détecteur de fuite d'eau

connecté à l'Arduino Cloud

Source : Adobe Stock

Figure 1. Mon compteur de consommation d'eau avec le capteur de débit attaché.

Yves Bourdon (France)

J'ai subi un préjudice de plusieurs milliers d'euros lorsqu'une conduite d'eau sous ma maison s'est fissurée sans que je m'en aperçoive. J'ai alors décidé de créer et d'installer un système de surveillance de la consommation d'eau afin d'éviter que des incidents aussi coûteux ne passent inaperçus à l'avenir. Grâce à l'Arduino Cloud, je peux désormais surveiller ma consommation d'eau sur mon téléphone portable à tout moment de la journée et depuis n'importe où dans le monde. Voici comment j'y suis parvenu.

L'eau est précieuse et devient de plus en plus chère. Il y a un an, une conduite d'eau s'est fissurée sous ma maison sans que je m'en aperçoive immédiatement, avec à la clé une facture d'eau de plusieurs milliers d'euros ! J'ai donc cherché un moyen d'être prévenu au cas où cela se reproduirait. Le service des eaux a installé un compteur intégrant un capteur de débit (**figure 1)** et un petit afficheur à distance sans fil. Malheureusement, la fiabilité n'était pas au rendez-vous, et il n'y avait aucun moyen de m'alerter, si ce n'est un « F » sur l'écran censé indiquer une fuite…

Un article d'Elektor présentait un détecteur de fuite d'eau développé par Denis Lafourcade [1]. Avec son aimable autorisation, j'ai décidé de reprendre son projet en le modifiant assez largement pour l'intégrer à mon système existant. De plus, j'ai voulu utiliser l'Arduino Cloud présenté dans l'édition 2022 d'Elektor Arduino Guest Edition [2].

Conditions préalables

Je voulais pouvoir accéder à mes données sur mon téléphone portable, et l'environnement de l'Arduino Cloud est particulièrement adapté à cet effet. Cela me permettait d'avoir un tableau de bord sur mon téléphone sans avoir à développer une application spécifique qui nécessite des mises à jour constantes. Soucieux de la sécurité, je ne voulais surtout pas ouvrir un port sur mon routeur pour accéder à mes données. Enfin, en cas de fuite, je souhaitais être alerté par une notification push accompagnée d'une alarme sonore.

Spécifications

- Utilisation de l'Arduino Cloud pour collecter des données et afficher des statistiques. Un graphique montre la consommation d'eau quotidienne en temps réel. Jusqu'à trois mois de données sont stockés dans le nuage. En outre, il est facile de télécharger un fichier CSV (valeurs séparées par des virgules) pour traiter les données dans un tableur, par exemple.

- Utilisation d'un processeur ESP32, et plus particulièrement de la carte de développement Heltec WiFi Kit 32 [3], qui comprend notamment un écran OLED et un chargeur de batterie.
- Connexion directe (via un optocoupleur pour éliminer les interférences) au capteur Reed intégré à mon compteur d'eau situé à environ 40 mètres, contre le mur d'enceinte de ma propriété. Notez que si vous n'avez pas de compteur équipé d'un capteur, vous pouvez facilement trouver des adaptateurs de relais Reed pour n'importe quel compteur existant (**figure 2**).
- Le système mesure le débit d'eau en litres par unité de temps au moyen d'interruptions en provenance de mon compteur.
- Chaque heure, la consommation d'eau en litres est enregistrée dans un tableau.
- Chaque jour à minuit, la valeur est enregistrée pour mesurer la consommation journalière (consommation à l'instant *t* moins la consommation à minuit précédent).
- Chaque année, le 31 décembre à minuit, la valeur de la consommation est également enregistrée. Cela permet de calculer facilement la consommation annuelle (consommation à l'instant *t* moins la consommation au 1er janvier).
- Les mesures sont enregistrées par tranches de 24 heures. Cela permet de vérifier la consommation d'eau même dans des tranches horaires inhabituelles. Par exemple, mon adoucisseur d'eau nettoie sa résine à 2h15 du matin tous les 15 jours.
- Toutes les 15 minutes, la consommation des dernières 24 heures est calculée. Si elle dépasse le seuil d'alarme (750 l / jour dans mon cas), une alarme sonore se déclenche, un indicateur virtuel sur le tableau de bord Arduino s'allume, et un email (ou une notification push) est envoyé via l'Arduino Cloud. Tout cela pour s'assurer d'être alerté à temps !
- De même, la présence de deux tranches horaires successives sans consommation d'eau (souvent le cas la nuit) est vérifiée. Si cette condition n'est pas remplie, une fuite d'eau peut être suspectée et des alertes sont envoyées.
- La seule opération de maintenance nécessaire est la saisie périodique de la valeur réelle affichée par le compteur d'eau. La différence entre les mesures effectuées par le relais Reed et la valeur réelle affichée par le compteur est très faible. Je le réajuste environ tous les deux mois.

