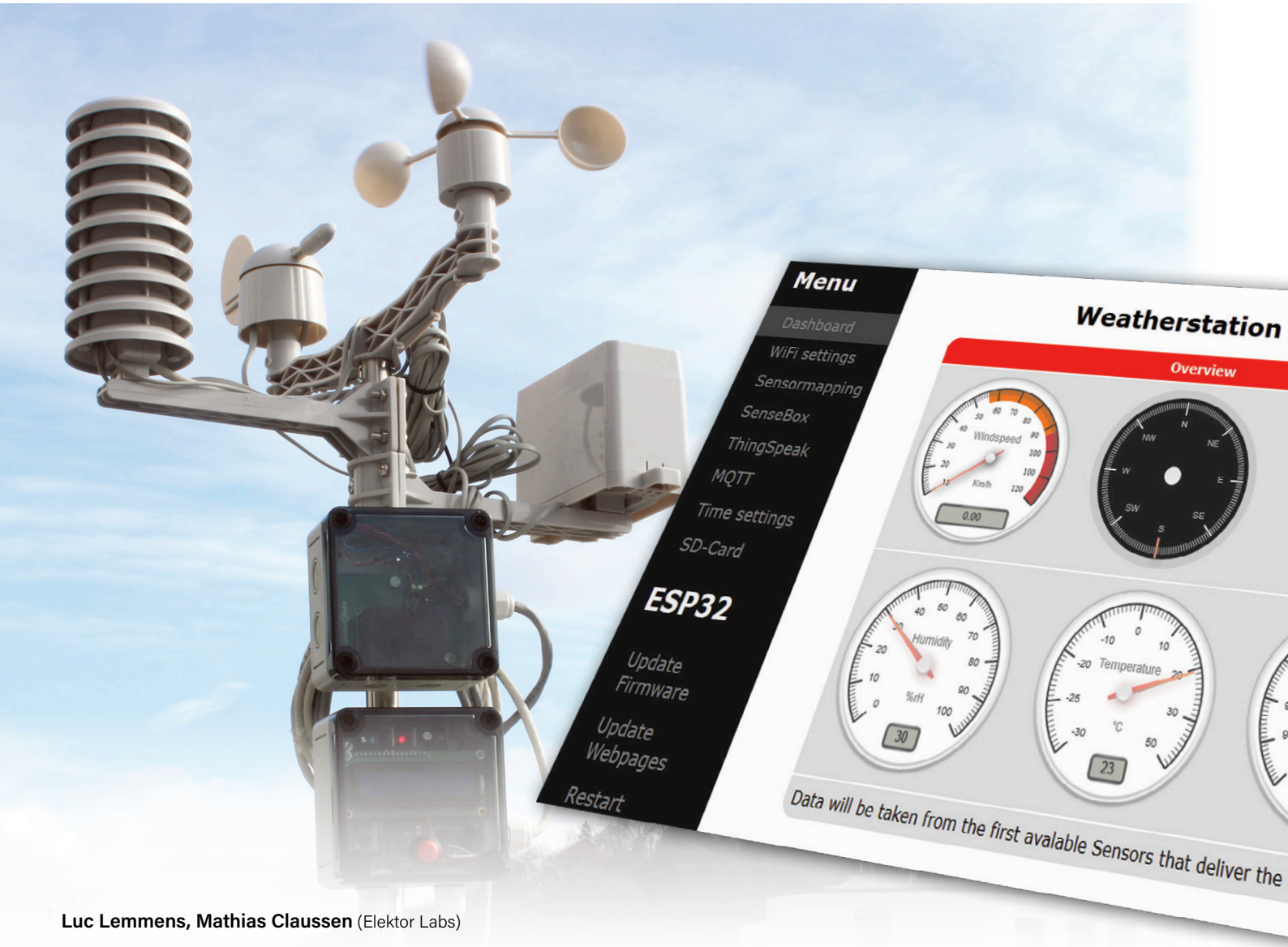
station météo en réseau ouvert V.2

2ème partie : le logiciel



Luc Lemmens, Mathias Claussen (Elektor Labs)

Dans le numéro de mai/juin 2020 [1], le labo d'Elektor présentait sa nouvelle station météo en réseau, à code source ouvert. Le système se compose d'un kit mécanique (anémomètre et pluviomètre, coffret) disponible dans la boutique en ligne Elektor ainsi que d'une électronique autour d'un ESP32, pour envoyer les données collectées à des services dématérialisés comme *openSenseMap* et *ThingSpeak*. Ce matériel n'est qu'une partie du projet. Ici vous apprendrez comment interagissent les blocs fonctionnels du logiciel modulaire. Grâce à un cartographe intelligent, les données des capteurs peuvent être reliées de manière flexible aux canaux de communication. Le serveur web, qui joue un rôle central dans le projet, est également examiné en détail.

