interrupteur crépusculaire DeLux

Une solution pour une commutation de haute précision, commandée par la lumière

Clemens Valens (Elektor)

On trouve de nombreux interrupteurs crépusculaires pour environ 10 €, mais pour la plupart le changement d'état s'effectue quelque part à la tombée du jour. Parfois, les applications exigent plus de précision et de maîtrise que ne le permettent ces interrupteurs bon marché.
Vous avez besoin d'un contrôle de luminosité précis au lux près ? Si oui, ce projet est pour vous.

Il existe de très nombreux modèles d'interrupteurs crépusculaires et la plupart fonctionnent parfaitement dans l'application pour laquelle ils ont été conçus. Il s'agit généralement d'allumer un éclairage lorsque le niveau de luminosité ambiante descend en dessous d'un certain seuil et de l'éteindre à nouveau lorsque le niveau de luminosité augmente. On y ajoute aussi parfois une minuterie.
Vous pourriez penser qu'il est possible de couvrir toutes les applications avec ces modèles, mais ce n'est pas le cas. La raison en est qu'ils manquent tous de précision. Basés sur une LDR (photorésistance) ou un phototransistor, ils ont tendance à commuter quelque part dans la zone crépusculaire. Or, les niveaux de luminosité varient bien plus que cela.

La luminosité est subjective

Pour les humains, l'intensité ou la luminosité de la lumière du jour est assez constante. Bien sûr, nous remarquons les variations dues aux nuages et au soleil, mais nous n'y sommes pas très sensibles. Cela s'explique par la réponse logarithmique de l'œil à la luminosité. Par temps nuageux, la luminosité peut varier entre 5 000 et 10 000 lux, alors qu'elle nous semble presque identique. Au soleil, la lumière peut atteindre des niveaux supérieurs à 25 000 lux, ce que nous percevons évidemment, mais sans réaliser que la luminosité est au moins trois fois plus élevée. Les plantes, quant à elles, sont beaucoup plus sensibles à l'intensité lumineuse que les humains. Les agriculteurs le savent, et ils éclairent artificiellement certaines de leurs cultures, même pendant la journée, pour améliorer leur rendement. Par temps ensoleillé, cela n'est généralement pas nécessaire, mais en cas de nuages, cela peut être utile. Pour faire cela de manière économique, ils ont donc besoin de commutateurs commandés par la lumière capables de détecter les différences de luminosité avec plus de précision qu'une LDR ou un phototransistor.
Il existe aujourd'hui des capteurs lumineux qui convertissent directement la luminosité en une valeur en lux avec des résolutions allant jusqu'à 16 bits. Certains de ces capteurs ne mesurent pas seulement les lux, mais aussi l'intensité des rayons UV et de la lumière blanche. Avec un tel

capteur, il est assez facile de construire un interrupteur de haute précision commandé par la lumière.

Capteur de lumière ambiante de haute précision

Un capteur de lumière courant est le VEML7700 de Vishay. Il s'agit d'un capteur de lumière ambiante de haute précision avec interface I²C que l'on peut trouver monté sur un petit module pour quelques euros. De la même famille, on peut également citer le VEML6075, un capteur de lumière UVA et UVB également avec interface I²C.

Grâce à son interface numérique I²C, le capteur ne nécessite pas de convertisseur analogique-numérique et peut être connecté directement à la plupart des microcontrôleurs. Ses données de sortie sont disponibles dans deux registres de 16 bits : lumière ambiante (également appelée « ALS », « Ambient Light Sensor ») et lumière blanche. La lumière blanche couvre un large spectre allant de 250 nm à 950 nm. Le spectre ALS est beaucoup plus étroit, d'environ 450 nm à 650 nm, car il est optimisé pour la perception humaine. Les plantes n'étant pas des êtres humains, la sortie à utiliser dépendra de l'application.

La sensibilité du capteur est également importante. Le VEML7700 a une résolution de 0,005 lx par bit et un niveau de détection maximal de 167 000 lx (0,01 lx pour le minimum). Une gamme aussi étendue nécessiterait des valeurs sur 25 bits, alors que le composant ne délivre que 16 bits ; c'est pourquoi on peut spécifier une valeur de sensibilité (parfois appelée « gain ») pour ramener les choses dans la gamme. Sa haute sensibilité permet d'utiliser le capteur derrière des surfaces à faible transmittance (c.-à-d. sombres) et d'obtenir néanmoins de bons résultats.

