ISSN 0181-7450 F : 15,50 €

(B) 16,50 €
(CH) 29,00 FS
(CAN) 20.99 $ca
(And) 15,50 €
DOM surface 16,50 €
DOM avion 16,90 €
N Cal/S 2000 cfp

elektor 5H

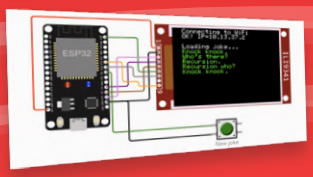
créer > partager > vendre

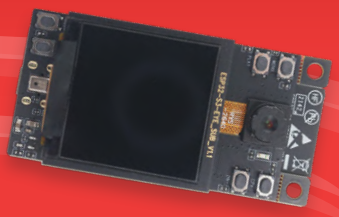
DÉCEMBRE 2023 & JANVIER 2024
ELEKTORMAGAZINE.FR

ÉDITION SPÉCIALE

140 Pages

Rédaction invitée
pour ce numéro

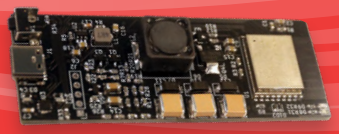




ESPRESSIF

prototypage avec **les puces Espressif**

ADF, IDF et autres **SDK**



conseils et astuces
des **ingénieurs d'Espressif**

ESP32 et **ChatGPT**

automatisation
avec Rainmaker
et Matter

essai de
l'ESP32-S3-BOX-3

dans ce numéro

- un serveur de reconnaissance de la parole open-source
- deux outils puissants : Rust et les systèmes embarqués
- reconnaissance faciale avec l'ESP32-S3-EYE
- talkie-walkie avec ESP-NOW
- arbre de Noël circulaire

et bien plus encore !

L 16651 - 5H - F: 15,50 € - RD

INELTEK

Design-In Expertise & Service

Your Partner for ESPRESSIF

- **Ineltek** — Leading Espressif franchise distribution partner
- **Experience** — 36 years of experience in the semiconductor industry
- **Latest News** — Ineltek is always up to date regarding Espressif's innovations – Contact us!
- **Innovation** — Features are missing? We address your requests for next product generations
- **Application Support** — Dedicated and focused in-house support offering
- **Time to Market** — Espressif & Ineltek's FAE team are in close & regular contact to speed up your designs
- **Information** — We inform you on Espressif's products & wireless trends in trainings & webinars
- **Supply** — Popular Espressif SoCs, modules and EVKs available from stock

Contact us for Parts & Support

www.ineltek.com

Ineltek is the worldwide independent distributor with a **Passion for Innovation** and a **Commitment to Service**.

Founded in 1987, Ineltek gained the trust of thousands industrial customers as a technical semiconductor and design-in service provider. We work with your team to ensure our mutual success by providing the highest level of technical and commercial services for your projects.

Start your career with us!

Apply now at personal@ineltek.com

- Field Sales Engineer (m/f/x)
- Field Application Engineer (m/f/x)
- Line Management / Marketing (m/f/x)

Follow us!

INELTEK GmbH

Heidenheim, Wien, Castelfranco, London
Hamburg, München, Frankfurt, Dresden

46ème année n° 5H
décembre 2023 - janvier 2024
ISSN 0181-7450

N° de TVA Intracommunautaire :
FR90319937454

Dépôt légal : mai 2023
CPPAP 1125 T 83713
Directeur de la publication : Donatus Akkermans

Elektor Magazine est publié 8 fois par an par
PUBLITRONIC SARL – c/o Regus Roissy CDG
1, rue de la Haye – BP 12910
FR - 95731 Roissy CDG Cedex
www.elektor.fr | www.elektormagazine.fr

Pour toutes vos questions : service@elektor.fr

Devenez membre : www.elektormagazine.fr/abo

Publicité : Ouafae Hassani
Tél. : +31 (0)6 41312932
ouafae.hassani@elektor.com
www.elektormagazine.fr/publicité

Tarifs Annuels :
France 1 an 129,95 € (8 numéros)

Droits d'auteur
© 2023 Elektor International Media B.V.

Toute reproduction ou représentation intégrale ou partielle, par quelque procédé que ce soit, des pages publiées dans la présente publication, faite sans l'autorisation de l'éditeur est illicite et constitue une contrefaçon. Seules sont autorisées, d'une part, les reproductions strictement réservées à l'usage privé du copiste et non destinées à une utilisation collective, et, d'autre part, les analyses et courtes citations justifiées par le caractère scientifique ou d'information de l'oeuvre dans laquelle elles sont incorporées (Loi du 11 mars 1957 -art. 40 et 41 et Code Pénal art. 425).
Certains circuits, dispositifs, composants, etc. décrits dans cette revue peuvent bénéficier de droits propres aux brevets; la Société éditrice n'accepte aucune responsabilité du fait de l'absence de mention à ce sujet. Conformément à l'art. 30 de la Loi sur les Brevets, les circuits et schémas publiés dans Elektor ne peuvent être réalisés que dans des buts privés ou scientifiques et non commerciaux. L'utilisation des schémas n'implique aucune responsabilité de la part de la Société éditrice. La Société éditrice n'est pas tenue de renvoyer des articles qui lui parviennent sans demande de sa part et qu'elle n'accepte pas pour publication. Si la Société éditrice accepte pour publication un article qui lui est envoyé, elle est en droit de l'amender et/ou de le faire amender à ses frais; la Société éditrice est de même en droit de traduire et/ou de faire traduire un article et de l'utiliser pour ses autres éditions et activités, contre la rémunération en usage chez elle.

Imprimé aux Pays-Bas par Senefelder Misset, Mercuriusstraat 35, 7006 RK Doetinchem

Distribué en France par M.L.P. et en Belgique par A.M.P.

Certifié PEFC
Ce produit est issu de forêts gérées durablement et de sources contrôlées
PEFC
PEFC/30-31-151
www.pefc-france.org

Accélérer l'innovation IdO

C. J. Abate *(Directeur du contenu, Elektor)*
Jens Nickel *(Rédacteur en chef, Elektor)*

Teo Swee Ann
(Fondateur/PDG, Espressif Systems)

Après le succès retentissant des collaborations avec SparkFun (2021) et Arduino (2022), nous sommes ravis d'accueillir Espressif en tant que rédacteur invité pour 2023. Merci, Espressif !
Avec des solutions révolutionnaires telles que l'ESP8266 et l'ESP32, l'équipe d'Espressif a toujours fourni des outils de pointe aux ingénieurs professionnels les plus innovants, aux makers passionnés et aux étudiants les plus doués du monde. Cette édition spéciale du magazine Elektor et la collaboration passionnante qui en résulte avec les talentueux ingénieurs d'Espressif nous permettent de mettre à la disposition des membres d'Elektor du monde entier des exemples variés de solutions proposées par Espressif.
Alors, que vous réserve ce numéro du magazine Elektor ? Fidèle à la tradition Elektor, ce numéro est riche en contenu, des projets aux tutoriels, sur des sujets tels que l'identification de chansons sur une ESP32, le développement embarqué avec Rust, l'ESP32 et ChatGPT, la reconnaissance faciale avec l'ESP32-S3-EYE, des astuces techniques pratiques, et bien d'autres encore. Nous avons la certitude que ce numéro – ainsi que l'édition bonus gratuite que nous publions sur notre site et via notre lettre d'information – stimulera l'innovation pendant des mois et donnera naissance à des dizaines de nouveaux projets et applications électroniques fascinants.
Que vous soyez expérimenté avec les solutions Espressif ou que vous découvriez les produits de la société, veillez à publier tous vos projets basés sur les produits Espressif sur la plateforme en ligne Elektor Labs à l'adresse *elektormagazine.fr/labs*. Nous sommes impatients de découvrir vos créations. Bonne lecture et bonne chance pour votre prochain projet !