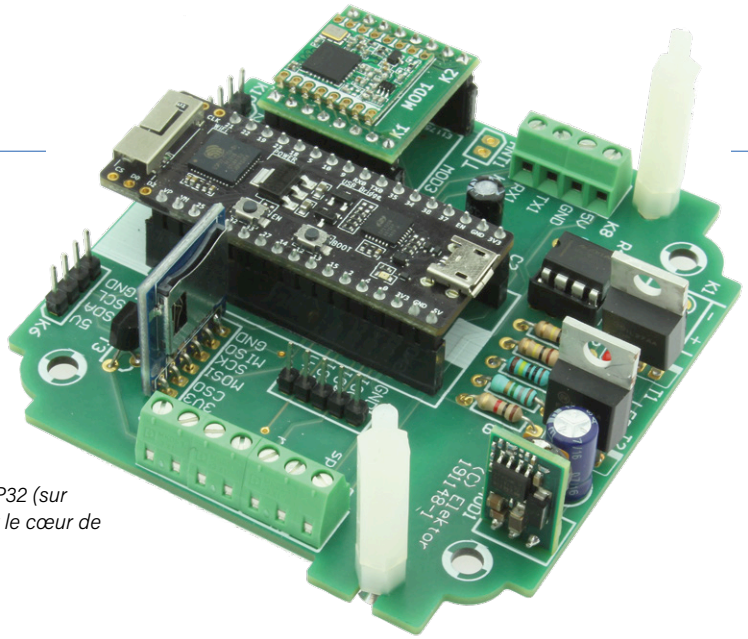


Figure 1. Le µC ESP32 (sur le module noir) est le cœur de la station météo.

Au labo d'Elektor, nous apprécions les suggestions de nos lecteurs pour améliorer ou enrichir nos projets. D'ailleurs, dès le début, nous prenons soin de programmer les logiciels de manière à pouvoir réaliser rapidement les extensions ultérieures. Le matériel de notre station météo [1] peut aussi être facilement étendu. Avec sa matrice E/S, l'ESP32 (**fig. 1**) offre une grande souplesse d'accès sur presque toutes les broches aux fonctions requises, qu'il s'agisse de SPI, I²C ou simplement d'une entrée numérique pour un poussoir.

Voyons les modules inclus dans notre logiciel et ce qui lui confère sa flexibilité, en commençant par les fonctions mises en œuvre par le logiciel [2][3].

Fonctions de base

Capteurs

Actuellement, les trois paramètres de base (précipitations, direction et vitesse du vent) sont mesurés par des broches GPIO. En outre, un bus I²C commande et lit entre autres le BME280 (température, humidité et pression atmosphérique). Les capteurs suivants sont pris en charge par le logiciel :

- VEML6070 (UVA)
- VEML6075 (UVA/UVB)
- TSL2561 (lumière)
- TSL2591 (lumière)
- WSEN-PADS (pression atmosphérique)

Pour la mesure de concentration de particules fines, le logiciel peut s'adresser par l'UART au SDS011 ou au Honeywell HPMA115SXXX.

D'autres capteurs peuvent être connectés, à condition d'adapter le micrologiciel.

Au démarrage, le microprogramme cherche les capteurs connectés et tente de sélectionner automatiquement le pilote approprié.

Services dans le nuage

Les services dématérialisés permettent de stocker périodiquement les données de la station météo, à condition d'être connecté. Les services supportés ici sont *ThingSpeak* et *openSenseMap*, ainsi qu'une connexion à un courtier MQTT pour le nuage local. Si vous utilisez un courtier MQTT comme *Mosquitto* sur *Raspberry Pi*, *Node-RED* [4] permet de traiter et de stocker les données.

Pour chaque service, on peut définir individuellement la périodicité du téléchargement des données, et spécifier quelles données transférer à quel service.

Carte SD, afficheur et bouton

Pour permettre le stockage de données hors connexion WiFi, la station météo est dotée d'un logement pour carte SD, qui accepte des cartes de plus de 4 Go. Les relevés de tous les capteurs sont stockés sur la carte SD au format CSV à des intervalles définis par l'utilisateur. Les données sont écrites directement dans le répertoire principal où un fichier nouveau est créé chaque jour.

L'afficheur de la station est utilisé pour des messages d'état (**fig. 2**). Après le démarrage, l'état du WiFi est affiché ici et un message indique si la carte SD est utilisée par le logiciel ou si elle peut être retirée en toute sécurité. Lorsque l'afficheur est actif, une pression sur un bouton permet d'éjecter ou de monter la carte SD. Si vous n'appuyez pas sur le bouton pendant 30 s, l'afficheur s'éteint pour ne pas gaspiller d'énergie, mais se rallume dès que vous appuyez à nouveau sur le bouton. Outre l'état de la carte SD, l'afficheur signale une connexion à un réseau WLAN existe et, si oui, avec quelle intensité de signal. L'adresse IP de la station est également indiquée.

Ce bouton est connecté aux mêmes broches que le bouton de démarrage de l'ESP32. Après le démarrage, il est mis à la disposition du logiciel utilisateur (voir ci-dessous).

Serveur web et site web

Comme de nombreux projets Elektor, celui-ci fait appel à un serveur web avec options de configuration. Voyez par exemple (**fig. 3**) le réglage de l'heure et de la date. Pour

Figure 2. L'affichage des messages d'état. NOSD indique l'absence de carte de mémoire.