Pour les conditions de faible luminosité, le capteur dispose d'un paramètre d'intégration allant jusqu'à 800 ms. Enfin, il est possible de définir des seuils bas et hauts qui peuvent déclencher des interruptions, ce qui facilite la création d'alarmes ou permet une commutation automatique.

Associons-le à un ESP32

Le module VEML7700 choisi pour mes expériences a été acheté chez Adafruit. Une bonne plateforme pour l'utiliser est le thermostat connecté ESP32 [1] (alias Automator, voir [2]). Le module possède cinq broches, mais comme il dispose de deux options d'alimentation, nous n'en avons besoin que de quatre. Ces quatre broches ont le même ordre que sur le connecteur de l'écran OLED du thermostat, et nous pouvons donc brancher le module sur K9. Assurez-vous que la broche VIN ne soit pas connectée et que le module soit orienté vers le haut (contrairement à la façon dont l'écran OLED serait branché, voir **figure 1**).

Le logiciel avec ESPHome

Après avoir connecté le capteur à l'ESP32, nous devons ajouter du logiciel pour que tout fonctionne. Comme l'objectif est une sorte d'interrupteur commandé par la lumière, un choix logique serait d'utiliser une plateforme domotique et ma préférée est ESPHome. Elektor a publié plusieurs projets utilisant ESPHome [3][4], et donc vous savez probablement comment l'utiliser et le configurer, mais ce projet introduit un concept que je n'avais pas encore traité : la création d'un capteur personnalisé. Si vous ne connaissez pas ESPHome, je vous recommande de lire et de regarder d'abord [3] et [4].

Il nous faut un capteur personnalisé

ESPHome prend en charge de nombreux capteurs, mais (au moment de la rédaction de cet article) pas le VEML7700. Il connaît d'autres capteurs de lux, mais pas celui-ci. Cependant ce n'est pas un problème, car ESPHome propose une méthode pour ajouter votre propre capteur. Pour cela il faut écrire du code C++, ce qui rend les choses un peu plus complexes.

Comme précédemment (voir [3] et [4]), nous devons déclarer une section *sensor* dans le fichier de configuration de ESPHome pour ce capteur. On doit maintenant préciser *custom* dans le champ *platform* du capteur. Cela indique à ESPHome que vous allez lui fournir tous les détails.

Suit une section *lambda* qui consiste en quelques lignes de code C++ pour indiquer à ESPHome d'inscrire un capteur de notre conception (que nous avons nommé *veml7700*). Nous devons également spécifier le(s) flux de données que notre capteur produit. Il a trois sorties ici : ALS, lux et lumière blanche. L'ordre est important et doit être respecté chaque fois qu'elles sont référencées ailleurs dans le fichier YAML.

Nous continuons avec des déclarations YAML normales pour spécifier davantage les sorties des capteurs. Cela se fait avec une section *sensors* (au pluriel !) où nous pouvons spécifier pour chaque flux de données son nom, ses unités et le nombre de décimales à utiliser. L'ordre est le même que dans l'instruction *return* de la section *lambda*.

Seul le flux de lux a des unités (« lx »), et aucun flux ne nécessite de décimales, donc nous les mettons à zéro.

La dernière chose à faire est d'indiquer à ESPHome où trouver le pilote pour le capteur personnalisé. Nous le faisons dans la section *esphome* en haut du fichier de configuration où nous ajoutons un fichier (veml7700.h) à la sous-section *includes*. Sur

