connectez **votre sonnette** aux **objets sur l'internette**

avec Home Assistant et ESPHome

Ont contribué à cet article :
Idée, circuit, texte et illustrations : **Clemens Valens**
Schéma : **Kurt Diedrich**
Rédaction : **CJ Abate**
Traduction : **Hervé Moreau**
Maquette : **Giel Dols**

ESPHome et *Home Assistant* sont deux projets domotiques *open source* facilitant grandement la programmation des appareils à puces ESP8266 et ESP32. Après avoir partagé avec vous ma découverte de ces deux outils dans le numéro précédent [1], je vous invite ici à découvrir comment je les ai mis à profit pour ajouter la sonnette de notre maison à son système domotique.

Figure 1. Comment transformer une sonnette rudimentaire en système connecté complexe.

Il y a quelques années, j'ai remplacé le carillon électromécanique de notre maison par un modèle sans fil offrant différentes mélodies. Deux récepteurs en allongeaient considérablement la portée, mais l'immanquable usure de leurs piles finissait par rendre la sonnette capricieuse. En outre les visiteurs n'entendaient pas la mélodie lorsqu'ils appuyaient sur le bouton. Comme le fameux facteur, ils sonnaient toujours deux fois. Je me suis dit que l'heure de la modernité avait sonné pour cette sonnette.

Le résultat final est une sonnette connectée capable de produire un « ding-dong » mécanique, jouer différentes mélodies numériques et envoyer des notifications dans le nuage (**fig. 1**). Tout ça en même temps. C'est en quelque sorte le croisement de trois générations de sonnettes. Et cette fois-ci les visiteurs l'entendent de l'extérieur.

Circuit de l'interface de sonnette

J'ai choisi une carte NodeMCU à ESP8266 (**fig. 2**) pour construire l'interface Wi-Fi car elle offrait une entrée analogique dont j'allais avoir besoin. Comme elle se programme sans câble USB-série, ce projet ne nécessite aucun outil spécial, fer à souder excepté.

Toujours en place avec ses fils et son alimentation, le boîtier du vieux carillon électromécanique offrait assez d'espace pour y loger l'interface Wi-Fi et l'alimenter. J'ai modifié le bouton de la sonnette sans fil et l'ai relié au carillon en profitant du câblage resté en place. Comme ce bouton est à l'extérieur, j'ai protégé la connexion avec des résistances en série, un condensateur de filtrage et des diodes de niveau. Avec ses contacts ne pouvant absorber qu'une cinquantaine de milliampères, ce

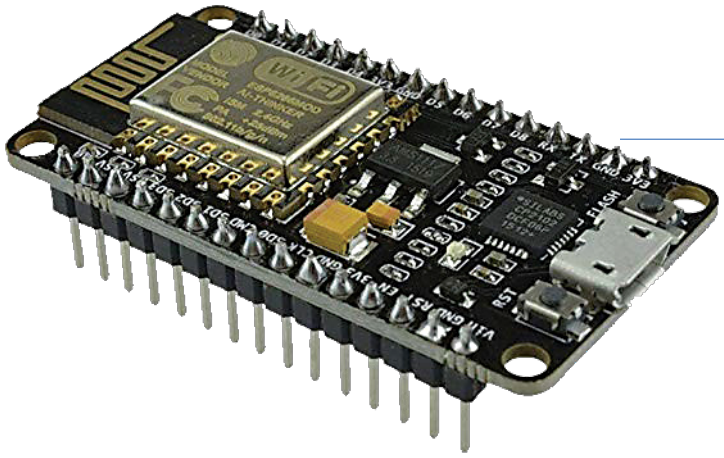

Figure 2. L'entrée analogique du module NodeMCU est flanquée d'un diviseur de tension qui divise par 3,2 la tension d'entrée.

petit bouton tactile aurait été bien incapable d'activer la bobine du carillon électromécanique puisqu'il lui aurait fallu pour cela délivrer 1 A. J'ai réglé le problème avec un petit relais de 5 V commandé par le µC via un transistor. La **figure 3** montre le schéma complet.