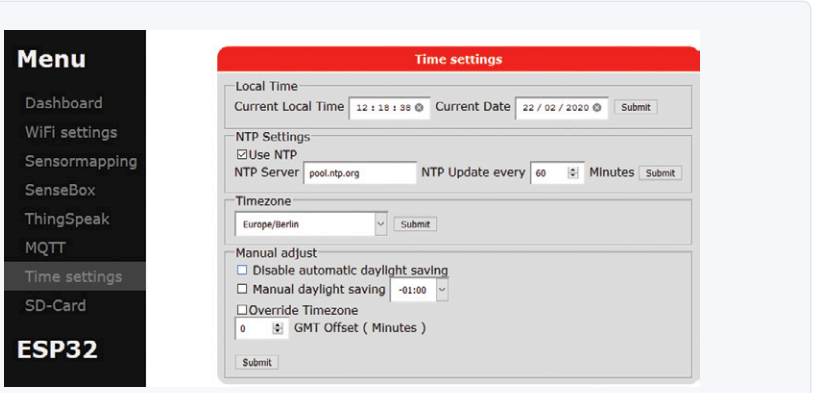


Figure 3. Le serveur web est central dans le logiciel. Il fournit les pages web pour la configuration de la station météo.

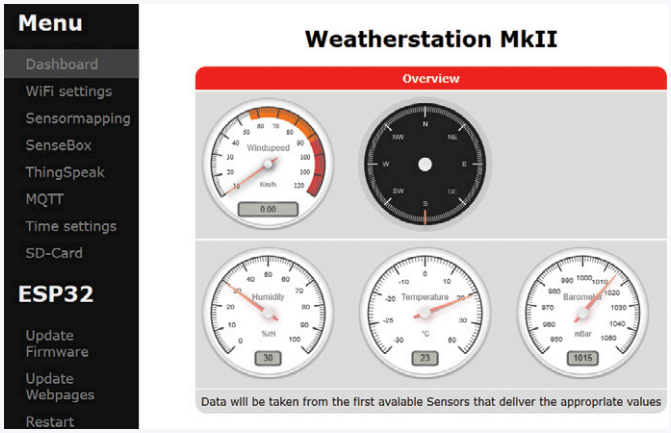


Figure 4. Le serveur web fournit également les principales valeurs de mesure, affichées ensuite dans un navigateur (p.ex. sur un téléphone) sous la forme d'un élégant graphique.

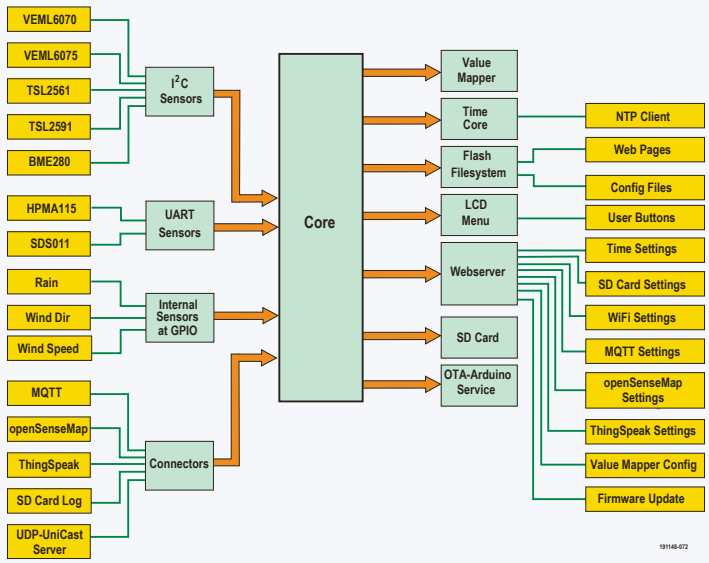


Figure 5. Aperçu des modules logiciels.

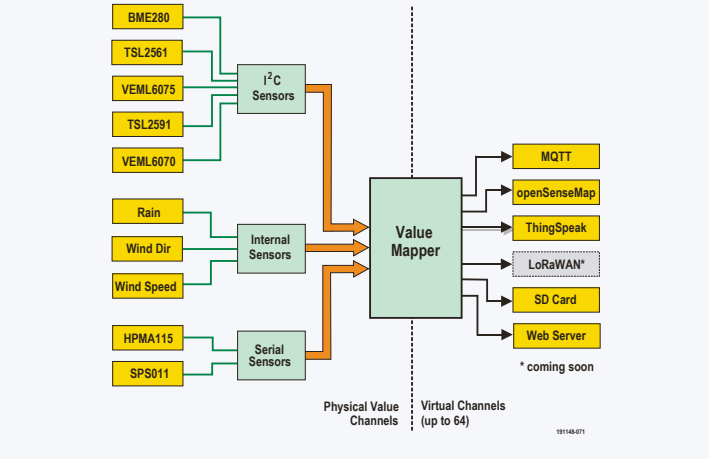


Figure 6. Le cartographe relie les sources de données (capteurs) aux puits de données. Les connecteurs vers les plateformes dématérialisées et au serveur web pour l'affichage). Il fournit 64 canaux de mesure virtuels dans le système.

afficher les données dans un navigateur, il faut plusieurs éléments emboîtés. Un serveur web ESP32 ne fournit pas seulement des pages web affichables dans un navigateur, il envoie également des données à la demande au navigateur. JSON est utilisé pour faciliter l'analyse des données par le navigateur. Les pages web ne se composent pas seulement de HTML, mais aussi de plusieurs lignes de JavaScript.

Cela permet de transférer partiellement la puissance de calcul nécessaire à l'affichage de ESP32 vers le navigateur. Comme le tableau de bord (**fig. 4**), qui permet d'afficher les valeurs mesurées de la station météo dans tel navigateur de votre choix (p. ex. sur le téléphone). Les graphiques ne sont pas transmis, mais dessinés dans le navigateur avec Javascript.