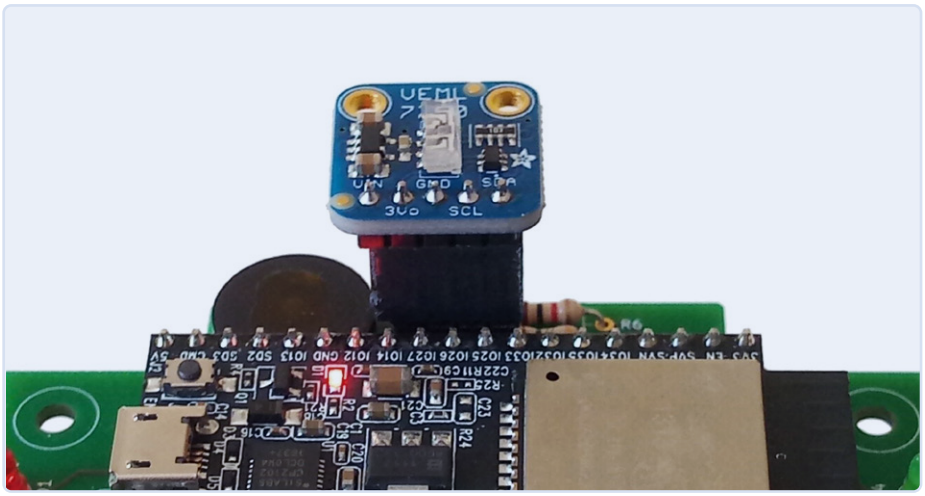


Figure 1. Le module VEML7700 à 5 broches se branche sur le connecteur à 4 voies K9. Sa broche VIN n'est pas connectée, même si elle en a l'air.

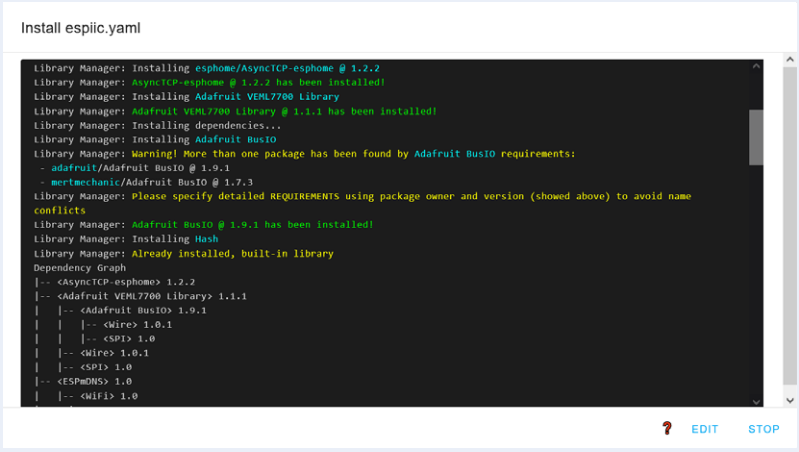


Figure 2. ESPHome installe automatiquement la bibliothèque Adafruit.

le système qui exécute ESPHome, ce fichier doit se trouver dans le même dossier que le fichier YAML du périphérique.

La partie difficile

C'est fini ? Pas vraiment. Vient maintenant la partie la plus difficile, qui consiste, bien sûr, à créer le pilote pour le capteur personnalisé. Ce pilote doit être conforme aux normes ESPHome pour les composants (un capteur est un composant), c'est-à-dire qu'il doit fournir certaines fonctions qu'ESPHome attend de ses composants, et il doit communiquer avec le capteur lui-même.

Pour la deuxième partie, nous pouvons compter sur Adafruit qui, en plus de fabriquer le module VEML7700, a également créé une bibliothèque Arduino pour celui-ci [5]. Notre pilote doit fournir l'interface entre ESPHome et la bibliothèque Arduino.

Installation d'une bibliothèque tierce

ESPHome fournit un mécanisme pour ajouter des bibliothèques communautaires *open source* : il suffit d'ajouter le nom exact (!) de la bibliothèque à la sous-section *libraries* de la section *esphome*. Dans notre cas, il s'agit de « Adafruit VEML7700 Library ». Maintenant, lorsque ESPHome traitera le fichier de configuration, il installera d'abord la bibliothèque (s'il ne l'a pas déjà fait, voir **figure 2**) avant de tout compiler. Dans votre pilote personnalisé, il suffit d'inclure la bibliothèque comme n'importe quelle autre.

Je ne me suis pas aventuré plus loin sur ce terrain, donc je ne connais pas les critères qu'une bibliothèque tierce doit remplir pour être utilisée de cette façon. Pour plus de détails, consultez la documentation de platformIO, la chaîne d'outils utilisée par ESPHome.