ESPRESSIF × elektor

Bienvenue dans cette édition spéciale du magazine Elektor – connu pour son dévouement à la promotion de l'innovation, de la collaboration et du partage des connaissances – corédigée par Espressif Systems. Nous tenons à remercier l'équipe d'Elektor et ses rédacteurs pour leur travail acharné et leur soutien continu et leur contribution qui ont permis à cette publication de voir le jour.
Nous tenons également à exprimer notre profonde gratitude à la communauté Espressif et à nos estimés partenaires, qui non seulement nous ont soutenus, mais ont également contribué activement à cette édition. Votre dévouement sans faille et vos idées précieuses ont contribué à élaborer le contenu de cette édition.
Cette édition témoigne de l'esprit de collaboration et de la passion partagée pour la technologie open-source qu'incarnent notre communauté et nos partenaires. Dans ce numéro, nous nous penchons sur les principes qui ont propulsé Espressif Systems à l'avant-garde de l'industrie IdO. Découvrez nos nouveautés et la manière dont elles ont contribué à la création d'une solide communauté de développeurs qui, à son tour, a joué un rôle essentiel dans notre parcours d'innovation.
Nous nous tournons également nos regards vers le futur, en observant le potentieldes technologies d'IA générative, notamment ChatGPT. Nous verrons comment elles sont prêtes à remodeler les industries - et comment Espressif Systems prévoit de les exploiter avec la participation de notre communauté et de nos partenaires.
Rejoignez-nous dans ce voyage où nous explorerons la technologie, l'innovation et la communauté. Ensemble, nous découvrirons le contenu fascinant de cette édition spéciale, qui suscitera votre enthousiasme pour les possibilités illimitées qu'offre le monde connecté d'aujourd'hui.
Continuez à innover et à partager !

notre équipe

Rédacteur en chef : Jens Nickel | **Rédaction :** Asma Adhimi, Roberto Armani, Eric Bogers, Jan Buiting, Stuart Cording, Rolf Gerstendorf (RG), Ton Giesberts, Hedwig Hennekens, Saad Imtiaz, Alina Neacsu, Dr. Thomas Scherer, Jean-Francois Simon, Clemens Valens, Brian Tristam Williams | **Contributeurs réguliers :** David Ashton, Tam Hanna, Ilse Joostens, Prof. Dr. Martin Ossmann, Alfred Rosenkränzer| **Maquette :** Harmen Heida, Sylvia Sopamena, Patrick Wielders | **Des questions techniques :** redaction@elektor.fr

DANS CE NUMÉRO

Elektor n° 5H
Décembre 2023 - janvier 2024

Rubriques

Articles de fond

Industrie

commutateur alimenté par pile bouton, basé sur un ESP32-C2
une durée de vie d'environ 5 années
126

Projets

Bientôt dans ces pages

Le numéro de janvier et février 2024
Vous retrouverez dans le prochain magazine Elektor l'habituel mélange stimulant de réalisations originales, de circuits, d'articles de fond, de sujets nouveaux, de trucs et d'astuces pour les électroniciens. Le thème de ce numéro sera « alimentations et énergie »

Bientôt dans ces pages :
- compteur d'énergie basé sur l'ESP32
- optimisation des centrales électriques de balcon
- bloc d'alimentation linéaire variable
- régulateur de puissance photovoltaïque simple
- récipient d'épicerie intelligent pour la cuisine
- programmation pour PC, tablette et smartphone
- projet : chargeur/déchargeur

Le numéro de janvier – février 2023 du magazine Elektor sera publié aux alentours du 10 janvier 2024. La date d'arrivée du magazine papier chez les abonnés dépend des aléas d'acheminement.

ÉDITION BONUS
Vous voulez plus de contenu de la part d'Elektor et d'Espressif ? Dans les semaines à venir, nous publierons une édition bonus du magazine Elektor - également rédigée en collaboration avec Espressif – qui regorge de projets, de tutoriels et d'articles de fond : un dekatron dans le style d'Espressif, un dongle d'authentification basé sur l'ESP32, un système ChatGPT parlant basé sur l'ESP-BOX, une interview avec le fondateur de Home Assistant Paulus Schoutsen, et bien plus encore ! Inscrivez-vous à la lettre d'information d'Elektor (*elektormagazine.fr/ezine*) pour recevoir l'édition bonus directement dans votre boîte de réception !

cadre photo couleur à encre électronique Wifi

Jeroen Domburg, Espressif

Les écrans à encre électronique occupent une large place sur le marché, mais ils sont principalement en noir et blanc, et les quelques modèles polychromes sont plutôt chers et difficiles à mettre en œuvre dans un projet d'amateur. Dans cet article, nous verrons comment utiliser un écran sept couleurs d'un prix raisonnable pour obtenir une bonne fidélité des couleurs - obtenue grâce à des techniques de tramage - et avec la capacité de se connecter à un réseau Wifi, en utilisant un module ESP32-C3-WROOM-02 d'Espressif.

Notre petite dernière est née il y a déjà quelque temps. Malheureusement, la dispersion de ses (arrière-) grands-parents de par le monde rend les visites en personne assez peu fréquentes. Bien sûr, les appels vidéo sont monnaie courante de nos jours, ce qui nous permet de rester en contact, mais je voulais que tout le monde puisse aussi la voir lorsque nous ne sommes pas connectés. Alors, il m'est venue une idée : construire un cadre photo autour d'un écran couleur et d'une puce Wi-Fi. Comme elle grandit très vite, ce cadre serait quotidiennement rafraîchi avec la photo du jour.

Vous avez sans doute déjà vu un écran à encre électronique : sinon une liseuse, du moins dans un magasin comme le supermarché local, où ils ont remplacé les anciennes étiquettes de prix en papier. Ces écrans sont parfaits pour afficher des images statiques, car ils conservent l'image la plus récente sans consommer d'énergie. La plupart de ces écrans sont en noir et blanc, avec parfois du rouge ou du jaune en option. Ils sont agréables, mais une image en noir et blanc ne rend pas justice aux couleurs vibrantes d'un bébé heureux trottant avec ses jouets colorés.