Dans le microprogramme du ESP32

Toutes les fonctions mentionnées nécessitent un logiciel. Voici les modules logiciels (**fig. 5**). Tout d'abord, nous nous penchons sur le traitement des valeurs mesurées. Nous avons ici trois parties principales : le chauffeur ou pilote, pour nous adresser aux capteurs, le connecteur, pour envoyer les données de la station aux services dématérialisés et les écrire sur la carte SD, et enfin, comme médiateur, un cartographe.

Mapper

Le cartographe fournit 64 canaux de mesure virtuels dans le système, qui peuvent être interrogés par les connecteurs. Un canal virtuel dans le cartographe peut être attribué à la valeur mesurée d'un capteur ; notez que les capteurs produisent parfois plusieurs valeurs mesurées (p. ex. le BME280 délivre trois valeurs : température, humidité et pression atmosphérique). Si l'un des connecteurs veut avoir la valeur pour le canal virtuel 0, le cartographe effectue une translation selon une table interne et obtient la valeur correspondante par le pilote de capteur approprié. Ces relations sont illustrées par la **figure 6.** C'est un peu compliqué au début, mais l'avantage est que le connecteur lui-même n'a pas besoin de savoir comment s'adresser au capteur. C'est le boulot du cartographe. Si un autre bus ou un autre type de capteur est ajouté plus tard, seuls le cartographe et les conducteurs doivent être adaptés, les connecteurs eux-mêmes continuent de fonctionner comme avant. Belle équipe !

Le cartographe est bien sûr également connecté au serveur web. Il ne se contente

PROGRAMMATION DE TÂCHES

Dans le logiciel, les **connecteurs** sont des tâches distinctes, de sorte qu'ils peuvent lire et traiter les données des capteurs indépendamment les uns des autres. Cette solution engendre des risques à prévenir à résoudre, à savoir un accès éventuellement simultané au cartographe et donc aux capteurs sur le bus. appelés Appelons Tom et Jerry deux connecteurs actifs, cherchant chacun des données dans le cartographe toutes les minutes et accédant ainsi aux capteurs. Tom serait le premier à lire les données. Il devra peut-être attendre que les données à jour soient disponibles ou converties, et renoncera donc à son temps de calcul par l'unité centrale pour le céder à une autre tâche active. Supposons que ce soit Jerry. Celui-ci doit à son tour chercher des données dans le cartographe et donc dans les mêmes capteurs connectés à un bus. Si c'est le bus I²C, il est possible que Tom vienne de commencer la conversion des données d'un capteur et attende que le capteur soit prêt ; si Jerry commence maintenant à lire aussi les données du bus I²C, surtout à partir du même capteur, c'est la foire !
Dans le système d'exploitation d'ESP32, FreeRTOS, plusieurs mécanismes permettent de résoudre ce problème dont le plus simple est un *mutex*. On peut considérer un mutex comme la clé d'une zone bouclée par une clé unique. Quiconque veut entrer dans cette zone a besoin d'une clé, ouvre l'accès et entre dans la zone qui se referme sur lui. Si d'autres veulent entrer, ils doivent attendre que la clé soit sortie de la zone avec son propriétaire du moment. Le suivant peut alors recevoir la clé. Avec FreeRTOS, le développeur doit s'assurer que la zone est d'abord protégée par un mutex (clé) et qu'une fois ressortie de la zone protégée, la clé (c'est-à-dire le mutex) soit remise en place. Si cela ne se produit pas – ce qui arrive plus souvent que ne le voudraient les développeurs – le système s'arrêtera parce que la clé a été égarée.

Voilà pour l'accès réglementé aux données. Reste à veiller à ce que le temps de calcul soit cédé quand une tâche est oisive. Avec un logiciel sans système d'exploitation, tout fonctionne en super-boucle et il faut des mécanismes de gestion sensée du temps, mais avec un RTOS, une tâche peut être mise en sommeil. Le système d'exploitation cherche alors d'autres tâches qui ne dorment pas et les exécute. Avec les connecteurs (par exemple celui d'*openSenseMap*), le temps entre deux intervalles d'envoi n'est pas passé dans une boucle avec des instructions `delay()`. On utilise un autre mécanisme très similaire au mutex : le sémaphore binaire. Dans FreeRTOS, un mutex et un sémaphore binaire ont, à quelques petites différences près, une fonction très similaire. Si cela vous intéresse, voyez la série d'articles FreeRTOS [5]. Vous pouvez, dans FreeRTOS, spécifier le temps d'attente pour un sémaphore ; si un sémaphore se libère pendant le temps d'attente, système d'exploitation le signale. De cette façon, un temps d'attente ou un intervalle défini peut être créé et le temps de calcul peut être cédé automatiquement.
Pourquoi utiliser un sémaphore et ne pas simplement indiquer au système d'exploitation, avec la commande `vTaskDelay()`, après quel délai continuer la tâche à exécuter ? La réponse réside dans la méthode utilisée ici pour informer la tâche au sujet de changements de configuration que l'utilisateur peut demander par l'interface web. En cas de dépassement du délai lors de l'attente du sémaphore, la modification de la configuration serait perdue. Si en revanche l'attente d'un sémaphore se produit sans erreur, le nouveau réglage est validé. Les paramètres sont alors rechargés complètement et appliqués, et une nouvelle attente commence.

pas d'interroger les valeurs mesurées des différents canaux virtuels afin d'afficher les données elles-mêmes, mais exige également des informations sur les capteurs connectés, pris en charge ou manquants.
Chaque valeur de capteur a une adresse définie dans le logiciel. Il se compose du bus, du type de valeur et de l'identifiant du capteur approprié pour l'accès. Par exemple, «*BME280.0.Pressure.I2C*» signifie : *BME280* est l'identifiant du capteur, un nom lisible pour les humains, *0 est l'*identifiant de la valeur du capteur, *Pressure* est le type de valeur mesurée. L'expression *I2C indique que* le capteur est connecté au bus I²C de l'ESP32. Le cartographe voit ainsi qu'il s'agit d'un périphérique I²C et qu'il faut rechercher le pilote approprié pour l'accès.