Un croquis Arduino encapsulé

Notre pilote est une classe C++ qui doit fournir au moins un constructeur et une fonction nommée *update*. Nous avons également besoin d'une fonction *setup* afin d'initialiser le pilote Adafruit. Les fonctions *setup* et *update* de notre classe font ce qui serait normalement fait dans les fonctions *setup* et *loop* d'un croquis Arduino typique utilisant le pilote d'Adafruit. En fait, notre classe encapsule un croquis Arduino incluant des variables globales et y ajoute des éléments spécifiques à ESPHome. Pour ESPHome, nous devons insérer des appels à *publish_state* à la fonction *update* pour chaque flux de données (ALS, lux et lumière blanche). Cela permettra de mettre les données à la disposition du reste du monde. L'ordre des appels doit être le même que dans l'instruction de retour de la sous-section *lambda* de la section *sensors* (voir ci-dessus).

Comme notre classe hérite de la classe *PollingComponent* qui elle-même hérite de la classe *Component*, vous disposez d'autres fonctions qui peuvent vous servir. Un composant *polling* est un composant qui est appelé périodiquement, et la fréquence d'appel peut être spécifiée (par ex. dans notre constructeur). Reportez-vous à [6] pour plus de détails.

Ajout du bus I²C

Il ne reste plus maintenant qu'à connecter le capteur au bus I²C au sein du logiciel (c.-à-d. créer une connexion logique entre les deux, puisque nous avons déjà une connexion physique). La bibliothèque Adafruit utilise à cet effet la bibliothèque *Wire* d'Arduino. Dans le fichier YAML de ESPHome, nous ajoutons donc une section *i2c* pour que ESPHome sache qu'elle est nécessaire.

L'ESP32 possède deux bus I²C avec des signaux SDA et SCL qui peuvent être connectés à presque toutes les broches de la puce. ESPHome prend donc en charge plusieurs bus I²C avec des broches librement assignables, ce qui permet de spécifier où va quoi. Mais la bibliothèque Arduino Wire ne supporte qu'un seul bus I²C, alors comment lui dire d'utiliser le bus que vous voulez ? Eh bien, vous ne pouvez pas, car il utilisera toujours le bus par défaut. Dans ESPHome, le bus I²C par défaut est le premier bus spécifié dans la section *i2c*. Il aurait été bien de pouvoir spécifier un bus I²C en utilisant son ID, et ESPHome vous aurait volontiers permis de le faire, mais la bibliothèque Arduino Wire sous-jacente ne le permet pas.

C'est fini

C'est ici que se termine cet article. Si l'Automator est programmé par ESPHome avec le code décrit ci-dessus, vous obtiendrez un dispositif qui peut être autonome ou commandé par un assistant domotique comme Home Assistant. Il n'y a pas de règles d'automatisation dans le fichier YAML d'ESPHome, donc sans assistant, il se contentera de mesurer l'intensité de la lumière ambiante. Les règles d'automatisation peuvent être ajoutées au fichier YAML lui-même, ou créées dans Home Assistant, par exemple. Pour une description du reste du code YAML, qui n'a rien de particulier, référez-vous à [3] et [4]. Le fichier de configuration YAML et le code C++ peuvent être téléchargés depuis la page web de cet article [7].

210190-04

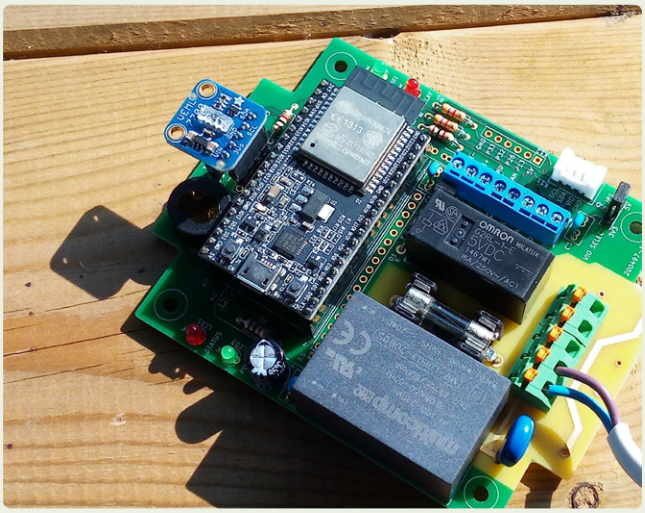
Contributeurs

Conception, texte et photographies : **Clemens Valens**
Rédaction : **Jens Nickel, C. J. Abate**
Mise en page : **Harmen Heida**
Traduction : **Denis Lafourcade**