Sur le marché actuel

Au moment où j'écris ces lignes (début 2023), les liseuses dotées d'un écran couleur commencent enfin à apparaître. Leur qualité semble bonne, avec des couleurs proches de celles d'un journal. Malheureusement, elles sont chères et les écrans eux-mêmes ne semblent pas encore être disponibles en tant que composants, de sorte que les récupérer pour un projet d'amateur est pratiquement impossible.

Cependant, il existe un modèle d'écran couleur à encre électronique utilisable pour les projets de bricolage. Son principal vendeur, Waveshare, est une entreprise chinoise spécialiste du marché des bricoleurs, et le modèle le plus courant est un écran de 5,65 pouces à sept couleurs (640×448) (**figure 1**). Avec seulement sept couleurs et un temps de rafraîchissement d'environ une minute, il semble destiné à devenir le grand frère des étiquettes de prix, pour l'affichage d'informations statiques, avec les couleurs utilisées pour, par exemple, remplir des aplats, mais sûrement pas pour produire des images photoréalistes.

C'est un peu dommage : un écran à encre électronique constitue une excellente imitation d'une photo imprimée, car il ne nécessite pas de rétroéclairage et est entièrement statique. Je ne suis pas le seul à le penser, et un certain nombre de personnes ([1], [2], [3]) ont déjà essayé d'utiliser le tramage pour obtenir un affichage d'images avec une certaine fidélité. Voici ma propre tentative.

Figure 1. L'écran de 5,65 pouces à sept couleurs et 640×448 de Waveshare. (Source : Waveshare)

◀

Exigences en matière de matériel

Commençons par le matériel, avec quelques exigences de conception. Je voulais que le cadre photo se connecte à un serveur par Wifi une fois par jour pour récupérer de nouvelles images et en afficher une. En cas de succès, c'est ce qu'il ferait, sinon il continuerait à en afficher une ancienne. Il se remettrait ensuite en veille pendant 24 heures. Le choix d'une puce n'a pas été difficile, car je travaille pour Espressif, une société qui fabrique d'excellents microcontrôleurs gérant le Wifi et disposant d'un mode sommeil profond qui fonctionne assez bien. Pour ce projet, j'ai choisi un ESP32C3 : sympathique et bon marché, il possède toutes les fonctions dont j'avais besoin. J'avais également quelques exigences concernant l'alimentation électrique. Je voulais que cet appareil ressemble le plus possible à un cadre photo normal (sauf, bien sûr, qu'il change d'image chaque nuit), donc pas question d'une alimentation externe. Elle devait donc être une sorte de batterie interne, et comme je désirais expédier l'appareil à l'étranger, il fallait qu'elle soit amovible, ce qui excluait tout type de cellule Li-ion rechargeable. Comme vous pouvez le voir sur le schéma de la **figure 2**, j'ai décidé d'utiliser deux piles AA ; elles sont disponibles partout et même les grands-parents n'auraient aucun mal à les remplacer le moment venu. Comme deux piles AA délivrent une tension totale qui démarre à environ 3,2 volts et qui diminue au cours de leur durée de vie, j'avais besoin d'un circuit élévateur pour atteindre les 3,3 V nécessaires pour l'écran et l'ESP32C3.

Figure 2. Schéma complet du projet, comprenant le convertisseur DC-DC 3,3 V, le module ESP32-C3-WROOM-02 et l'alimentation de l'écran.

▼

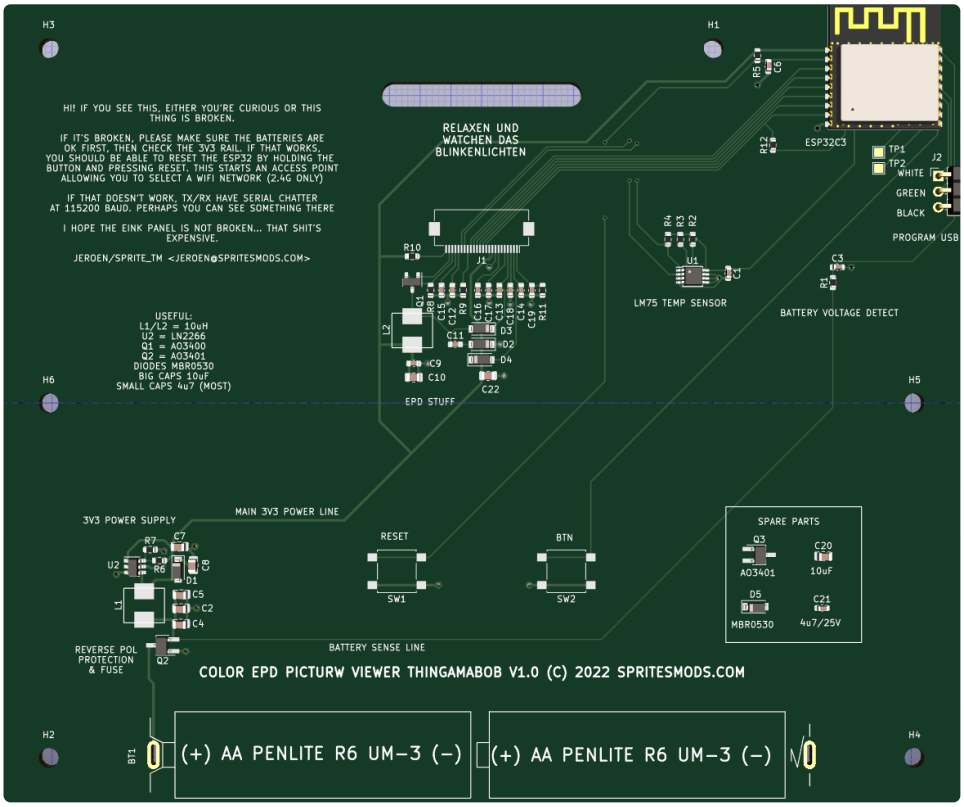


Figure 3. Sérigraphie (côté composant) du circuit imprimé spacieux du projet.

Les courbes de décharge [4] montrent que si je voulais tirer le meilleur parti d'une paire de piles alcalines, le circuit élévateur devait fonctionner avec une tension d'entrée de 2,0 V environ. De plus, comme l'ESP32-C3 doit être correctement alimenté même en veille profonde, le courant de repos doit être suffisamment faible. Avec ces exigences, je me suis mis à la recherche d'un convertisseur élévateur réalisable.

J'ai opté pour le Natlinear LN2266. Cette petite puce est fabriquée par un fabricant chinois, ce qui signifie qu'elle souffre un peu moins de la pénurie actuelle de silicium. Elle fonctionne à partir de 2 V, a un courant de repos de 56 µA et peut fournir les 500 mA de démarrage du Wifi dont l'ESP32-C3 a besoin. J'ai conçu le circuit de telle sorte que le LN2266 tire son énergie des deux piles via un MOSFET à canal P configuré pour protéger les piles si elles sont insérées dans le mauvais sens. La tension de la pile est également connectée via un filtre RC à l'une des broches CAN de l'ESP32-C3 afin qu'il puisse évaluer l'état des piles. J'avais prévu un fusible de protection, mais il provoquait une chute de tension inacceptable, alors je l'ai remplacé par une résistance de 0 ohm sur mes circuits imprimés et je l'ai entièrement supprimé dans les fichiers de conception.