Chauffeur

Le chauffeur reçoit du cartographe des informations sur le type de valeur concerné et sur son identifiant. À partir de ces informations, le chauffeur sait à quel capteur s'adresser et quelle valeur extraire de ce capteur.
Pour s'assurer que cela fonctionne pour le bus I²C p. ex., le chauffeur I²C scrute le bus une fois pour tous les capteurs connus après le démarrage. Cela peut se faire par un *START* dans le bus et en vérifiant si un appareil réagit. Si un appareil répond, il est marqué comme disponible et un *STOP est* envoyé. Cette séquence est répétée pour tous les capteurs connus. À la fin, on dresse une liste de ce qui est connecté et actif dans le bus. Avec cette liste, le chauffeur décide si une erreur (*capteur inexistant*) doit être renvoyée directement en réponse à une demande d'accès à ce capteur, ou si un accès à ce capteur doit être tenté. Si la valeur a pu être lue avec succès à partir du capteur, elle est renvoyée au cartographe, sinon c'est une erreur qui est renvoyée.

Connecteurs

Un connecteur est le lien entre les valeurs mesurées et un endroit où les données doivent être stockées (*puits de données*). Les connecteurs sont conçus comme des tâches distinctes dans le micrologiciel ESP32, de sorte qu'ils peuvent lire et traiter les données des capteurs indépendamment les uns des autres. Fondamentalement, chaque connecteur dispose d'une liste de canaux virtuels qui doivent être lus et d'une logique qui garantit que les valeurs, par exemple pour *ThingSpeak*, sont préparées de manière appropriée.
En raison de la programmation des tâches, les connecteurs sont indépendants les uns des autres. Ils posent certains problèmes qui doivent être résolus comme le montre l'encadré **Programmation des tâches**.

Interface web

Nous savons comment les données des

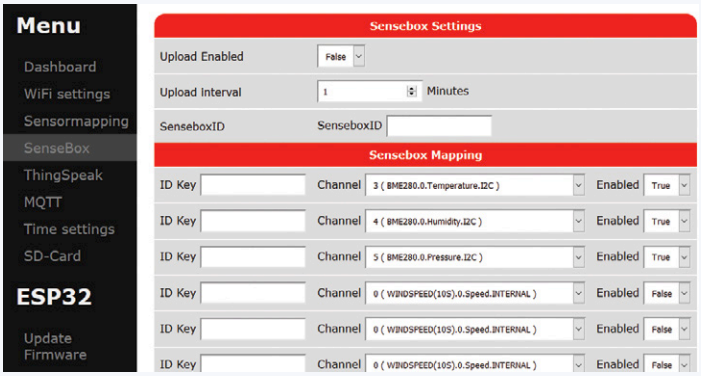


Figure 7. Configuration de l'accès à openSenseMap…

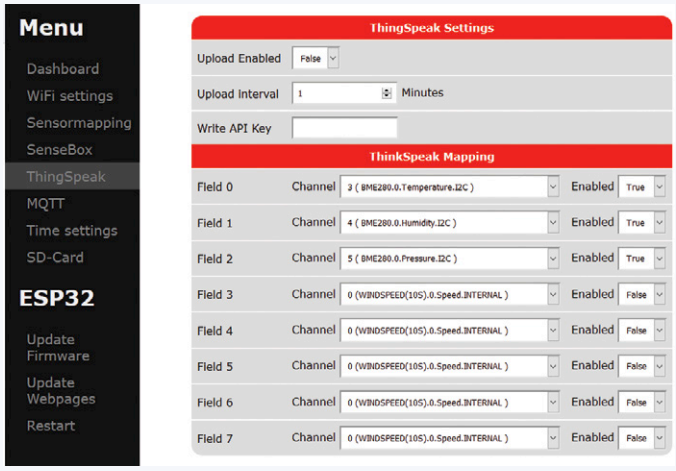


Figure 8. … à ThingSpeak et…

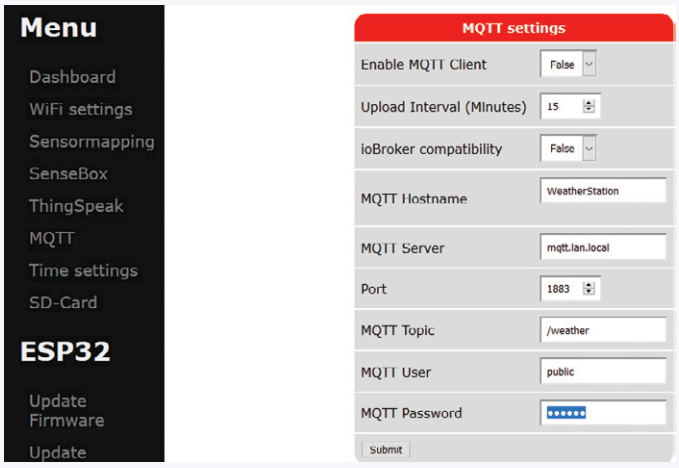


Figure 9. … au courtier MQTT. Chaque page de configuration comporte des éléments de commande différents.

capteurs sont transférées aux services dans le nuage. Nous arrivons à l'interface utilisateur de la station, divisée en plusieurs parties. Pour le cartographe et chaque connecteur, il existe une partie modulaire séparée dans l'interface web, de sorte que les nouveaux connecteurs peuvent être connectés assez facilement au serveur web existant.

