l'EEPROM. Une fois que tous les paramètres ont été transmis, la balise est prête pour sa première utilisation.

### De la balise à la carte géographique

Après sa configuration, la balise envoie vers le LoRaWAN des données qui doivent être traitées pour qu'une position puisse être affichée sur une carte géographique, par exemple. Ce traitement peut être confié à Node-RED tournant sur un RPi (ou sur une machine virtuelle, ou sur un ordinateur sur lequel il est installé). L'OS Raspberry Pi (ci-devant Raspbian) ne contient pas Node-RED. L'installation sous Raspbian (version Jessie ou supérieure) est décrite en détail sur la page web du projet [7]. Un article [8] est consacré à une introduction à Node-RED.

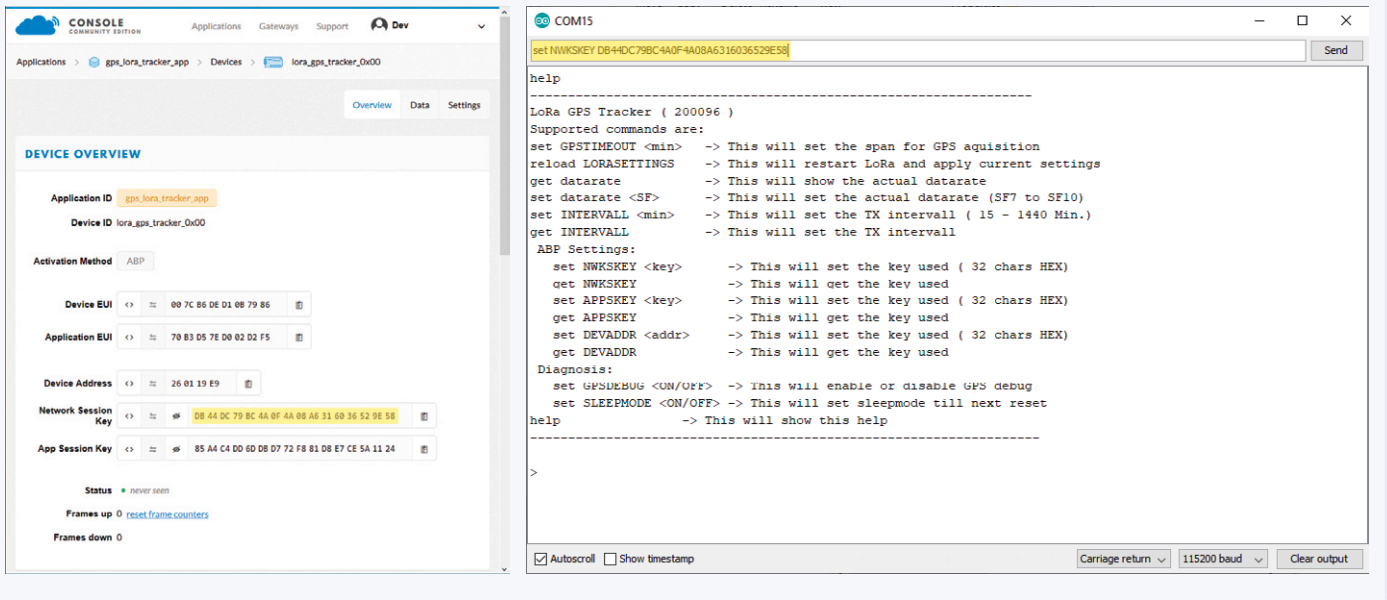



*Figure. 10. Copie des clés depuis la fenêtre* **Device Overview** *dans le moniteur sériel.*

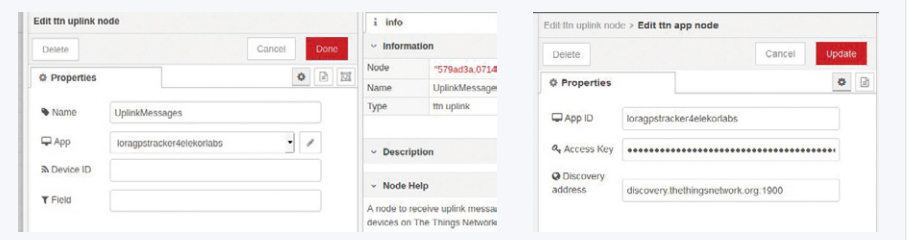



*Figure. 11. Transfert des données TTN vers NodeRED (***Node_Red_TTN_node_Info** *et* ***TTN_Access****).*