L'ESP32-C3 se présente sous la forme d'un module ESP32-C3-WROOM-02, visible à droite sur le schéma de la figure 2. Ce module contient le microprocesseur Wifi et 4 Mo de mémoire flash, ainsi que tous les composants RF nécessaires, y compris une antenne imprimée. Pour le programmer, j'ai ajouté un bornier interne, sur lequel je peux souder un câble USB. Le convertisseur USB-JTAG-série interne s'occupe du reste. J'ai ajouté une fonction de mise à jour OTA (*Over the Air*) du micrologiciel à partir de mon serveur, qui m'offre la possibilité d'effectuer des mises à jour sur des unités qui sont déjà fermées et en service.

Ensuite, il y a l'écran à encre électronique. Il est connecté au circuit imprimé à l'aide d'un câble plat flexible, et il a besoin de quelques composants pour fonctionner, comme on peut le voir en bas à gauche de la figure 2 : un MOSFET, un inducteur, et quelques condensateurs et diodes pour générer les tensions dont il a besoin, quelques condensateurs de découplage et une ou deux résistances. L'écran dispose également d'une connexion pour un capteur de température LM75 externe. J'en ai prévu un, mais le micrologiciel actuel ne l'utilise pas, car l'écran fonctionne tout aussi bien sans lui.

Tout cela est placé sur un circuit imprimé très spacieux, dont la sérigraphie côté composants est illustrée à la **figure 3**. Comme l'écran à encre électronique est en

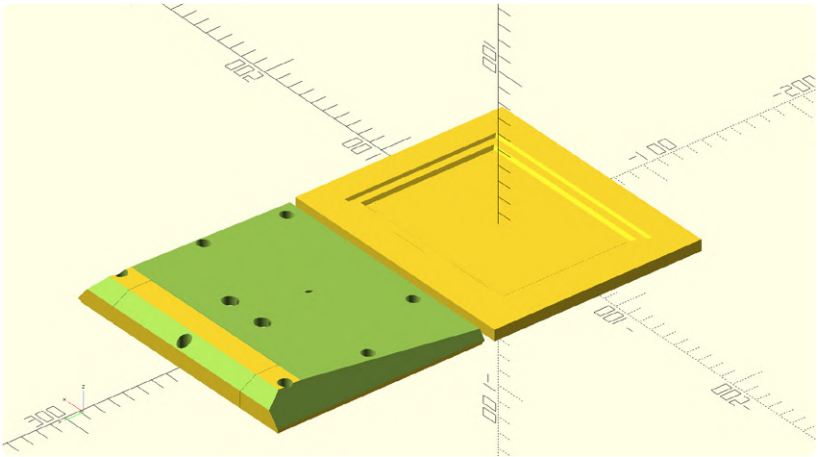


Figure 4. La structure du boîtier conçu avec OpenSCAD.

verre et quelque peu fragile, j'ai conçu le circuit imprimé pour qu'il serve de support et que le produit soit moins susceptible de se briser. Le circuit imprimé est un peu plus grand que cela, car les trous de montage, les composants traversants (tels que les contacts des piles) et l'antenne Wifi doivent tous être placés à l'extérieur de l'espace occupé par l'écran.

Le boîtier

Enfin, il y a le boîtier, que j'ai conçu avec OpenSCAD, comme le montre la **figure 4**. J'ai utilisé quelques astuces pour le rendre moins encombrant : il doit être assez épais pour loger les piles AA, et je ne voulais pas d'une grosse bosse dans le bas, alors j'ai chanfreiné les bords et donné une pente à l'arrière. Cela ne se voit pas si l'on regarde le cadre photo de face et rend son aspect plus élégant. Le cadre photo semblait également un peu encombrant par rapport à la zone visible de l'écran E-ink. Pour y remédier, j'ai ajouté un passe-partout. Comme ce passe-partout est imprimé 3D en blanc et qu'il contraste joliment avec le boîtier noir, il brise la « mer de noir » et donne un équilibre visuel à l'ensemble.

À l'arrière, il y a le logement des piles. Il s'ouvre en dévissant une vis (comme fermeture, je n'ai pas voulu me fier à de fragiles ergots imprimés en 3D) et, comme la sérigraphie du circuit imprimé indique l'orientation correcte des piles, leur remplacement devrait être facile.

Logiciel/micrologiciel et connexion Wifi

Comme l'ESP32-C3 est alimenté par des piles, j'ai essayé de faire en sorte qu'il en fasse le moins possible. Il doit se connecter au Wifi, communiquer avec mon serveur, voir s'il y a de nouvelles images qu'il n'a pas encore téléchargées et, si c'est le cas, les télécharger, déterminer quelle est la meilleure image à montrer (la plus récente ou celle qui a été montrée le moins souvent), l'afficher et s'éteindre. Évidemment, il y a d'autres choses à faire, comme la gestion des erreurs et la gestion des piles faibles.

Pour se connecter au Wifi, l'appareil a besoin d'un SSID et d'un mot de passe. Je n'ai pas voulu les coder en dur, au cas où ils devraient être changés. Ainsi, le micrologiciel inclut une copie de ESP32-WiFi-Manager [5], modifié pour corriger quelques bogues. Plus précisément, lorsque vous appuyez sur l'un des boutons à l'arrière du cadre photo tout en réinitialisant le cadre avec l'autre bouton, il démarre un point d'accès auquel vous pouvez vous connecter. Un serveur web intégré vous présente alors une interface utilisateur pour choisir un nouveau SSID et saisir son mot de passe.

Une fois que le cadre photo s'est connecté au Wifi, il tente de récupérer des instructions à partir d'une URL codée en dur. L'URL encode également certaines données d'état, telles que l'adresse MAC de l'appareil, la tension des piles et la version du micrologiciel installé. Le serveur renvoie la version du micrologiciel le plus récent pour l'appelant, ainsi qu'une liste des dix images les plus récentes. Certaines préférences sont également envoyées, telles que le fuseau horaire et l'heure à laquelle le cadre photo doit se réveiller pour tenter une mise à jour.

Le cadre photo dispose d'un espace de stockage pour dix images dans sa mémoire flash. Si le serveur dispose d'images non encore présentes localement, elles sont téléchargées et viennent écraser les images les plus anciennes. De cette manière, le cadre dispose toujours d'une réserve d'images relativement récentes, ce qui est utile en cas de dysfonctionnement de la connexion. Il permet même de déplacer le cadre photo d'un endroit à un autre : bien qu'en l'absence de connexion Wifi, il réutilise les anciennes photos, il aura toujours quelque chose de différent à afficher chaque jour.