Se connecter à l'Arduino Cloud

On peut se passer de l'EDI classique d'Arduino sur son ordinateur lorsqu'on travaille avec l'Arduino Cloud [4]. Le compilateur en ligne est suffisant pour développer confortablement une application, et les mises à jour successives et transparentes améliorent considérablement l'environnement de développement. La mise à jour OtA (Over-the-Air) étant native dans cet environnement, on peut choisir de télécharger le logiciel via un port USB (obligatoire pour le premier téléchargement) ou à distance, ce qui est très pratique.

Il existe de nombreux tutoriels sur internet pour apprendre à maîtriser le cloud, mais dans l'ensemble, ce n'est pas si compliqué. Tout est gratuit si on ne dépasse pas cinq variables à rafraîchir sur l'application. C'est peu, mais suffisant pour le détecteur de fuite d'eau si l'on se contente d'un affichage limité : consommation journalière, deux tableaux de tranches horaires, consommation annuelle et valeur du compteur. La version gratuite est limitée à deux applications et il existe des

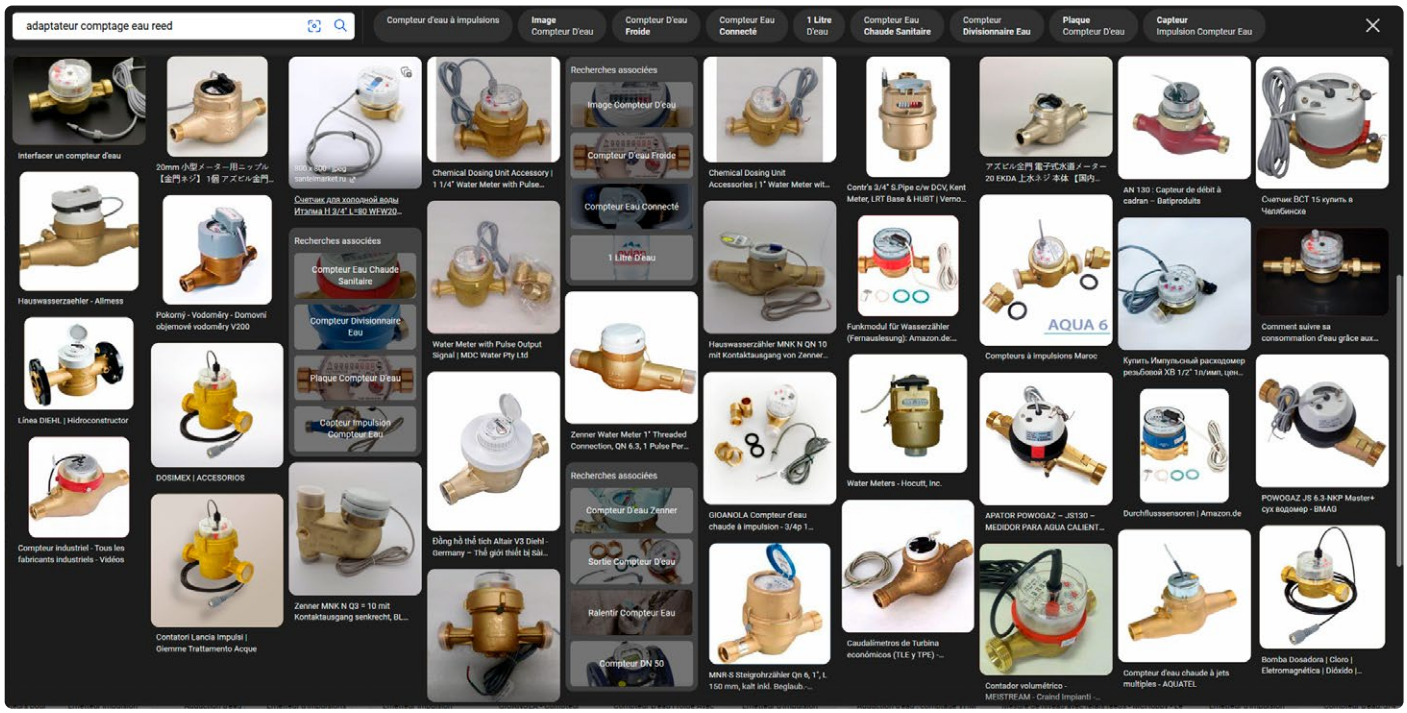


Figure 2. On trouve facilement des capteurs Reed pour tous les types de compteurs.