Pile sous surveillance

Le fil de commande du bouton de la sonnette est relié à V_{BAT} par sa résistance de rappel. Lorsque le bouton n'est pas pressé, il règne donc entre ses bornes la tension de la pile. C'est pourquoi je l'ai relié à une broche GPIO du µC, mais aussi à son entrée analogique : cela permet à *Home Assistant* de surveiller le niveau de la pile et de relayer les messages de la sonnette. C'est pratique, car la portée de la sonnette sans fil est proportionnelle à la tension de la pile, et le ou les récepteurs ne peuvent plus être atteints lorsque cette tension est trop basse.

Notez qu'il serait bien sûr possible d'utiliser l'entrée analogique pour détecter les pressions sur le bouton et surveiller le niveau de la pile, mais cela compliquerait l'écriture du micrologiciel. L'exploitation d'une broche GPIO libre évite cette complication.

Alimentation

Le carillon électromagnétique n'offrait qu'un transformateur de 12 V alors que le relais et la carte NodeMCU attendaient 5 V CC. Un petit aménagement s'imposait. Comme la bobine du carillon charge lourdement le transformateur, celui-ci voit sa tension de sortie

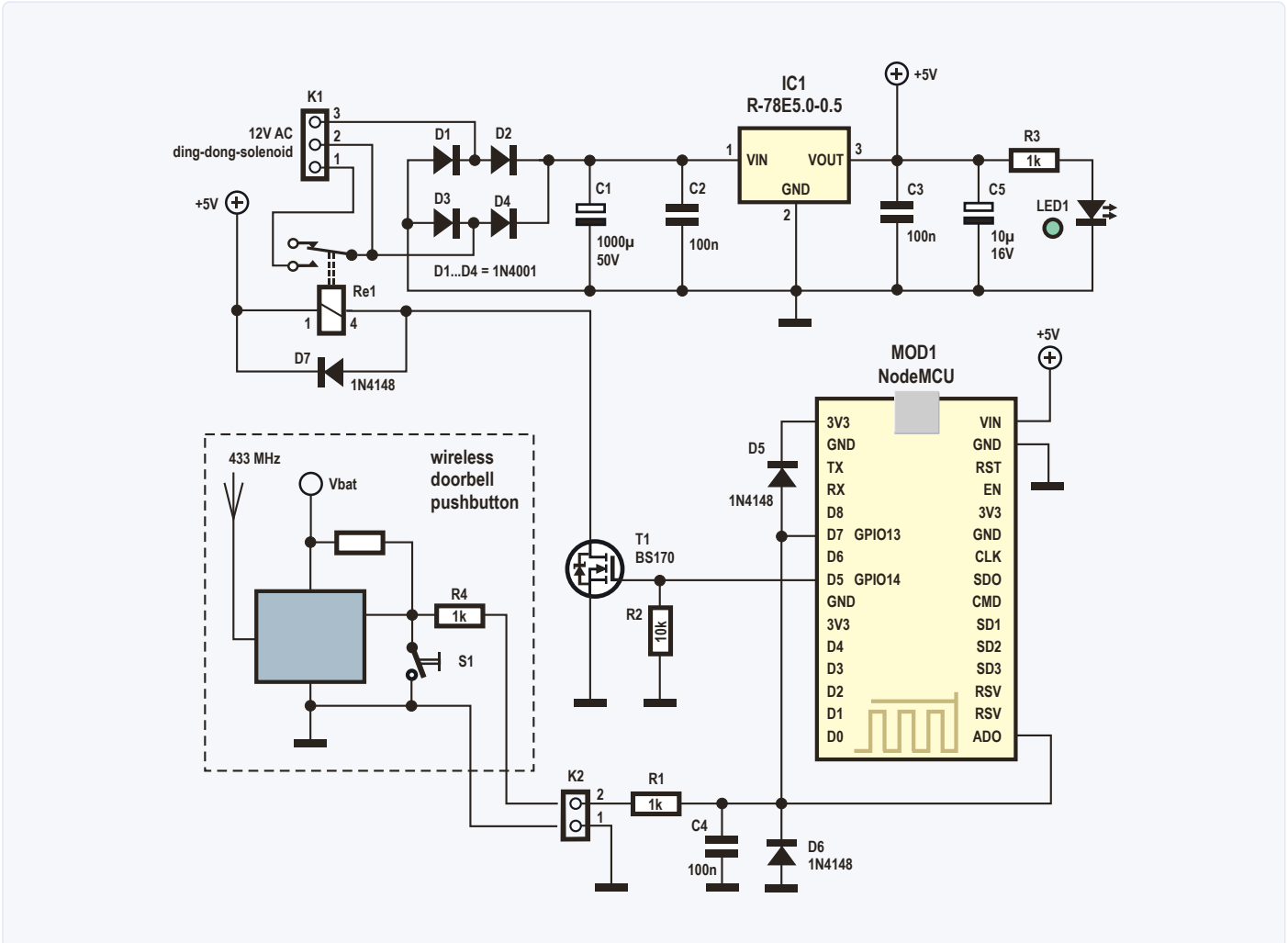

Figure 3. L'interface a besoin d'une alimentation régulée. Les broches reliées au bouton de la sonnette sont protégées des aléas naturels par une circuiterie particulière.

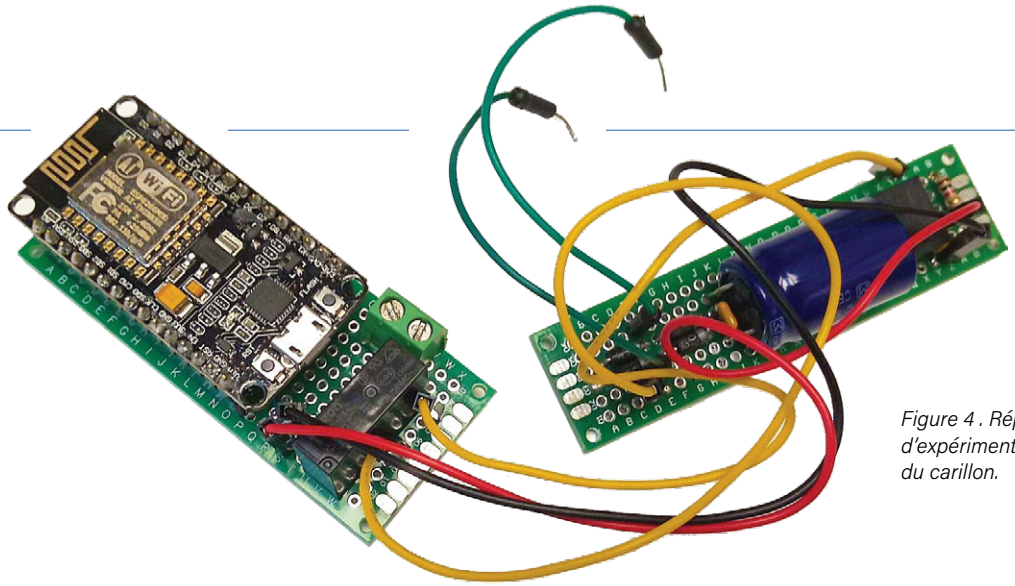

Figure 4 . Répartir l'interface sur deux platines d'expérimentation a permis de la faire tenir dans le boîtier du carillon.

BREF RAPPEL

ESPHome est un micrologiciel open source pour dispositifs à ESP8266 ou ESP32 : prises intelligentes, LED, capteurs, etc.
https://esphome.io/

Home Assistant est un contrôleur domotique open source qui relie des dispositifs connectés (capteurs, prises intelligentes, etc.) à des actionneurs ou des services, et permet de commander l'ensemble au moyen de règles complexes.
https://www.home-assistant.io/

Pour plus de détails, voir [1].

baisser chaque fois que la bobine est activée. Un condensateur de grande capacité filtre ces baisses indésirables. Sans lui, le module NodeMCU redémarrerait à chaque coup de sonnette.