**Votre avis, s'il vous plaît ?**
Veuillez adresser vos questions et vos commentaires à l'auteur :
mathias.claussen@elektor.com

Les données sont présentement envoyées par la balise au *TheThingsNetwork*, qui ne les stocke pas, mais les envoie à un système prêt à les recevoir. Comme interface de récupération des données, on utilise le protocole MQTT qui a déjà servi dans d'autres projets Elektor. On profite ainsi de la présence dans Node-Red d'éléments déjà prévus pour le *TheThingsNetwork*, qu'il suffit de configurer au moyen de quelques paramètres, ce qui simplifie considérablement le travail. Le nœud *UplinkMessages* requiert l'établissement préalable d'une liaison avec l'application. Pour cela, on a besoin de l'*Application ID* et de l'*Access Key* de la console de *The Things Network*, ce qui permet de s'identifier comme utilisateur légitime auprès des serveurs et de recevoir les données. Ces données doivent être copiées depuis la console TTN et insérées aux emplacements correspondants du nœud *UplinkMessages* (**fig. 11**).

Dès que le nœud est choisi, on ouvre un nouveau dialogue en cliquant sur l'icône crayon à côté de *App*. On y dépose les données de la console TTN par copier-coller, on valide avec *Update* et on termine avec *Done*. Comme pour toutes les modifications, un *Deploy* est nécessaire. Le nœud devrait maintenant afficher un *Connected* dans sa vue d'ensemble (**fig. 12**).
On reçoit ainsi du nœud de nouvelles données à traiter. Les informations ne contiennent pas seulement les données utiles envoyées par la balise, mais d'autres données comme les passerelles qui ont reçu l'information. À la configuration, l'utilisateur peut préciser l'emplacement des passerelles, ce qui va bientôt se montrer très utile.

Si la balise n'a pas été en mesure de relever une position exacte, elle envoie quand même des données toutefois sans aucune position GPS valide. On peut donc essayer de déterminer au moins la position de la passerelle qui a reçu les données. On ne peut pas en déduire la position exacte de la balise, mais avec les réglages en cours on obtient une zone circulaire d'un rayon de 2 km environ, dans laquelle la balise doit se trouver (fig. 13).
L'affichage lui-même est effectué par une extension de World-Map, une carte interactive qu'on peut afficher dans un navigateur. Dans l'exemple Node-RED joint, on peut y accéder par **:1883/worldmap** (**fig. 14**).
Les données cartographiques proviennent du projet OpenStreetMap. Cette carte accepte des données de position d'un objet qu'elle affiche sous la forme d'une icône, d'un rayon