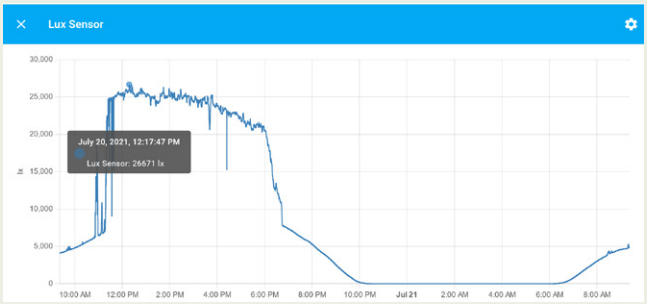
Des questions, des commentaires ?

Envoyez un courriel à l'auteur (clemens.valens@elektor.com) ou contactez Elektor (redaction@elektor.fr).

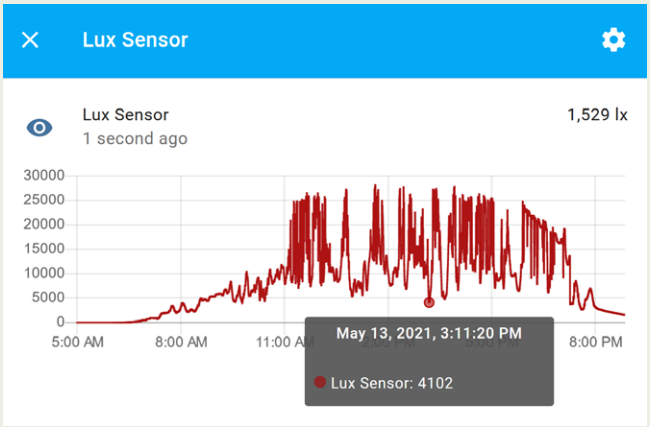
Quelques résultats



Le capteur de luminosité placé en plein soleil d'été. À midi, avec une sensibilité réglée sur 0,125 et un temps d'intégration de 100 ms, le capteur a indiqué des valeurs d'environ 48 000 pour l'ALS et d'un peu plus de 30 000 pour la lumière blanche. Ces valeurs correspondent à des intensités lumineuses d'environ 22 000 lx. À 10h30, dans les mêmes conditions, j'ai mesuré 16 800 lx, soit une différence de 5 200 lx. Cependant, à mes yeux, la luminosité était à peu près la même dans les deux situations.



Évolution de la luminosité par une journée ensoleillée de juillet. Le « rebond » sur la gauche est dû à l'ombre d'un arbre. De midi à 18h, la luminosité diminue lentement, mais on ne le remarque pas vraiment. Les creux sont causés par les nuages. L'ombre de la maison commence à couvrir le capteur à 18h, ce qui explique la forte baisse. Ensuite, l'intensité lumineuse continue de diminuer jusqu'à la nuit. Pour les humains, 5 000 lx est déjà une forte luminosité..



Par un jour partiellement nuageux en mai, les niveaux de luminosité peuvent varier entre 4 000 et 27 000 lx environ.

Tous les graphiques ont été créés à l'aide de Home Assistant à une latitude de 47°N.

PRODUITS

- **ESP32 DevKitC**
  www.elektor.fr/18701
- **Écran OLED, I²C, 128×64 de 2,5 cm**
  www.elektor.fr/18747
- **Livre en anglais, « Getting Started with ESPHome », Koen Vervloesem**
  www.elektor.fr/19738

LIENS

[1] « Thermostat connecté à ESP32 », Y. Bourdon, Elektor 09-10/2021 : www.elektormagazine.fr/200497-04
[2] Elektor Automator : www.elektormagazine.fr/labs/automator
[3] Guide Home Assistant et ESPHome (en anglais) : www.elektormagazine.fr/labs/how-to-home-assistant-esphome
[4] « la domotique, c'est facile avec... », C. Valens, Elektor 09-10/2020 : www.elektormagazine.fr/200019-02
[5] Bibliothèque Adafruit VEML7700 : https://github.com/adafruit/Adafruit_VEML7700
[6] ESPHome sur GitHub : https://github.com/esphome
[7] Téléchargements pour cet article : www.elektormagazine.fr/210190-04