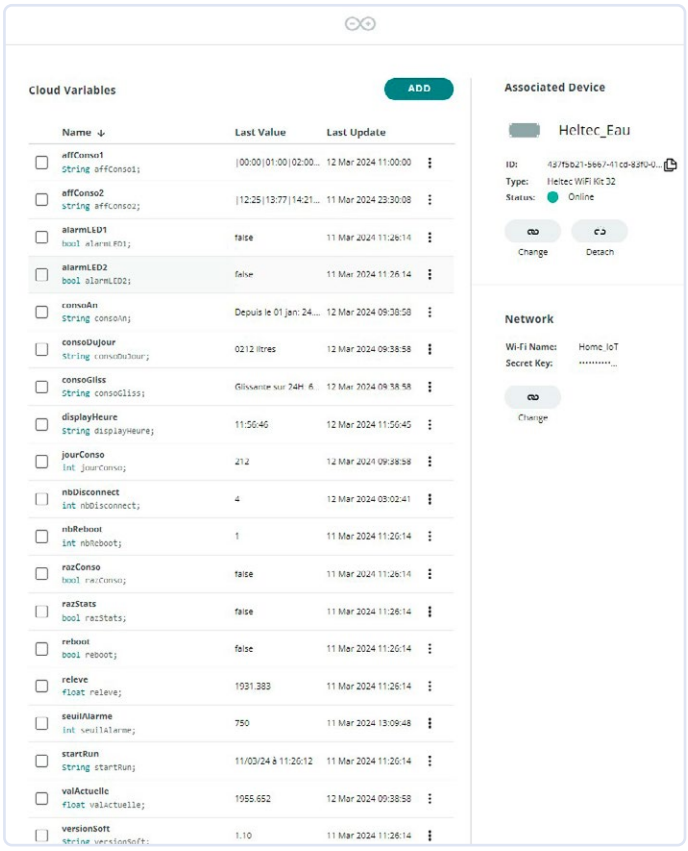


Figure 3. Étape 2 - configuration d'une ***Thing****.*

```
Elektor_Thing_mar12a.ino | ReadMe.adoc | thingProperties.h | Secret Tab

const char DEVICE_LOGIN_NAME[]  = "b41b1548-7f5a-4c33-a20e-a24cc0a67106";

const char SSID[]               = SECRET_SSID;    // Network SSID (name)
const char PASS[]               = SECRET_OPTIONAL_PASS;    // Network password (use for WPA, or use as key for WEP)
const char DEVICE_KEY[]  = SECRET_DEVICE_KEY;    // Secret device password

void onReleveChange();
void onValActuelleChange();
void onSeuilAlarmeChange();
void onRazConsoChange();
void onRazStatsChange();
void onRebootChange();

String affConso1;
String affConso2;
String consoAn;
String consoDuJour;
String consoGliss;
String displayHeure;
String startRun;
String versionSoft;
float releve;
float valActuelle;
int jourConso;
int nbDisconnect;
int nbReboot;
int seuilAlarme;
bool alarmLED1;
bool alarmLED2;
bool razConso;
bool razStats;
bool reboot;

void initProperties(){

  ArduinoCloud.setBoardId(DEVICE_LOGIN_NAME);
  ArduinoCloud.setSecretDeviceKey(DEVICE_KEY);
  ArduinoCloud.addProperty(affConso1, READ, ON_CHANGE, NULL);
  ArduinoCloud.addProperty(affConso2, READ, ON_CHANGE, NULL);
  ArduinoCloud.addProperty(consoAn, READ, ON_CHANGE, NULL);
  ArduinoCloud.addProperty(consoDuJour, READ, ON_CHANGE, NULL);
  ArduinoCloud.addProperty(consoGliss, READ, ON_CHANGE, NULL);
  ArduinoCloud.addProperty(displayHeure, READ, ON_CHANGE, NULL);
  ArduinoCloud.addProperty(startRun, READ, ON_CHANGE, NULL);
  ArduinoCloud.addProperty(versionSoft, READ, ON_CHANGE, NULL);
  ArduinoCloud.addProperty(releve, READWRITE, ON_CHANGE, onReleveChange);
  ArduinoCloud.addProperty(valActuelle, READWRITE, ON_CHANGE, onValActuelleChange);
  ArduinoCloud.addProperty(jourConso, READ, ON_CHANGE, NULL);
  ArduinoCloud.addProperty(nbDisconnect, READ, ON_CHANGE, NULL);
  ArduinoCloud.addProperty(nbReboot, READ, ON_CHANGE, NULL);
  ArduinoCloud.addProperty(seuilAlarme, READWRITE, ON_CHANGE, onSeuilAlarmeChange);
  ArduinoCloud.addProperty(alarmLED1, READ, ON_CHANGE, NULL);
  ArduinoCloud.addProperty(alarmLED2, READ, ON_CHANGE, NULL);
  ArduinoCloud.addProperty(razConso, READWRITE, ON_CHANGE, onRazConsoChange);
  ArduinoCloud.addProperty(razStats, READWRITE, ON_CHANGE, onRazStatsChange);
  ArduinoCloud.addProperty(reboot, READWRITE, ON_CHANGE, onRebootChange);

}

WiFiConnectionHandler ArduinoIoTPreferredConnection(SSID, PASS);
```

Figure 4. Voici ce que vous devriez obtenir après avoir défini toutes les variables.

offres spéciales. Pour ma part, j'ai souscrit un abonnement Maker pour 5,99 € par mois car j'utilise cette application pour de nombreux projets. Avec mon abonnement Maker, mes données sont conservées pendant trois mois, et je peux télécharger un fichier CSV contenant les données horodatées.
Si vous souhaitez bénéficier de toutes les fonctionnalités que j'ai développées (y compris les statistiques sur le fonctionnement et les commandes de connexion avec réinitialisation), je vous recommande de souscrire au moins un mois au plan Maker pour l'essayer.
J'ai optimisé la fiabilité de la connexion comme suit :

- Si la connexion Wi-Fi échoue, le système le détecte et tente de se reconnecter.
- Si la connexion à l'Arduino Cloud est perdue (ce qui peut arriver), le programme le détecte et, après plusieurs tentatives de reconnexion, l'unité centrale est redémarrée.