La plupart des logiciels côté serveur ne sont pas très compliqués : il y a une page web frontale simple créée autour de *Cropper.js* [6], que vous pouvez ouvrir sur votre téléphone ou votre PC. Elle vous permet de sélectionner une image et d'en découper la partie que vous souhaitez afficher sur le cadre. Côté client, un peu de code JavaScript découpe et redimensionne l'image et envoie le résultat au serveur, lequel prend ces données, les convertit en données brutes pour l'écran, qu'il stocke dans une base de données MariaDB.

Lorsqu'un cadre photo se connecte, le serveur stocke les données reçues, de sorte que je dispose d'un journal des tensions des piles et que je peux voir si une mise à jour du micrologiciel a effectivement eu lieu. Le serveur vérifie ensuite dans la base de données MariaDB les dix dernières images, ainsi que d'autres informations, telles que la dernière version du micrologiciel, encode le tout en JSON et l'envoie. Tout cela n'est pas très compliqué.

Diffusion d'erreur

La seule partie réellement compliquée est la conversion de l'image RVB en les sept couleurs très spécifiques que l'écran est capable d'afficher. Prenons l'image de la **figure 5**, par exemple.

Si nous voulions convertir cette image en noir et blanc,

▲ *Figure 5. L'image utilisée pour les essais.*

Figure 6. Premier essai de conversion, en utilisant 100 % de noir et 100 % de blanc.

Figure 7. Conversion en noir et blanc par un processus de diffusion.

Figure 8. L'image de la figure 7 avec l'ajout de valeurs RVB (voir texte).

Figure 9. L'image obtenue par application de la norme CIEDE2000.

nous pourrions simplement vérifier la luminance (clarté) de chaque pixel et, si elle est plus proche du noir, rendre le résultat noir ; si elle est plus proche du blanc, rendre le résultat blanc. En d'autres termes, nous prenons la « couleur » la plus proche (en limitant les « couleurs » à 100 % de noir et 100 % de blanc) et nous obtenons l'image représentée sur la **figure 6**.

De toute évidence, cette image ne ressemble guère à l'image originale. Même en noir et blanc, nous pouvons faire mieux en utilisant ce que l'on appelle la *diffusion des erreurs*. Chaque fois que nous mettons un pixel gris en noir ou en blanc, nous prenons la différence entre la luminance du pixel dans la photo originale et la luminance du pixel que nous affichons réellement sur l'écran (l'« erreur » dans « diffusion d'erreur ») et en ajoutons une fraction aux pixels voisins (la « diffusion »). Le processus de diffusion peut être réalisé de plusieurs manières, la méthode Floyd-Steinberg étant la plus courante, et il permet d'obtenir une très bonne image noir et blanc tramée, comme illustré à la **figure 7**.

Nous pouvons également l'utiliser pour notre écran à sept couleurs. Le problème est que la définition de la « couleur la plus proche » devient beaucoup plus compliquée, de même que la définition de l'« addition ». Même la définition du terme « couleur » est délicate, car, en l'absence de rétroéclairage, les couleurs perçues dépendent de la lumière ambiante : à la lueur de la flamme d'une bougie, vous ne verrez pas les mêmes couleurs qu'en plein soleil. Pour obtenir les couleurs, j'ai pris une lampe à température réglable, je l'ai réglée sur 4800 K (la température de couleur moyenne à laquelle je pense que l'écran sera regardé), j'ai affiché les 7 couleurs sous forme d'aplats rectangulaires sur l'écran à encre électronique que j'ai photographié. J'ai importé cette photo dans mon ordinateur et j'ai ajusté les couleurs manuellement jusqu'à ce que les couleurs affichées à l'écran soient aussi proches que possible de celles de l'écran à encre électronique. J'ai ensuite pris les valeurs RVB moyennes des sept couleurs et les ai entrées dans mon programme.

Parmi ces sept couleurs, pour obtenir la couleur la plus proche sur n'importe quel pixel de l'image source, nous avons besoin d'un moyen de comparer deux couleurs et, comme nous utilisons des couleurs réelles et pas seulement du noir et du blanc, nous ne pouvons plus nous contenter d'utiliser la luminance. Une façon rapide et pratique de comparer deux couleurs consiste à considérer

l'espace RVB (linéarisé) comme un espace tridimensionnel et à utiliser la distance euclidienne pour mesurer la proximité de deux couleurs. Avec ce modèle, il suffit d'ajouter les valeurs RVB pour ajouter des couleurs. Si nous modifions l'algorithme de Floyd-Steinberg pour adopter cette méthode de sélection de la couleur la plus proche, nous obtenons une image raisonnablement acceptable, comme le montre la **figure 8**.

Ce n'est déjà pas si mal ! Cependant, il apparaît quelques étranges artefacts de couleur. Le plus évident est que le pot de fleurs n'est pas de la même nuance de bleu : C'est parce que l'écran à encre électronique ne dispose tout simplement pas des couleurs nécessaires pour reproduire cette nuance, et aucun algorithme ne peut compenser cela. Mais il y a d'autres bizarreries sous la forme d'étranges bandes de couleur, par exemple dans l'ombre sous le bras gauche du singe, et le ventre du singe est plus orange que sur l'image originale.

L'approche par les espaces de couleurs

Le problème du calcul des différences de couleur dans l'espace RVB est que vos yeux ne travaillent pas réellement dans un espace RVB linéaire. La différence entre deux couleurs est en fait beaucoup plus difficile à définir, et il existe de nombreuses approches pour y parvenir. L'une des premières approches consistait à convertir les couleurs RVB dans l'espace colorimétrique CIELAB et à prendre la différence. Largement recommandée sur l'internet, cette approche ne donne pas vraiment de bons résultats si les couleurs ne sont pas complètement saturées. Pour moi, la meilleure approche a été d'appliquer la norme CIEDE2000 [7]. Il s'agit de l'un des modèles de perception les plus récents et, bien qu'il ne soit pas d'emploi facile, il donne les meilleurs résultats, comme le montre la **figure 9**. Heureusement que j'avais déjà décidé de faire ce calcul laborieux côté serveur : côté cadre, il aurait coûté cher en piles !

Notez que les bandes de couleur dans les ombres ont disparu et que le ventre du singe est d'un rouge plus précis. L'image présente encore des défauts, comme la couleur du pot de fleurs, mais, comme je l'ai déjà mentionné, c'est faute des couleurs nécessaires pour afficher correctement cette teinte particulière.

Pour plus de rapidité, toute cette logique est codée dans un simple programme C. Après avoir été éditée par un utilisateur (sur une page web utilisant *Cropper.js*), l'image est envoyée au serveur qui la soumet à ce programme qui la convertit en pixels d'écran à encre électronique lesquels sont stockés dans une table MariaDB, à la disposition des cadres photo à chaque fois qu'ils se connectent.

La page web est également utilisable sur téléphone mobile, de sorte que s'il vous arrive de prendre une image particulièrement belle, vous pouvez l'éditer immédiatement et la mettre en file d'attente pour la distribuer à tous les cadres photos (**figure 10**).