Le serveur fournit les pages web et les données dont le navigateur a besoin pour restituer l'interface. Les fichiers idoines se trouvent dans une partie de la mémoire flash de l'ESP32, le SPIFFS. Si vous connaissez cette mémoire, la plupart ne l'auront utilisée dans leurs propres programmes que pour fournir des pages web par le biais d'un ESP32. Or, les SPIFFS peuvent également être utilisés pour écrire des fichiers que le serveur web peut livrer directement, comme un fichier JSON, tout comme une carte SD. Toutefois, cette méthode présente un inconvénient. Les fichiers journaux ne doivent pas être écrits dans le SPIFFS, car la mémoire flash a une durée de vie limitée et ne peut être remplacée aussi facilement qu'une carte SD.

Le microprogramme utilise une partie du SPIFFS pour une partie des fichiers de configuration, principalement pour les connecteurs. Dans les **figures 7, 8** et **9**, vous pouvez voir les pages web qui permettent à l'utilisateur de configurer les connecteurs vers les plateformes dématérialisées *ThingSpeak* et *openSenseMap* ainsi que le client MQTT. Ces pages ont des éléments de contrôle différents. Les pages doivent donc être produites de façon dynamique en fonction du connecteur. Nous avons déjà utilisé un tel mécanisme au labo d'Elektor pour d'autres serveurs web ESP32. En fonction de l'URL que le navigateur envoie au serveur web, celui-ci remplit sa propre fonction. Il est utilisé pour transférer les configurations et les paramètres du système au serveur.

JavaScript

Ce n'est pas seulement le serveur web qui est impliqué dans l'interface web, mais aussi le navigateur. En déplaçant la charge de travail du serveur vers le navigateur, on soulage l'ESP32. Notre site web en fait usage, p. ex. en ne transférant pas les cercles du tableau de bord sous forme de graphiques. Au moment de l'exécution, ils sont dessinés dans le navigateur par des scripts (JavaScript). De même, la préparation des données et des tableaux ne se fait plus dans l'ESP32, mais directement par les scripts de la page web. Pour que cela soit possible, ESP32 doit créer un moyen pour la page web d'accéder aux

ESP32 ET JAVASCRIPT

Par défaut, les navigateurs modernes chargent plusieurs fichiers à la fois. Selon le navigateur, cela peut être 4, 6, 8 ou davantage. Pour éviter à l'utilisateur d'avoir à attendre la fin du chargement de tous les fichiers, le navigateur commence à exécuter des scripts, p. ex. pour afficher les valeurs, les tableaux et les éléments d'affichage de la page. Si les scripts sont distribués dans plusieurs fichiers par souci de clarté du code, il est possible que des parties du code ne soient pas encore chargées et que l'exécution se plante. Un script a donc regroupé tout le code dans un seul fichier pour éviter le problème du morcellement. Ce problème ne s'est manifesté que sur l'ESP32, alors que le serveur web du système de développement était lui capable de fournir les données assez rapidement.
Un autre obstacle à négocier a été, dans JavaScript, le chargement de fichiers multiples effectué lors de l'exécution du script. Leur chargement (ou non) est signalé par un rappel (*Callback*). Si plusieurs fichiers sont nécessaires au même endroit, le chargement devra être enchaîné, c'est-à-dire que le fichier 1 devra être chargé et qu'un rappel pour le fichier 2 devra être donné, dans lequel le fichier 3 sera chargé. Pour contourner ce problème, la tâche a été automatisée à l'aide d'un tableau d'URL et une mémoire pour les réponses. Dans le fichier *basic.js,* vous pouvez voir comment fonctionne la fonction `loadMultipleData.`

Ne sous-estimez pas l'interaction entre interface web, serveur web et microprogramme dans vos propres projets. Des obstacles surgissent non seulement dans le microprogramme, mais aussi dans les navigateurs qui doivent fonctionner sur des appareils différents. Chaque navigateur a ses particularités et il faut souvent des scripts adaptés pour afficher la page correctement.

données dans le navigateur. C'est là qu'entrent en jeu les fichiers crées dynamiquement. Voici ceux qui peuvent être demandés par le navigateur web :

/setWiFiSettings
/getSSIDList
/mapping/mappingdata.json
/devices/supportedsensors.json
/devices/connectedsensors.json
/mapping/{}/value
/mqtt/settings
/sdlog/settings.json
/sdlog/sd/status
/sensebox/settings.json
/sensebox/mapping/{}
/sensebox/mapping.json
/thingspeak/settings.json
/thingspeak/mapping/{}
/thingspeak/mapping.json
/timesettings

Un fichier JSON est toujours livré comme réponse, de sorte que le JavaScript exécuté dans le navigateur peut l'analyser et le traiter très facilement.