J'ai également géré l'heure locale comme suit :

- Lorsque le système est connecté à l'Arduino Cloud, il récupère périodiquement l'heure UTC du nuage.
- Comme j'ai besoin de l'heure locale en tenant compte des changements d'heure d'été, l'heure locale de l'ESP32 est mise à jour en tenant compte de ces paramètres.

Étape 1 : Ajouter un dispositif

Cette étape prépare votre système à l'environnement cloud en lui attribuant un Device ID et une Secret Key. Commencez par sélectionner *third-party device* dans le menu Setup Device, puis *ESP32,* et enfin *Heltec WiFi Kit 32* dans la liste déroulante. Regardez bien, car la liste n'est peut-être pas triée par ordre alphabétique, mais ça y est. Donnez-lui un nom. Le nuage vous fournit alors les deux variables qui doivent être sauvegardées correctement en utilisant le PDF téléchargeable.

Étape 2 : Configurer une *Thing*

Il faut ensuite associer les variables que l'on veut afficher ou modifier (attention à respecter la casse !), voir la **figure 3**.
Vous devez définir les types `bool`, `int`, `float`, `string` lors de la création, ainsi que l'attribut *Read & Write.* L'attribut *Read & Write* correspond à une variable qui peut être modifiée dans l'application mobile, comme le seuil d'alarme. L'attribut *Read Only* (lecture seule) concerne les valeurs qui doivent uniquement être affichées, comme l'état de l'alarme. Conservez *Variable update Policy – On change* pour toutes les variables.
Vous devez également indiquer le SSID et le mot de passe du menu *Network*, ainsi que la *Secret Key* précédemment récupérée. La **figure 4** montre le résultat de cette étape.

Étape 3 : Le croquis

Allez dans l'onglet *Sketch* et choisissez *Open full editor*. Vous êtes maintenant dans l'environnement de développement.
Chargez les bibliothèques nécessaires à ce projet : *SSD1306.h*, *OLEDDisplayUi.h*, *TimeLib.h*, *math.h*, et *Preferences.h.* Je les ai placées dans le dossier *Libraries.* Il suffit de les télécharger en utilisant la flèche dans l'onglet *Libraries.*

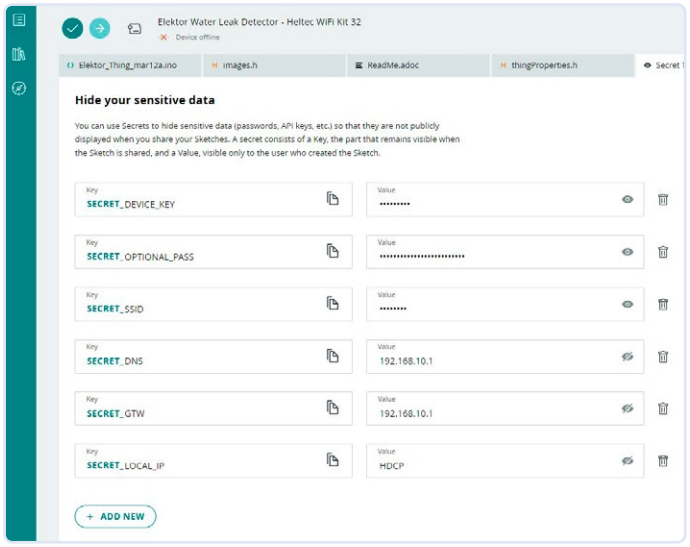


Figure 5. Complétez la page de l'onglet ***Secret*** *ainsi.*

Un bogue ?

J'ai Récemment remarqué un problème sur mon iPhone, mais pas sur ma tablette. Dans le tableau de bord Arduino, il est très compliqué d'entrer une nouvelle valeur pour le compteur d'eau, la dernière lecture du 1er janvier ou le seuil d'alarme. Pour une raison quelconque, la nouvelle valeur est immédiatement effacée avant même que vous n'ayez la possibilité de toucher le bouton Terminé.

Pour contourner ce problème, apparu après une mise à jour d'Arduino Cloud, il suffit d'appuyer sur le bouton de redémarrage, d'entrer la nouvelle valeur et d'attendre que le système redémarre. La nouvelle valeur sera alors prise en compte.

Il se peut que ce problème ait été résolu au moment où vous lirez cet article.

Téléchargez le code source depuis la page de ce projet sur Elektor Labs [5]. Ouvrez le fichier *Elektor_Thing_mar12a.ino* dans l'EDI Arduino classique ou dans un éditeur de texte pour copier facilement le programme dans l'EDI Arduino Cloud. Il suffit de copier et coller le code source dans le premier onglet.

Créez l'onglet *images.h* en utilisant le + de la barre principale, et copiez-collez le contenu de mon fichier. Vous devez également compléter l'onglet *Secret*, comme le montre la **figure 5**.

Compilez le programme en utilisant le bouton correspondant. Si tout est correct, vous pouvez télécharger le programme sur votre carte (après avoir installé Cloud Agent, comme le suggère l'EDI). La compilation est très rapide - rien à voir avec l'EDI classique.