Pour éviter les problèmes de dissipation de chaleur, j'ai choisi comme régulateur de 5 V un petit régulateur à découpage de type 7805. J'ai par ailleurs ajouté un témoin d'alimentation car la carte NodeMCU en est dépourvue et qu'une telle LED est toujours pratique.
J'ai réparti le circuit de l'interface sur deux plaques d'essai – une pour l'alimentation, l'autre pour le reste – et les ai nichées dans le boîtier de l'ancien carillon (**fig. 4**).

Configuration du micrologiciel ESPHome

Je me suis servi de l'outil *ESPHome* pour créer l'interface de ma sonnette. Avec *ESPHome*, une demi-heure de programmation suffit pour mettre en service un périphérique intelligent doté de sa propre interface web et d'une fonction de programmation à distance (*Over-The-Air*, OTA). Et quand je dis « programmation », il s'agit en fait d'écrire un fichier de configuration avec le langage de balisage YAML. Le périphérique sera en outre pris en charge par l'extension *Home Assistant* et pourra être modifié à tout instant depuis un smartphone, une tablette ou un ordinateur.

ESPHome prend en entrée un fichier de configuration (ou script) YAML, le traduit en code C/C++ et le compile avec le reste de la bibliothèque. Le script spécifie les éléments d'*ESPHome* à inclure (paramètres Wi-Fi, interface web ou non, ce genre de choses), quels types de capteurs et d'actionneurs sont reliés aux broches du µC, et que faire avec. La plupart des choses communes sont prédéfinies, d'où la facilité et la rapidité de la mise en route. YAML est un langage simple. Ses règles d'indentation sont strictes, mais on finit par s'y faire.

Association du relais au bouton

Le relais qui commande la bobine du carillon est défini en tant que `switch` interne relié à GPIO14. Mettre sa clé `internal` à `true` le rend invisible à *Home Assistant*, ce qui évite d'encombrer le système.

```
switch:
  - platform: gpio
    pin: GPIO14
    id: ding_dong_relay
    internal: true
```

Le bouton est un `binary_sensor` sur la broche GPIO13. La valeur `true` de sa clé `inverted` indique à *ESPHome* qu'il est actif au niveau bas.

```
binary_sensor:
  - platform: gpio
    name: "my doorbell pushbutton"
    pin:
      number: GPIO13
      inverted: true
    on_press:
      then:
          - switch.turn_on:
ding_dong_relay
        - delay: 200ms
          - switch.turn_off:
ding_dong_relay
```

ESPHome crée un évènement `on_press` lorsque le bouton est pressé, puis l'envoie à *Home Assistant*. Un tel évènement peut servir de signal dans le fichier YAML pour déclencher une ou plusieurs actions. Appelé *automation* dans la documentation, ce mécanisme est l'une des forces d'ESPHome.

Votre avis, s'il vous plaît…
Pour toute question ou commentaire relatifs à cet article, contactez l'auteur (clemens.valens@elektor.com) ou la rédaction (redaction@elektor.fr).

Ci-dessus j'ai utilisé l'évènement `on_press` pour produire une impulsion de 200 ms sur la sortie du relais `ding_dong_relay`. Ainsi ma sonnette actionnera-t-elle systématiquement le carillon électromécanique, même en cas de coupure de courant. C'est ce qu'elle faisait il y a 40 ans, mais à l'ancienne, par électricité pour ainsi dire. Aujourd'hui elle est entourée d'une mégalopole de transistors.