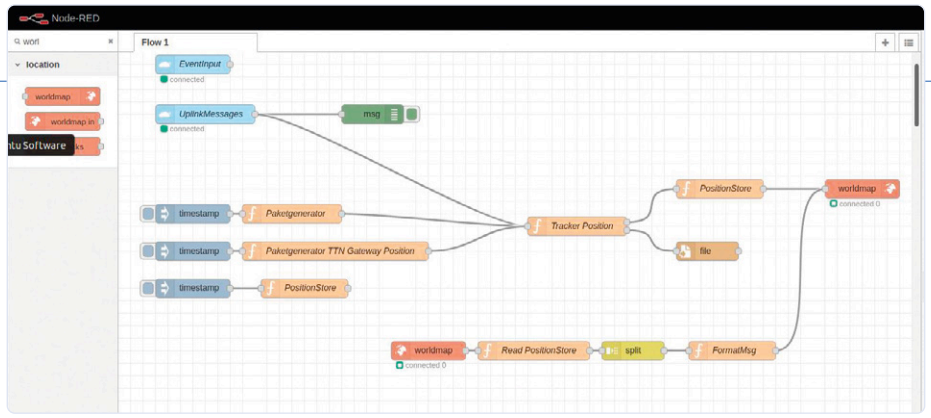



*Figure. 12. Organigramme de Node-RED.*

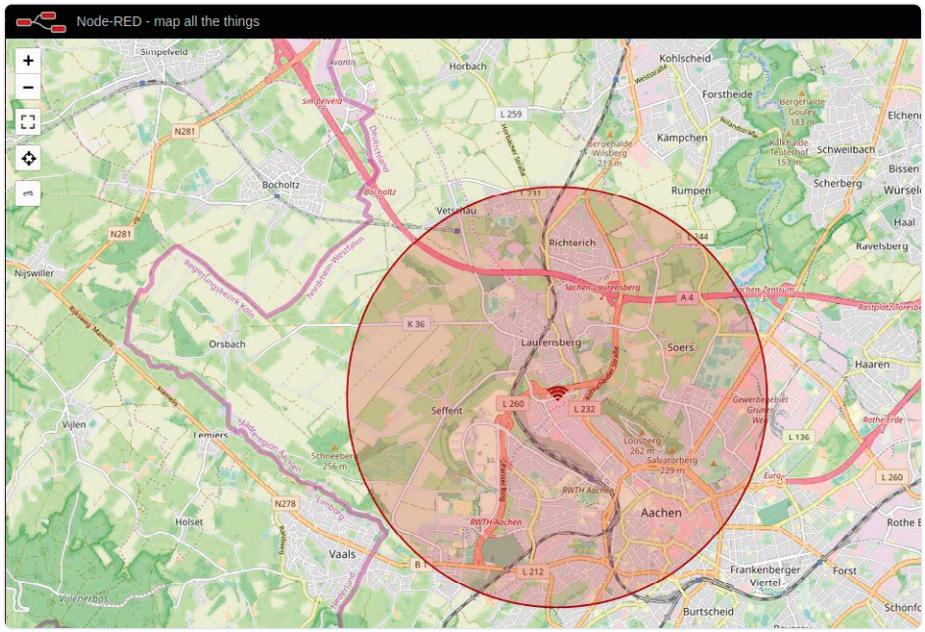



*Figure. 13. Il est possible de localiser la balise même sans réception GPS (© OpenStreetMap contributors).*

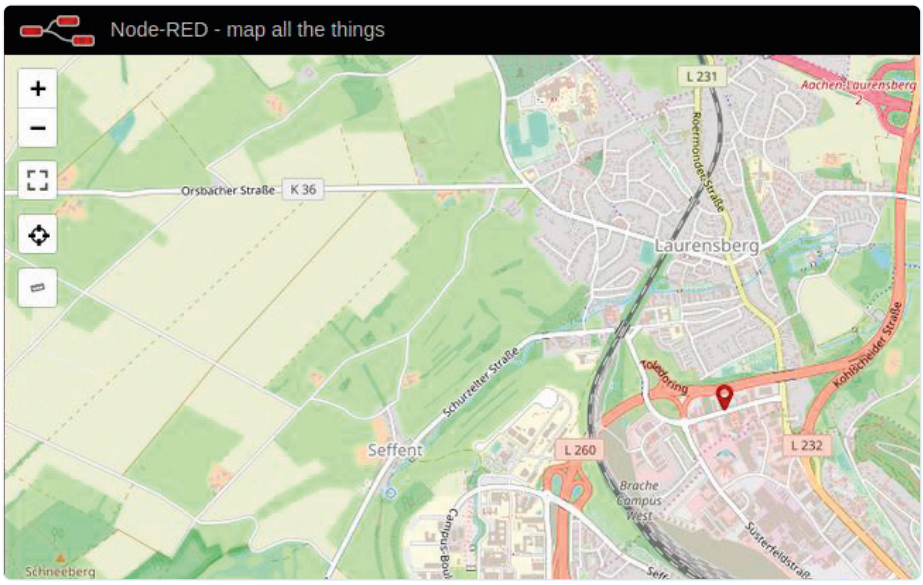



*Figure. 14. La balise dévoile la position exacte du quartier général d'Elektor. Vous savez maintenant où nous trouver ! (© OSM contributors).*