L'expression {} est utilisée comme paramètre d'une fonction dynamique. Par exemple, */mapping/0/value* renvoie la valeur du capteur du premier canal logique, */mapping/1/value* la valeur du second canal logique. */sensebox/mapping/0* renvoie le premier canal logique dont la valeur est transférée à *openSenseMap*. Un coup d'œil sur le code source montre comment l'ensemble a été mis en œuvre. Tout d'abord la commande qui attribue la fonction dans le serveur web :

```
server->on(«/sensebox/mapping/{}», HTTP_GET, GetSenseboxChMapping ) ;
```

Cela indique au serveur web que lors de l'accès à *sensebox/mapping/* la dernière partie de l'URL doit être considérée comme un paramètre et la fonction `GetSenseboxChMapping` doit être appelée. Dans la fonction elle-même, le paramètre est ensuite traité comme une chaîne de caractères avec `String IDs = server->pathArg(0) ;.`

Un problème possible, né de la combinaison de l'ESP32 et de JavaScript, est signalé dans l'encadré **ESP32 et JavaScript**.

Démarrage du système

Un LCD pour l'affichage des données est parfois très utile, et il faut au moins un bouton pour le faire fonctionner. On s'efforce bien sûr de réduire le nombre de trous dans le boîtier d'une station météo. Voici donc un petit aperçu de l'affichage et de la fonction du bouton de démarrage. En appuyant sur la broche de démarrage, l'ESP32 peut être mis en mode *bootloader* au démarrage, après quoi il est disponible pour le logiciel.
Lançons le micrologiciel et appuyons sur le bouton, la station passe en mode AP (une note s'affiche à l'écran). Un nouveau réseau WLAN appelé *ESP32-xx-xx-xx* apparaît, qui peut être connecté à un appareil. Si la connexion est réussie, la page de configuration de la station météo est accessible à l'URL *http://192.168.4.1.* La page pourra vous paraître familière, elle a déjà été utilisée dans d'autres projets. Une fois saisis le réseau WiFi de votre routeur et les données d'accès, l'ESP32 redémarre et doit se connecter au réseau WiFi. Afin d'évaluer la qualité de la connexion, la valeur RSSI est également fournie.
L'afficheur est commandé avec la même touche. Pour économiser l'énergie, l'écran est éteint au bout de 30 s. Pour le réactiver, appuyez une fois brièvement sur le bouton : le rétro-éclairage se rallume et les informations d'état s'affichent. Une fois l'écran réactivé, une nouvelle pression sur le bouton permet, selon l'état affiché, de retirer ou d'insérer la carte SD en toute sécurité. La première ligne de l'écran affiche «_SD_» si la carte est insérée dans le système, ou «NOSD» si aucune carte n'est insérée ou détectée.
Dans le logiciel, il a fallu résoudre le problème des rebonds mécaniques du bouton. Ici, en raison de l'architecture du contrôleur ESP32 et GPIO, le bouton déclenche une interruption, la même sur le front montant et le front descendant. Cela implique qu'au cours de l'interruption, il faut déterminer s'il s'agissait d'un front montant ou descendant. La suppression du rebond se fait en mesurant le temps entre l'appui sur la touche et son relâchement ; si ce temps est de moins de 100 ms, la touche est rejetée. La signalisation d'une frappe est assurée par un sémaphore à partir de l'interruption, de sorte qu'une tâche distincte peut l'interroger.
L'affichage LCD est réalisé de manière similaire. Une tâche distincte met à jour l'affichage toutes les secondes pendant 30 s, puis s'endort et attend une nouvelle frappe.

Paramètres

Une fois le micrologiciel installé, la station météo créera une configuration de base

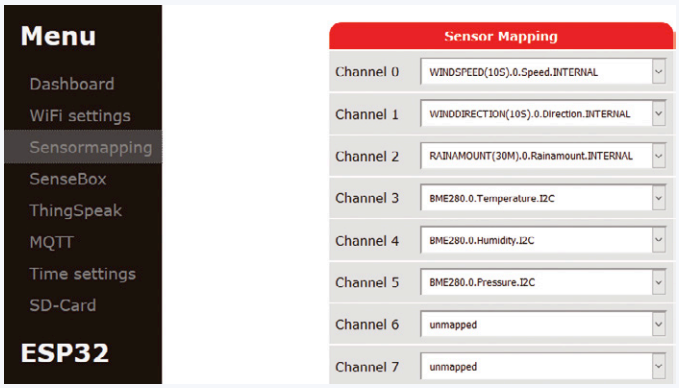


Figure 10. Affectation des valeurs des capteurs aux canaux virtuels.

basée sur un pluviomètre, un anémomètre et un BME280 connectés. Si aucun BME280 n'est connecté, ce n'est pas un problème, les canaux correspondants ne fourniront pas de valeur.
Lors de la configuration, les canaux virtuels doivent être réglés en premier, à raison d'une valeur de capteur affectée à un canal virtuel (**fig. 10**).
Ces canaux sont ensuite interrogés par les connecteurs, qui établissent la connexion avec des puits de données comme *openSenseMap*, *ThingSpeak* ou la carte SD.
Les paramètres eux-mêmes affectent également le tableau de bord. Le tableau de bord utilise les valeurs de la direction et de la vitesse du vent, de la température, de l'humidité et de la pression atmosphérique des canaux virtuels.