Note : J'ai regroupé plusieurs paramètres dans l'onglet *Secret.* En plus des paramètres par défaut `SECRET_SSID, SECRET_PASSWORD`, et `SECRET_DEVICE_KEY`, vous pouvez définir des paramètres réseau (IP, Gateway, DNS). Si l'adresse IP est *dhcp* ou *DHCP*, alors la connexion Wi-Fi est établie à l'aide de DHCP.

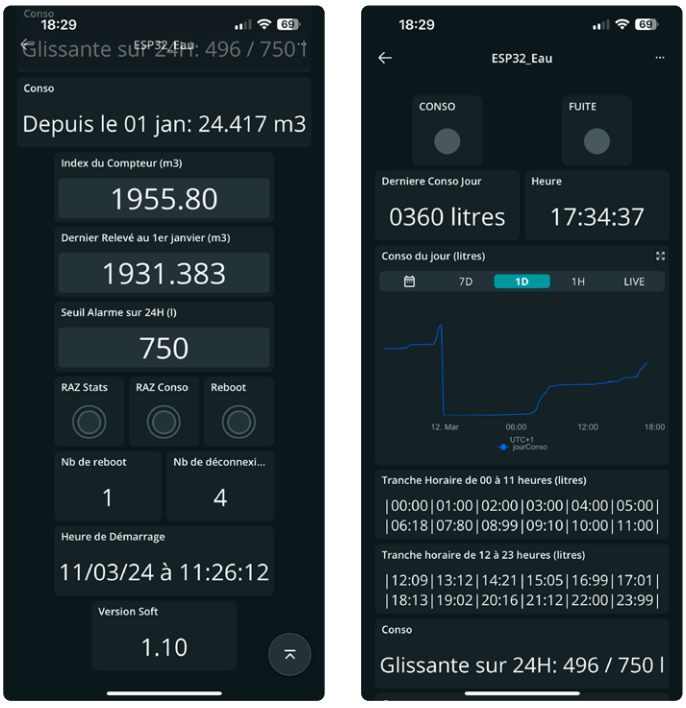


Figure 6a et b. Voici ce que je vois sur mon téléphone portable !

Le tableau de bord

La création d'un tableau de bord comme le mien est simple (**figures 6a** et **b**). Cliquez sur + *Dashboard* dans le menu principal. Vous pouvez le renommer. Appuyez ensuite sur *Add*. Sélectionnez le type de widget que vous souhaitez afficher. Il ne reste plus qu'à faire *Link Variable* pour attribuer une valeur à ce widget. Validez par *Done* et répétez cette procédure pour toutes les variables que vous souhaitez afficher sur le tableau de bord.

J'ai également assigné la variable `jourConso` à un widget graphique pour avoir un historique de la consommation d'eau.

Une fois toutes les variables assignées, sélectionnez la fonction d'édition (deux flèches croisées) pour déplacer et redimensionner vos widgets. Cliquez sur l'icône du téléphone pour obtenir une présentation différente sur votre mobile. La création d'un tableau de bord est amusante et ne prend que quelques minutes pour obtenir un résultat impressionnant.

Notifications par email et push

Là encore, rien de plus simple. Il suffit d'aller sur la page des *Triggers* dans le menu principal. Cliquez sur + *Trigger* et liez la variable `alarmLED1` (par exemple), puis choisissez d'envoyer un email ou une notification push, ou les deux.

Le circuit

J'ai réutilisé le même circuit (**figure 7)** que celui utilisé dans d'autres projets, que vous pouvez trouver sur ma page Elektor Labs [6] (ESP32 Thermostat, VMC Regulator, Fuel Measurement, Linky Analyzer and Load Shedding, NTP Server).

J'ai placé mon appareil près de mon panneau de brassage, secouru par un onduleur, mais vous pouvez également connecter une batterie au lithium rechargeable (le module Heltec dispose d'un connecteur à cet effet) qui se chargera automatiquement et maintiendra l'alimentation pendant environ 20 minutes. Comme les données sont sauvegardées périodiquement, seule la consommation d'eau des 15 dernières minutes est perdue si vous n'avez pas d'alimentation électrique de secours.

Le compteur d'eau se trouve à près de 40 m de ma maison. Par conséquent, pour garantir la fiabilité du comptage des impulsions, j'ai préféré utiliser un optocoupleur au lieu d'un filtre RC pour bloquer les interférences. Non seulement il fournit une isolation galvanique, mais il filtre