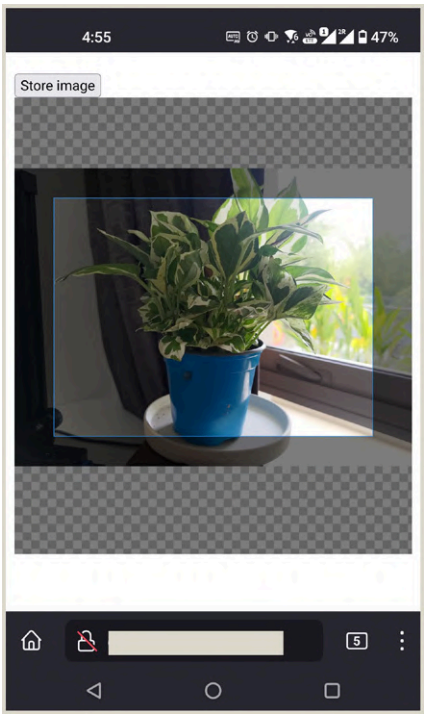


Figure 10. Édition d'image pour une utilisation sur téléphone mobile.

Figure 11. Circuit imprimé complètement assemblé.

Figure 12. Les deux parties du boîtier réalisé pour ce projet.

Résultat

Une fois le circuit imprimé réalisé et les pièces imprimées en 3D, il est temps d'assembler le tout, comme le montre la **figure 11**.

L'arrière du circuit imprimé contient toute l'électronique. Le circuit imprimé est en fait conçu pour être réparable : s'il se casse et que je ne suis pas là pour le réparer, il se peut que j'aie un ou deux amis qui s'y connaissent en électronique et qui peuvent y jeter un coup d'œil. Cela signifie qu'il y a des instructions de débogage à l'arrière et même quelques composants de rechange. Sinon, le circuit imprimé est assez peu peuplé : ses dimensions sont essentiellement définies par la taille de l'écran à encre électronique situé de l'autre côté. L'espace disponible m'aurait permis d'utiliser des composants plus gros, mais j'ai une multitude de bobines de composants 0603, qu'il faut bien que je consomme ; avec un bon fer à souder et un microscope binoculaire, je suis parfaitement à l'aise pour les souder à la main.

Le boîtier est illustré à la **figure 12** avec le passe-partout

Figure 13. Le compartiment à piles avec son couvercle.

Figure 14. Le résultat final, au pied de l'original.

blanc à l'intérieur. Celui-ci comporte une découpe pour l'écran à encre électronique : Même si l'adhésif qui fixe l'écran au circuit imprimé venait à se décoller, il resterait en place. Le boîtier se ferme à l'aide de vis qui sont vissées dans des inserts en laiton. Ces inserts sont mis en place en les chauffant à l'aide d'un fer à souder (avec une panne usagée – je ne veux pas endommager une panne utilisable) ce qui fait fondre le plastique.

Tout est assemblé à l'aide d'une série de vis M3. Le compartiment des piles est doté d'un petit couvercle (**figure 13**), de sorte qu'il n'est pas nécessaire de démonter l'ensemble pour remplacer les piles. D'après mes calculs, les piles devraient durer plus d'un an. Notez également qu'il y a deux boutons à l'arrière. L'un est connecté directement au reset de l'ESP32, et l'autre à un GPIO général. Vous pouvez utiliser le bouton de réinitialisation pour que l'appareil effectue un cycle manuel de connexion et de rafraîchissement, ce qui a pour effet secondaire d'afficher l'image suivante. Lorsque l'autre bouton est maintenu enfoncé pendant que le cadre photo est réinitialisé, il démarre la fonction point d'accès qui vous permet de vous connecter au cadre et de reconfigurer les paramètres de la connexion Wifi.

Enfin, sur la **figure 14**, vous pouvez admirer le résultat final. Cette image n'est pas la plus fidèle qui soit, mais c'est largement compensé par l'affichage d'une nouvelle photo chaque jour depuis l'autre bout du monde.

Comme d'habitude, ce projet est open source : avec une imprimante 3D, un accès à un serveur et quelques compétences, vous pouvez réaliser votre propre version de ce cadre. Tous les sources, les dessins des circuits imprimés, etc. sont disponibles sur GitHub [8].

VF : Helmut Müller — 230464-04

Produits

- **Waveshare 5.65" ACeP 7-Color E-Paper E-Ink Display Module (600×448)**
  www.elektor.fr/19847
- **ESP32-DevKitC-32E**
  www.elektor.fr/20518
- **Dogan Ibrahim,** *The Complete ESP32 Projects Guide,* **Elektor 2019**
  www.elektor.fr/18860

Questions ou commentaires ?

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

À propos de l'auteur

Jeroen Domburg est Senior Software and Technical Marketing Manager chez Espressif Systems. Avec plus de 20 ans d'expérience dans le domaine de l'embarqué, il est participe au processus de conception logicielle et matérielle des SoC d'Espressif. Pendant son temps libre, il aime bricoler avec l'électronique pour réaliser des appareils qui ont une utilité pratique ou pas !

LIENS

[1] Cadre photo papier électronique 7 couleurs de GitHub : https://github.com/robertmoro/7ColorEPaperPhotoFrame
[2] Cadre de papier électronique à Raspberry Pi : https://reddit.com/r/raspberry_pi/comments/10dbhnj/i_built_a_raspberry_pi_epaper_frame_that_shows_me
[3] Projet d'image papier électronique sur YouTube : https://youtu.be/YawP9RjPcJA
[4] Graphiques de décharge : https://powerstream.com/AA-tests.htm
[5] ESP32-WiFi-Manager : https://github.com/tonyp7/esp32-wifi-manager
[6] Cropper.js : https://fengyuanchen.github.io/cropperjs
[7] Norme CIEDE2000 : https://en.wikipedia.org/wiki/Color_difference#CIEDE2000
[8] Dépôt GitHub : https://github.com/Spritetm/picframe_colepd

tutoriel ESP-Launchpad

flashaer des micrologiciels en quelques minutes

Dhaval Gujar, Espressif

Vous travaillez avec des puces Espressif ? ESP-Launchpad est un outil web qui simplifie l'évaluation et le test des micrologiciels. Observons-le de plus près.

Cet article présente ESP-Launchpad, un outil web qui permet de vérifier et de tester les micrologiciels pour les puces Espressif en toute simplicité. Il simplifie la configuration de l'environnement de développement, exploite les navigateurs web pour le flashage des micrologiciels et est destiné aussi bien aux développeurs qu'aux autres utilisateurs.

Pourquoi choisir ESP-Launchpad ?

Lorsque vous créez un projet open-source, vous souhaitez offrir aux utilisateurs un moyen simple de l'étudier. Pour les développeurs embarqués, cet objectif implique la configuration de l'hôte de développement et des outils de programmation, le clonage du projet, la compilation et la programmation du matériel.

Comme l'hôte de développement et l'environnement logiciel peuvent être différents, il y a de fortes chances qu'une de ces étapes se passe mal. Et même si tout fonctionne bien, la tâche est trop compliquée pour les utilisateurs non-développeurs qui veulent simplement tester le projet.