## MODULES REQUIS

Elektor LoRa Node (circuit imprimé disponible chez Elektor*, liste des composants et fichiers KiCAD, micrologiciel sous [10])
RFM95 Ultra LoRa Transceiver Module (868/915 MHz) *
OPEN-SMART GPS – Serial GPS Module *
Câble SMA avec Pigtail et picots
Antenne SMA 868-MHz
Coffret Hammond 1551K
Câble batterie avec connecteur 2 mm RM (Molex 51005)
Batterie (voir texte)

* cf. encadré ci-dessous

ou de quelque autre manière. Il y a un inconvénient : ces données ne survivent pas à la fermeture du navigateur. C'est pourquoi les derniers 2048 points de données soumis à la carte sont stockés dans la RAM du serveur Node-RED. Lorsque la carte est affichée à nouveau, on y retrouve ces points. Mais pour que les itinéraires et les changements de position pendant un temps assez long restent intelligibles, ils sont stockés, pour chaque balise, dans un fichier CSV compatible Excel. L'affichage sur la World-Map n'est donc pas une solution complète et définitive pour la représentation des données de position, elle est plutôt un moyen de visualiser nettement le flux de données de la balise vers l'utilisateur et de vous servir de base pour le développement d'idées personnelles.

## Dans l'e-choppe d'Elektor

- **Elektor LoRa Node circuit imprimé nu 180516-1**
  www.elektor.fr/180516-1
- **RFM95 Ultra LoRa Transceiver Module (868/915 MHz)**
  www.elektor.fr/18715
- **OPEN-SMART GPS – Serial GPS Module**
  www.elektor.fr/18733
- **USB to TTL Converter UART Module CH340G (3.3 V/5.5 V)**
  www.elektor.fr/19151
- **Livre «Programming with Node-RED»**
  www.elektor.fr/19224
  www.elektor.fr/19225

## Conclusion

LoRaWAN offre une plateforme intéressante pour suivre la position d'objets. Avec la solution proposée ici, on peut déterminer la position d'objets, la transmettre par LoRaWAN et l'afficher sur son propre ordinateur. Le caractère ouvert du matériel et du logiciel de cette solution permet à chacun de jeter un regard dans les coulisses et de s'en servir pour ses propres besoins.
Les données complètes de la balise LoRa-GPS sont disponibles sur GitHub [10] en tant que projet KiCad et peuvent être utilisées sous la licence OSHL V1.2. On trouvera aussi le logiciel sur GitHub, ainsi que le projet Node-RED d'affichage de la position. Après son utilisation comme balise, nous présenterons dans le futur (hélas indéterminé pour cause de Covid-19) d'autres applications du nœud LoRa Elektor.

200096-02

**Ont contribué à cet article :**
Conception, texte et figures : **Mathias Claussen**
Rédaction : **Rolf Gerstendorf**
Traduction : **Helmut Müller**
Maquette : **Giel Dols**

## LIENS

[1] **ElekTrack** : http://www.elektormagazine.fr/magazine/elektor-200710/10840
[2] **Itinérance ruineuse** : https://www.welt.de/vermischtes/article202525470/Roaming-Gebuehren-Als-die-Adler-in-den-Iran-flogen-wurde-es-teuer-fuer-die-Forscher.html
[3] **Noeud LoRa (Elektor mars-avril 2020)** : https://www.elektormagazine.fr/magazine/elektor-142/57184
[4] **LoRaWAN – décollage facile (Elektor mars-avril/2020)** : https://www.elektormagazine.fr/magazine/elektor-142/57188
[5] **ESP32 comme serveur de temps** : https://www.elektormagazine.fr/magazine/elektor-101/50947
[6] **Poste de soudage de CMS compact** : http://www.elektormagazine.fr/magazine/elektor-71/42314
[7] **Node-RED Getting Started** : https://nodered.org/docs/getting-started/raspberrypi
[8] **Prise en main de Node-RED** : http://www.elektormagazine.fr/magazine/elektor-152/58792
[9] **Page du projet sur Elektor Labs** : https://www.elektormagazine.fr/200096-01
[10] **Fichiers KiCad sur GitHub** : https://github.com/ElektorLabs/180516-Elektor_LoRa_Node