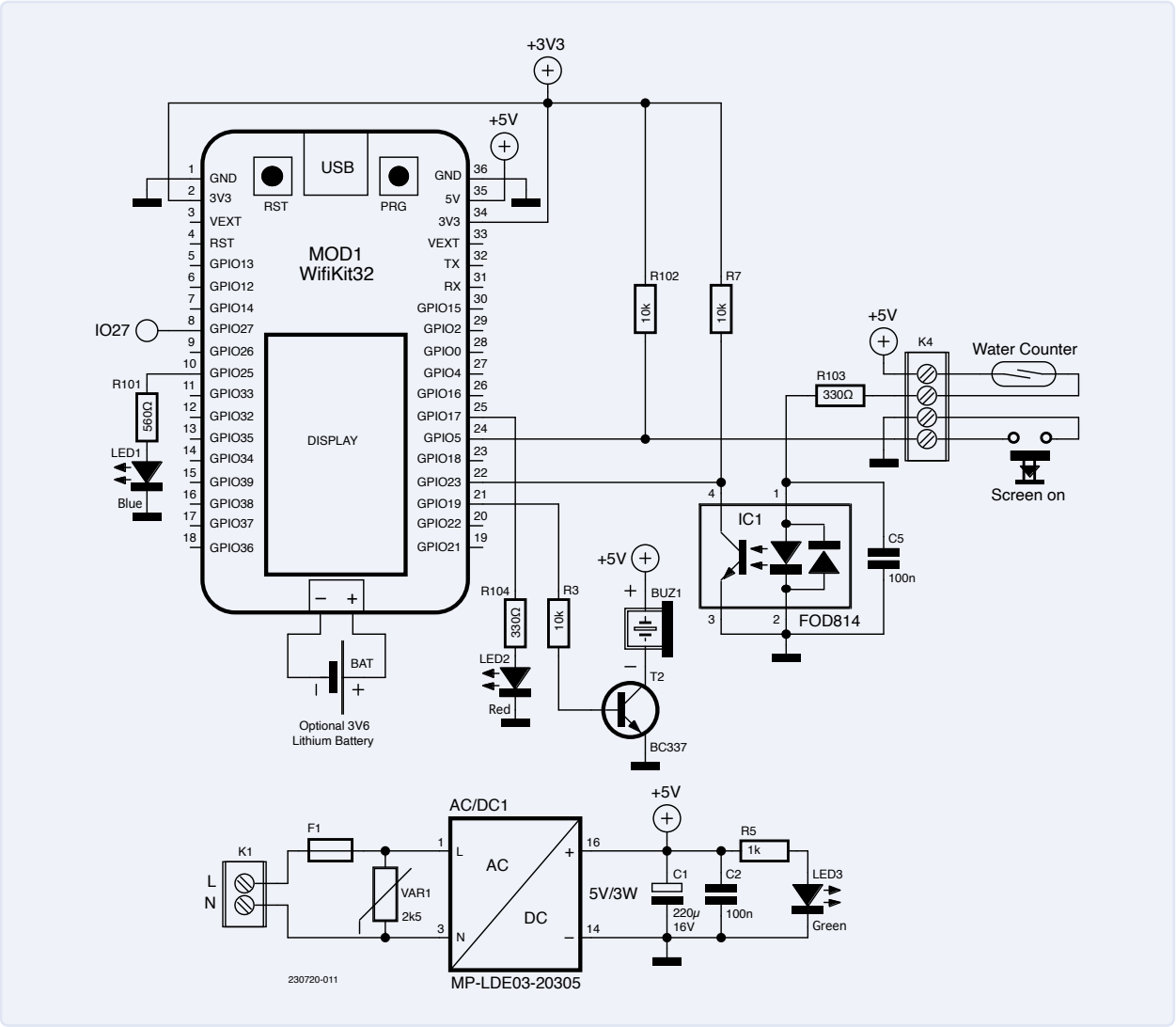


Figure 7. Le schéma du détecteur de fuites d'eau est une version adaptée du thermostat ESP32 [7]. Les résistances R101, R102 et R103 ont été ajoutées. La sortie de l'optocoupleur a été redirigée vers GPIO23.

Listage 1. Routine d'interruption.

```
/* Interrupt counting routine
 masking over 750 ms (the transmitter produces pulses of approximately 200 ms for each 0.51 of water consumed)
 at the fastest, 1 pulse every 725 ms, which corresponds to about 2.5 m3 per hour (physical limit of my counter)
 ------------------------------------------------------------*/
 void IRAM_ATTR handle Interrupt () {
   static unsigned long lastInterrupt = 0;
// debouncing of the pulse - we must wait 750 ms before processing another interrupt
   unsigned long interrupt = millis();  // start of interrupt
   if (interrupt - lastInterrupt >= 750) { // last interrupt more than 750 ms ago?
     portENTER_CRITICAL_ISR(&mux);
     indexCounter++;  // cumulative counter index, 1 pulse = 0.5 liter
     indexJour++;  // daily counter index
     portEXIT_CRITICAL_ISR (&mux);
     flagInterrupt = true;  // an interrupt has been detected
     digitalWrite(SENSOR_LED, HIGH);
// blue LED (sensor) is ON - will be put to OFF 80 ms later in the loop program
   }
   lastInterrupt = interrupt;  // reset and memorize the last interrupt to manage debouncing and blue LED
 }
```

également les impulsions parasites qui sont trop faibles pour allumer la LED de l'optocoupleur. Environ 10 mA à 3,3 V sont nécessaires pour allumer correctement la LED. Un effet secondaire positif d'un tel courant est qu'il évite l'oxydation des contacts du relais Reed. Un condensateur de 100 nF est suffisant pour filtrer la connexion au capteur Reed. Deux LED fournissent des informations sur l'état de l'appareil. La LED1 bleue, connectée à IO25 (SENSOR_LED), s'allume pendant environ 100 ms pour chaque impulsion reçue sur IO23. Lorsque la LED2 rouge, connectée à IO17 (CONNECT_LED), s'allume, cela signifie que le système n'est plus connecté à l'Arduino Cloud.