Si vous développez avec l'une des puces Espressif, vous avez désormais une meilleure possibilité de développer facilement votre projet. ESP-Launchpad simplifie ce processus et permet d'évaluer et de tester rapidement et facilement les micrologiciels développés pour la plateforme ESP.

À vos marques, prêts, ESP-Launchpad !

ESP-Launchpad est un outil web qui utilise les fonctions de navigation les plus modernes pour flasher de manière simple et sans fioritures des micrologiciels préconçus sur des contrôleurs ESP. Les installations traditionnelles de logiciels dédiés pour la mise en place d'hôtes de développement ne sont plus nécessaires ; à la place, on profite de l'accessibilité d'une plateforme web : l'interface web série des navigateurs Chrome, Edge et Opera permet de communiquer avec la carte de développement, de programmer et d'accéder à la console série depuis le navigateur.

Voyons comment lancer facilement un micrologiciel avec ESP-Launchpad.

ESP-Launchpad est un outil web qui utilise les dernières fonctions du navigateur pour flasher facilement des micrologiciels pré-conçus sur des contrôleurs ESP.

Préparation du micrologiciel

Une fois que votre firmware est prêt, l'étape suivante consiste à créer des builds pour tous les systèmes cibles que vous souhaitez prendre en charge et à les convertir en fichiers binaires individuels. Ne vous inquiétez pas, esptool.py (l'utilitaire Python principal d'Espressif pour tout ce qui concerne le flashage) offre une option pratique `merge_bin` qui vous permet de créer facilement un tel fichier binaire. Vous trouverez de plus amples informations à ce sujet sur notre page de documentation [1]. Ces différents fichiers binaires doivent être téléchargés vers une URL accessible au public, que nous utiliserons plus tard.

Remarque : ce tutoriel n'aborde pas en détail la manière d'y parvenir.

Ce qu'il faut savoir : le secret derrière ESP-Launchpad. est un portage JavaScript d'`esptool`, appelé `esptool-js`, qui utilise en interne l'API WebSerial.

Comprendre les bases : le pouvoir de TOML

Avant de nous plonger dans la pratique, nous devons nous familiariser avec la structure TOML de la configuration d'ESP-Launchpad, un aspect fondamental d'ESP-Launchpad. Les fichiers TOML servent ici de simples modèles de configuration qui décrivent en détail les composants de votre micrologiciel, le matériel pris en charge et les applications complémentaires.

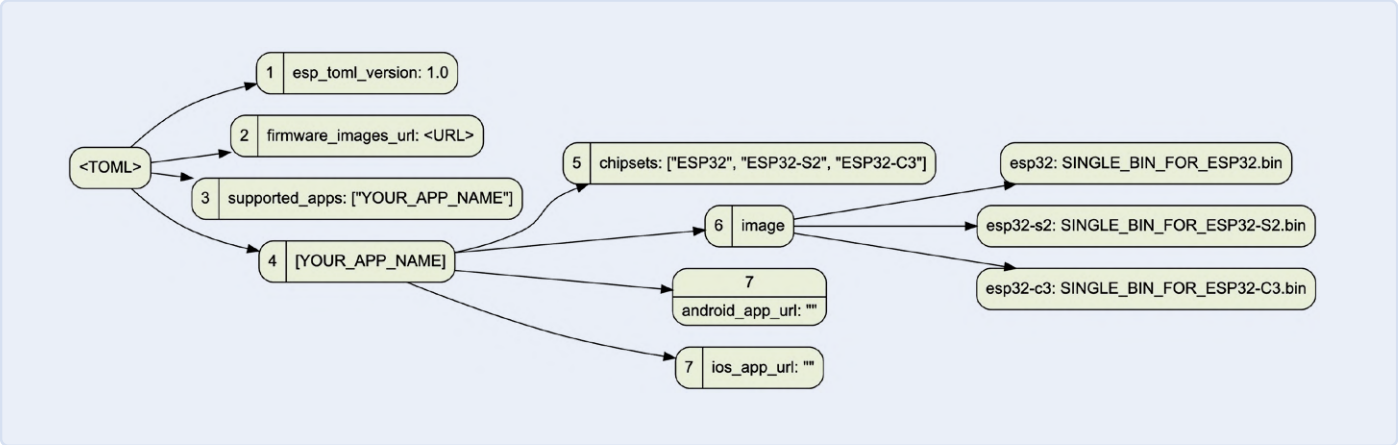


Figure 1. Diagramme en arbre de la structure TOML.

Voici un exemple :

```
esp_toml_version = 1.0
firmware_images_url = "<URL to your firmware images
directory>"
supported_apps = ["YOUR_APP_NAME"]

[YOUR_APP_NAME]
chipsets = ["ESP32", "ESP32-S2", "ESP32-C3"]
image.esp32 = "SINGLE_BIN_FOR_ESP32.bin"
image.esp32-s2 = "SINGLE_BIN_FOR_ESP32-S2.bin"
image.esp32-c3 = "SINGLE_BIN_FOR_ESP32-C3.bin"
android_app_url = ""
ios_app_url = ""
```

Pour mieux comprendre, jetez un coup d'œil à la **figure 1**.

- `esp_toml_version` indique la version du schéma TOML.
- `firmware_images_url` est une URL publique du serveur de fichiers sur lequel les fichiers binaires de votre firmware sont disponibles au téléchargement. Conseil de pro : nous hébergeons aussi bien nos fichiers TOML que nos fichiers binaires sur GitHub, en utilisant GitHub Pages !
- `supported_apps` est un tableau contenant la liste des applications supportées et pour lesquelles les fichiers binaires sont disponibles. Vous pouvez avoir plusieurs applications, qui apparaissent dans la liste déroulante des applications disponibles sur l'interface utilisateur d'ESP-Launchpad. Les trois valeurs que nous venons d'examiner étaient les paires clé-valeur du niveau supérieur.
- `[YOUR_APP_NAME]` — Il s'agit d'un tableau qui est pratiquement une collection de paires clé-valeur. Il contient
  - `chipsets` — Un tableau avec une liste de puces ESP pour lesquels vous allez fournir un micrologiciel prêt à l'emploi. Notez la convention de nommage ici, tout en majuscules, ESP32-C3.
  - `image.<chip-name>` — Ce sont des clés qui relient le nom de l'image binaire à chacune des destinations prises en charge. Celle-ci est ajoutée à `firmware_images_url` afin d'obtenir l'URL finale pour le fichier binaire. Respectez ici aussi la convention de nommage : tout en minuscules, esp32-c3.
  - `ios_app_url` et `android_app_url` sont des paires clé-valeu optionnelles mais spéciales sous la même table d'application, qui vous permettent d'afficher des liens sous forme de codes QR que les utilisateurs peuvent facilement scanner une fois que votre application a été flashée avec succès.

Voyons à quoi ressemble ESP-Launchpad (**figure 2**) lorsque nous fournissons l'exemple de TOML que nous venons de créer avec l'URL imaginaire suivante :
https://espressif.github.io/esp-launchpad/?flashConfigURL=https://some-nice-url/my_app.toml

OK, ça a l'air bien ! Et ensuite ?