Surveillance de la pile

La pile est définie en tant que `sensor` sur A0. Son type (`platform`) est `adc` puisque sa tension doit être convertie en valeur numérique. Comme le module NodeMCU divise par 3,2 la valeur lue sur A0, il nous faut la multiplier par 3,2 pour retrouver la valeur originale. C'est ce que fait le filtre `multiply` ci-dessous. Un filtre est un opérateur modifiant la valeur de sortie d'un capteur.

```
sensor:
  - platform: adc
    name: "doorbell battery level"
    pin: A0
    filters:
    - multiply: 3.2
```

PROGRAMMER ESPHOME SUR UN DISPOSITIF VIERGE

Tout nouveau dispositif à ESP8266 ou ESP32 doit d'abord être programmé par le port série avant de devenir compatible avec *ESPHome*. Pour cela il vous faudra peut-être installer un pilote USB-série sur l'ordinateur exécutant *Home Assistant*. La nécessité ou non de ce pilote dépend du dispositif. Les cartes NodeMCU à puce CP2102 USB de Silabs (*Silicon Laboratories*) que j'ai programmées depuis un Raspberry Pi n'en ont pas eu besoin.

Reliez votre dispositif à l'ordinateur exécutant Home Assistant, puis ouvrez le tableau de bord d'*ESPHome* depuis la barre latérale (si vous avez activé cette option). Vous pouvez aussi l'ouvrir depuis le panneau *Supervisor* en sélectionnant la carte de l'add-on *ESPHome* et en cliquant sur *Open Web UI*. Ouvrez la liste déroulante indiquant *OTA (Over-The-Air)* par défaut. Vérifiez que le port série est disponible. S'il ne l'est pas, redémarrez l'add-on *ESPHome* (en revenant au panneau *Supervisor*).

Cliquez ensuite sur le bouton rosâtre + du tableau de bord d'*ESPHome* et laissez-vous guider par l'assistant. C'est probablement stupide de ma part, mais je n'ai utilisé aucun mot de passe pour mes dispositifs…

Pour le chargement du micrologiciel, sélectionnez le port auquel le dispositif est connecté.

Exploitation de la LED embarquée

La carte NodeMCU est équipée d'un bouton *Flash* relié à GPIO0 et d'une LED reliée à GPIO16. Le premier peut p. ex. servir de bouton de test ; pour cela j'ai défini GPIO0 dans la section `binary_sensor` comme composant de type `binary`. Je me suis servi de la LED pour indiquer qu'un visiteur vient d'appuyer sur la sonnette et que *Home Assistant* a détecté l'évènement.

```
light:
  - platform: binary
    name: "LED1"
    output: led1

output:
  - platform: gpio
    id: led1
```

DANS L'E-CHOPPE D'ELEKTOR

- **Carte NodeMCU à microcontrôleur ESP8266**
  www.elektor.fr/nodemcu-microcontroller-board-with-esp8266-and-lua
- **Livre** *IoT Home Hacks with ESP8266*
  www.elektor.fr/iot-home-hacks-with-esp8266

```
pin:
  number: GPIO16
  inverted: true
```

J'ai défini un composant `light` relié à une sortie appelée `led1`. Cette définition crée une entité « light » dans *Home Assistant* qui offre d'autres possibilités que celles associées à un composant « switch » (et qui a une icône différente). La LED reliée à GPIO16 est définie en tant que `output` avec une clé `id` appelée `led1`. Sa clé `inverted` vaut `true` car elle est active au niveau bas.
Avec ces définitions, tout message envoyé par *Home Assistant* au composant `LED1` pour l'allumer ou l'éteindre sera transféré à la sortie correspondante, soit `led1`.
Le fichier de configuration YAML complet est à télécharger depuis [2]. N'oubliez pas de compléter les clés `ssid` et `password` de votre réseau Wi-Fi, et éventuellement de changer le nom de la sonnette.

Extension ESPHome pour Home Assistant

L'extension *ESPHome* pour *Home Assistant* permet d'écrire le script YAML d'un dispositif *ESPHome* directement depuis le logiciel *Home Assistant*, de le compiler, de le charger dans le µC, et aussi de le mettre à jour à distance. Un système ESP8266 vierge doit toutefois être d'abord programmé par le port série car la programmation OTA n'a pas encore été activée (cf. **encadré**).