Logiciel

La principale difficulté du projet était de ne pas manquer les impulsions du compteur d'eau. C'est pourquoi une grande attention a été portée à l'écriture de la routine d'interruption (**Listage 1**).
À chaque interruption, la LED bleue s'allume. La variable globale `Last_interrupt` enregistre l'heure de l'interruption, ce qui permet à la boucle principale d'éteindre la LED environ 100 ms plus tard (flash). La variable `indexCounter` est de type `volatile int`. Elle contient le nombre d'impulsions, c'est-à-dire le nombre de demi-litres d'eau consommés. Cette variable ne peut être modifiée que dans le cadre de la routine de service d'interruption à l'aide des commandes `portENTER_CRITICAL_ISR(&mux)` et `portEXIT_CRITICAL_ISR(&mux)`.
Mon capteur envoie deux impulsions par litre. On peut le configurer dans le logiciel comme 1, 2 ou 4 impulsions par litre. En fonction des caractéristiques du capteur, la valeur est divisée par 1, 2, 4, 8, etc. (par décalage de bits) dans la boucle principale :

```
Total_Counter = indexCounter >> pulsParLitre ; // 2 impulsions / l
```

La variable `indexJour`, également un `volatile int`, correspond à la variable `jourConso` dans l'Arduino Cloud. Elle n'est pas mise à jour si la valeur est modifiée manuellement dans le tableau de bord Arduino. Comme `indexCounter`, elle est divisée par `pulsParLitre`.

Toutes les 15 minutes

- Les variables `indexCounter` et `indexJour` sont sauvegardées dans la mémoire non volatile (NVS).
- Un tableau `conso[24]` contient la consommation d'eau pour chaque heure (pour des raisons d'affichage, seuls 99 l/h peuvent être comptés). Cela permet de vérifier s'il y a une fuite : s'il n'y a pas deux périodes horaires consécutives avec une consommation à zéro, cela signifie que de l'eau coule en permanence. Dans ce cas, une alarme est déclenchée et la variable `alarmLED2` est mise à 1.
- La quantité totale d'eau consommée depuis minuit est également vérifiée. Si elle dépasse un seuil défini par l'utilisateur, il y a surconsommation. Dans ce cas, une alarme est déclenchée et la variable `alarmLED1` est mise à 1.
- En cas d'alarme, une notification push et un email sont envoyés via l'Arduino Cloud.

Toutes les 60 minutes

La consommation pour l'heure suivante est remise à zéro, et l'écran est allumé pendant une période définie par l'économiseur d'écran (5 mn, voir ci-dessous). Le petit écran OLED permet d'afficher trois pages qui défilent toutes les cinq secondes. Il permet d'afficher les caractéristiques du réseau, l'heure, les différents comptages de consommation et, le cas échéant, le type d'alarme déclenchée. On utilise la routine `formatData()` pour n'afficher la consommation sur 24 heures que sur deux lignes.
Certaines fonctions supplémentaires permettent :

- La remise à 0 de toutes les périodes horaires (normalement utilisé uniquement pendant les tests).

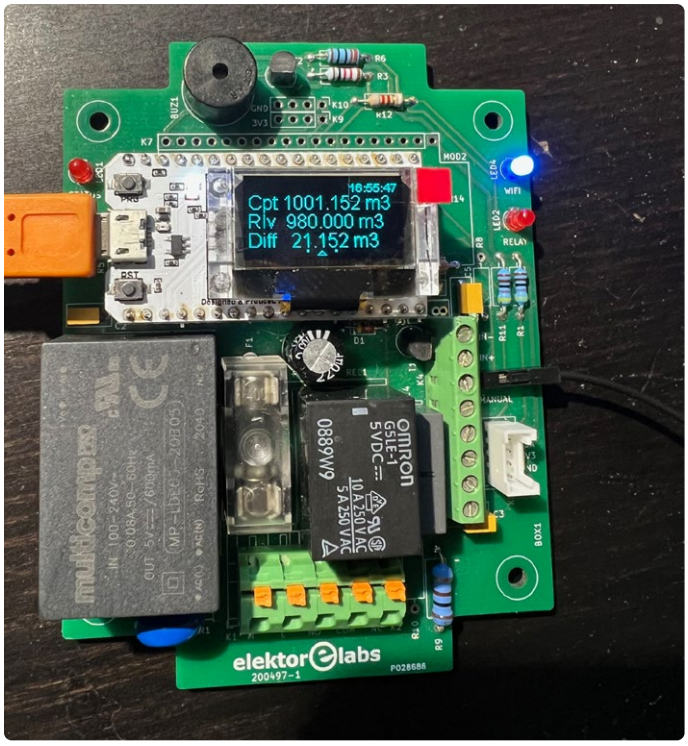


Figure 8. L'assemblage de la carte électronique portant le module Heltec ESP32.

Figure 9. La carte dans son boîtier.