Les connecteurs ne sont pas seulement le lien entre station météo et services dématérialisés. Le stockage sur la carte SD est également commandé par un connecteur. Pour la carte SD elle-même, les paramètres sont assez simples : l'intervalle dans lequel les données doivent être écrites et si l'écriture doit être active. Sur le site web, la carte SD peut également être retirée du système (ou montée) en toute sécurité. La capacité et l'espace utilisé de la carte SD sont indiqués à titre d'information. Les données elles-mêmes sont stockées sous forme de fichier CSV dans le répertoire racine de la carte SD, dans un fichier journalier séparé. Pour que l'horodatage des données soit correct, l'heure doit être définie dans les *Paramètres* (fig. 2).

Pour les services dématérialisés, d'autres réglages sont possibles et nécessaires. Pour *SenseBox* et donc *openSenseMap*, l'*ID de la SenseBox et* pour chaque capteur, une clé séparée sont nécessaires (fig. 6). Ces informations peuvent être facilement extraites de l'interface web d'*openSenseMap*. Un canal virtuel peut alors être attribué à chacune de

PRODUITS

- **Circuit imprimé principal, nu**
www.elektor.fr/191148-1
- **Circuit imprimé des connecteurs, nu**
www.elektor.fr/191148-2
- **Circuit imprimé du capteur de particules, nu**
www.elektor.fr/191148-3
- **2×16-Zeichen-LCD mit I2C-Erweiterung:**
www.elektor.de/2x16-character-lcd-blue-white-120061-77
- **Afficheur LCD bleu-blanc 2 x 16 avec I²C (120061-77)**
www.elektor.fr/2x16-character-lcd-blue-white-120061-77
- **Mini module pour carte micro-SD**
www.elektor.fr/19251
- **BoB BME280 , version I²C (160109-91):**
www.elektor.fr/bme280-mouser-intel-i2c-version-160109-91
- **Station météo pro WH-SP-WS02**
(kit avec anémomètre, capteur de pluie, thermomètre hygromètre et support)
www.elektor.fr/professional-outdoor-weather-station-wh-sp-ws02

LIENS

[1] **station météo en réseau ouvert V.2 – 1e partie**: www.elektormagazine.fr/magazine/elektor-148/58640
[2] **page du projet sur Elektor Labs** : www.elektormagazine.com/labs/remake-elektor-weather-station
[3] **la page en ligne de cet article** : www.elektormagazine.fr/191148-B-03
[4] **prise en main de Node-RED** : www.elektormagazine.fr/articles/prise-en-main-de-nodered
[5] **multitâche en pratique avec l'ESP32 (1)** : www.elektormagazine.fr/magazine/elektor-145/57063
[6] **multitâche en pratique avec l'ESP32 (2)** : www.elektormagazine.fr/magazine/elektor-142/57175
[7] **multitâche en pratique avec l'ESP32 (3)** : www.elektormagazine.fr/magazine/elektor-148/58643

ces clés de capteur. Au total, jusqu'à 16 valeurs peuvent être transmises à *openSenseMap*. Pour *ThingSpeak* (fig. 7), les paramètres sont similaires, mais ici, une seule *clé API* est nécessaire pour l'écriture et les champs à transmettre doivent être réaffectés aux canaux virtuels.

Le MQTT offre ici beaucoup moins de paramètres ; cependant, l'utilisateur a besoin de plus d'informations pour traiter les données (fig. 8). Outre l'intervalle auquel les données sont envoyées (et la décision d'envoyer les données), il y a aussi la question de la compatibilité *ioBroker*. Si vous traitez les données, par exemple avec *Node-RED*, vous n'avez pas besoin d'utiliser ce mode. Dans ce cas, les données sont transportées via MQTT afin que le courtier puisse les en extraire. On part du principe d'un transport générique via MQTT, au bout duquel se trouve un système tel que Node-RED [4], capable de traiter une chaîne JSON,

```
// the JSON produced by the
   Station looks like this:
{
 Data:[
    ,
     …


       ]
}
```

Dans le tableau, ne sont transmis que les canaux de `Data` également affectés à une valeur de capteur. Les chaînes sans affectation n'apparaissent pas.

Valeurs moyennes

En déterminant les trois valeurs mesurées de la quantité de pluie, de la direction et de la vitesse du vent, des valeurs moyennes sont formées ; pour la direction et la vitesse du vent sur 10 s, 60 s et 1 h. Pour la quantité de pluie, sont émises les valeurs pour 30 mn, 60 mn, 720 mn et un jour (1440 mn). Ces valeurs sont des moyennes mobiles sur le temps passé. Cette fonction est actuellement encore stockée dans les pilotes des valeurs, mais pourrait être déplacée ultérieurement par le noyau lui-même vers un «enregistreur», qui reçoit les données comme un connecteur et les rend disponibles dans des canaux virtuels. D'autres fonctions mathématiques pourraient également être réalisées avec ce principe.

Possibilité d'extension

Cette station météo est utilisable telle quelle, mais il serait fort surprenant qu'on ne nous propose pas d'étendre ses fonctions. Au labo, quelques idées ont déjà germé, comme le support de capteurs *onewire*. Il serait intéressant de disposer d'un navigateur de fichiers pour l'interface web afin de supprimer ou de télécharger directement les données de la carte SD. Nous attendons vos suggestions, que vous pouvez faire en laissant un commentaire sur la page Elektor Labs [2] !

191148-B-03