Automatisation depuis Home Assistant

Voyons maintenant comment associer une règle d'automatisation à la LED reliée à GPIO16 de façon à ce qu'elle s'allume lorsqu'un visiteur appuie sur la sonnette.
La règle pour cet automate est fort simple : déclencher l'action « mettre la LED sous tension » lorsque survient l'évènement « appui sur un bouton ».
L'éditeur d'automates de *Home Assistant* peut être ouvert de différentes façons. L'une d'elles est de cliquer sur *Configuration* dans le menu de gauche, puis de cliquer sur *Automations*.
On crée un nouvel automate en cliquant sur le bouton rond orange « + » situé dans le coin inférieur droit.
Il est possible d'entrer une instruction écrite et de laisser *Home Assistant* tenter de la convertir en code, mais trouver la bonne phrase n'est pas facile. La règle créée risque d'être boiteuse, et la corriger prend du temps. Bref, mieux vaut ici cliquer sur *Skip*.
Entrez un nom pour l'automate et, en option, une description. Laisser le mode sur *Single (default)*.

Déclencheurs et actions

Là aussi il existe plusieurs façons de spécifier un évènement déclenchant l'automate. Selon moi la plus intuitive est de choisir un *Device* comme type de déclencheur (*Trigger type*). Dans la liste *Device*, cherchez le nom que vous avez attribué à votre dispositif dans le fichier YAML. Cherchez de même le nom de votre bouton dans la liste *Trigger*.
Il n'y a pas de conditions ici, donc passez cette partie.
Les actions sont semblables aux déclencheurs, et là encore il y a plus d'une façon d'obtenir le résultat souhaité. L'une d'elles est de choisir *Device* comme type d'action (*Action type)* et de sélectionner ce dispositif dans la liste *Device*. Dans la liste *Action*, sélectionnez *Turn on LED1* (si vous aviez choisi un autre nom pour LED1, utilisez-le).
Enregistrez votre automate en cliquant sur l'icône orange située en bas à droite.
Vérifiez que l'automate fonctionne en appuyant sur la sonnette : la LED rouge s'allume (ou en tout cas elle le devrait) mais, petit problème, elle reste allumée…

Ajout d'une carte Light

Corrigeons cela en ajoutant une « carte » de type Light au tableau de bord de *Home Assistant*. Cette carte représentera la LED et permettra de l'éteindre (en cliquant dessus) et d'en connaître l'état à distance.
Dans le menu de gauche, cliquez sur *Overview*. Cliquez ensuite sur les trois points du coin supérieur droit (*Open Lovelace UI menu*) et sélectionnez *Configure UI*. Cliquez sur le bouton orange « + » du coin inférieur droit. Cherchez la carte *Light* et cliquez dessus. Cherchez maintenant la liste *Entity* pour la LED et sélectionnez-la (oui, je sais, on se balade beaucoup à dos de souris dans *Home Assistant*). Affectez-lui un nom, cela facilitera son identification dans l'interface. Car même si cette LED est pour l'instant la seule de son espèce, d'autres viendront s'ajouter à votre système domotique à mesure que celui-ci évoluera. Un nom bien choisi permet de repérer rapidement qui est qui et qui fait quoi.
Enregistrez votre carte, et quittez le mode de configuration en cliquant sur la croix du coin supérieur gauche (*Close*).
Cliquez sur la carte pour changer l'état de la LED.

Sonnez et entrez dans le monde de l'IdO

ESPhome et *Home Assistant* évoluant continuellement, il est possible qu'une nouvelle version parue entre-temps vous oblige à adapter l'une ou l'autre des étapes décrites ici. Quoi qu'il en soit vous devriez être en mesure d'assembler votre propre sonnette Wi-Fi connectée et d'automatiser ses réponses depuis *Home Assistant*. Vous connaissez les principes de base, à vous de franchir cette première marche pour créer une sonnette adaptée à vos besoins.

200089-02

LIENS

[1] **La domotique, c'est facile avec… :**
www.elektormagazine.fr/magazine/elektor-156/58991

[2] **How-To: Integrate Your Doorbell in Home Assistant Using ESPHome :**
www.elektormagazine.com/labs/esphome-doorbell