- Vérifier si l'Arduino Cloud s'est déconnecté (ce qui, bien sûr, n'entraîne pas d'erreurs de comptage). Si après 600 s (10 minutes) le système ne peut toujours pas se connecter, un problème de réseau est supposé et le système redémarre. Le nombre de redémarrages peut être consulté et, à chaque redémarrage, l'heure et la date de la reconnexion sont indiquées.
- Le 31 décembre à minuit, la consommation annuelle d'eau est remise à zéro et la nouvelle valeur du compteur de consommation d'eau (`Last_Counter`) est enregistrée.

Pour vérifier que tout fonctionnait comme prévu, j'ai placé un compteur d'impulsions en parallèle et je n'ai observé aucun écart entre ce que je mesurais et ce que le compteur affichait.
Comme j'utilise un écran OLED qui a tendance à vieillir rapidement lorsqu'il est allumé en permanence, j'ai également intégré une routine d'économiseur d'écran pour que l'écran s'éteigne au bout de cinq minutes. Il se rallume en cas d'alarme ou si l'on appuie sur une touche connectée au système.

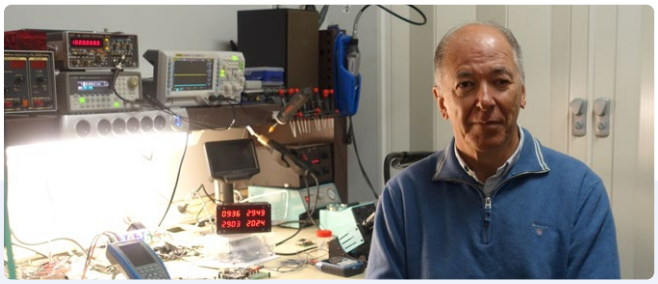
Mon installation

Pour le matériel, j'ai utilisé la carte que j'avais développée pour un thermostat ESP32 [7], voir les **figures 8** et **9**. L'avantage de cette carte est que l'alimentation secteur 230 VAC est intégrée à la carte, et que toutes les connexions sont effectuées à l'aide de borniers de 3,81 mm.

Une surveillance efficace

Le détecteur de fuites d'eau décrit dans cet article est opérationnel depuis plus d'un an. Jusqu'à présent, je n'ai pas remarqué de bogues car j'ai régulièrement mis à jour le système pour y apporter des améliorations et des corrections.
Il y a quelques mois, des toilettes situées à l'extérieur de la maison ont commencé à fuir très discrètement. Le système a détecté une fuite continue d'environ vingt litres par heure de minuit à 7 heures du matin. J'ai remarqué l'alarme à mon réveil. Sans le détecteur, cet événement aurait pu facilement se transformer en un désastre environnemental et financier.
Le système me permet également de savoir si mon adoucisseur d'eau a nettoyé sa résine à 2 heures du matin et quelle quantité d'eau a été utilisée à cet effet. De même, il peut indiquer s'il fonctionne mal. Le détecteur de fuites d'eau est devenu un atout précieux de notre système électrique et électronique domestique.

VF : Denis Lafourcade — 230720-04



À propos de l'auteur

Passionné d'électronique et de technologie depuis son plus jeune âge, Yves Bourdon a suivi des études d'ingénieur à l'INSA de Lyon. Avant même d'avoir terminé ses études, il crée sa première entreprise, ERIM, en 1981. En 1991, Yves crée le premier PC français avec un processeur Intel 286 d'un encombrement de seulement 10 cm × 10 cm. Après avoir vendu ses entreprises en 2009, Yves a consacré une grande partie de son temps à soutenir les entreprises innovantes dans leur développement. Il a notamment été président bénévole de Cap'Tronic (structure publique) pendant plus de 10 ans. Aujourd'hui âgé de 66 ans, Yves est à la retraite et passe une grande partie de son temps dans son laboratoire d'électronique à explorer les applications basées sur l'ESP32.

Questions ou commentaires ?

Envoyez un courriel à l'auteur (yb.electronique@orange.fr) ou contactez Elektor (redaction@elektor.fr).

Produits

- **Carte de développement LILYGO T-Display ESP32 (16 MB)**
  www.elektor.fr/19774
- **D. Ibrahim, *The Complete ESP32 Projects Guide*, Elektor 2019**
  www.elektor.fr/18860

LIENS

[1] D. Lafourcade, « surveillance de la consommation d'eau avec l'ESP32 », Elektor 7-8/2019 : https://www.elektormagazine.fr/magazine/elektor-101/50930
[2] S. Romero, « les projets connectés simplifiés », Elektor Guest Edition 12-2022/1-2023 : https://www.elektormagazine.fr/magazine/elektor-271
[3] Heltec WiFi Kit 32 : https://heltec.org/project/wifi-kit32-v3/
[4] Arduino Cloud : https://cloud.arduino.cc
[5] Ce projet sur Elektor Labs : https://elektormagazine.fr/labs/esp32-water-leak-detector-connected-to-iot-arduino
[6] Projets de l'auteur sur Elektor Labs : https://www.elektormagazine.fr/labs/13670/ybourdon/projects
[7] Y. Bourdon, « Thermostat connecté à ESP32 », Elektor 9-10/2021 : https://elektormagazine.fr/200497-04