Seulement trois étapes simples pour l'utilisateur !

- Connecter l'appareil : connectez l'ESP au port série USB de votre ordinateur.
- Effectuer la connexion : établissez une connexion avec l'appareil dans le menu de l'outil.
- Sélectionner et flasher : sélectionnez et flashez le firmware préconçu.

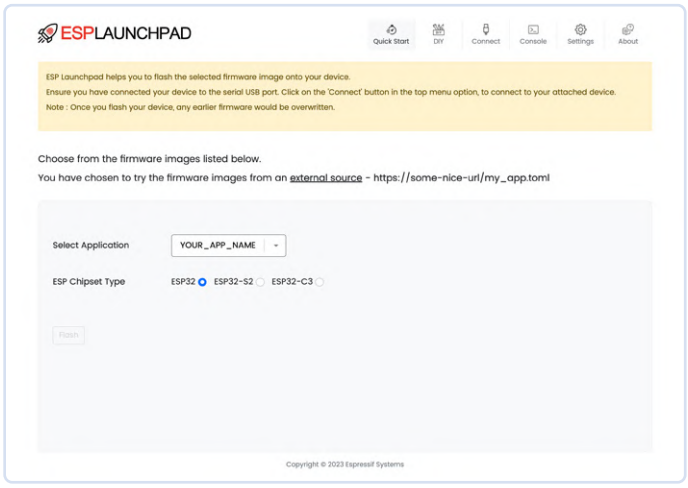


Figure 2. ESP-Launchpad : exemple de configuration chargé.

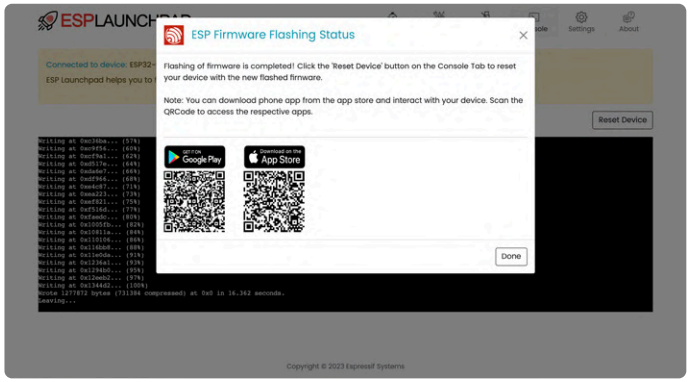


Figure 3. Écran final d'ESP-Launchpad

Après le flashage, l'utilisateur est accueilli par une console qui affiche le journal de l'appareil et un popup avec les codes QR des applications mobiles, si vous les avez ajoutées au TOML (**figure 3**).
Félicitations, vous venez de publier votre application avec ESP-Launchpad ! Conseil de pro : nous avons créé un badge (**figure 4**) que vous pouvez ajouter au fichier README ou sur la page web de votre projet et qui contient un lien hypertexte vers l'URL de configuration Flash de votre application ! Pour plus d'informations, consultez le fichier *README.md* du projet [2].

ESP-Launchpad : le firmware rencontre la communauté

La particularité d'ESP-Launchpad est qu'il offre aux utilisateurs la possibilité de mettre leurs applications de micrologiciel à la disposition d'autres personnes. Dans un simple fichier de configuration, les développeurs peuvent définir d'où proviennent les fichiers binaires prédéfinis de leur micrologiciel, quel matériel est pris en charge et même créer un lien vers des applications complémentaires. Cette approche globale garantit que le micrologiciel n'est pas seulement partagé en tant que fichier binaire, mais qu'il offre aussi une expérience visée par le développeur.

LIENS

[1] Commandes de base - fichiers binaires à flasher :
merge_bin : https://tinyurl.com/esp32mergebinaries
[2] Dépôt GitHub du projet :
https://github.com/espressif/esp-launchpad

Considérations

Le véritable potentiel de tout outil se révèle lorsqu'il est mis en pratique par les utilisateurs. Explorez ESP-Launchpad, intégrez-le dans vos processus de travail et partagez vos expériences ! Vos connaissances et vos contributions sont d'une valeur inestimable pour améliorer encore notre offre. Ensemble, redéfinissons l'expérience de l'évaluation open-source.

230596-04

Questions ou commentaires ?

Envoyez un courriel à l'auteur (dhaval.gujar@espressif.com) ou contactez Elektor (redaction@elektor.fr).

À propos de l'auteur

Dhaval Gujar est ingénieur chez Espressif Systems, spécialisé dans les systèmes embarqués. Ce qui le motive, c'est la convergence dynamique de technologies telles que le cloud, l'IdO et plus encore. Dhaval est passionné par le paysage technologique en évolution et par les changements technologiques modernes.

Produits

- **ESP32-DevKitC-32E (SKU 20518)**
  www.elektor.fr/20518
- **Carte de développement LILYGO T-Display-S3 ESP32-S3 (avec connecteurs) (SKU 20299)**
  www.elektor.fr/20299

SDK

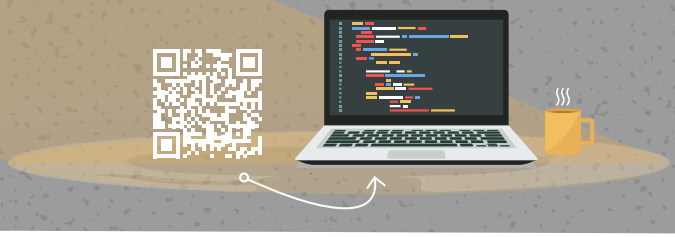
le protocole ESP-NOW

ESPRESSIF

pour une communication et un contrôle flexibles

Pour certains projets, vous souhaitez parfois disposer d'une communication directe d'appareil à appareil sans passer par un réseau d'infrastructure. Le protocole ESP-NOW prend en charge une portée de plus de 220 mètres dans un environnement ouvert. ESP-NOW permet la communication et le contrôle d'appareils un à un et un à plusieurs. Ce SDK fournit des exemples de la façon dont le protocole ESP-NOW peut être utilisé pour la transmission OTA, l'approvisionnement et le contrôle des appareils. Ce SDK contient également une implémentation pour un interrupteur à pile qui peut contrôler des appareils via le protocole ESP-NOW.

https://github.com/espressif/esp-now



ESP32 et ChatGPT

vers un système d'autoprogrammation...

Figure 1. Les cartes Arduino Nano ESP32, avec les environnements Arduino (à gauche) et MicroPython (à droite).

Saad Imtiaz (Elektor Lab)

Nous avons combiné la puissance de deux microcontrôleurs Arduino Nano ESP32, la communication sans fil et l'API ChatGPT pour créer un système de communication intelligent et interactif. L'une des cartes est connectée à ChatGPT ; les réponses et le code à renvoyer de cet outil d'intelligence artificielle populaire seront transférés sans fil à l'autre carte Nano. Ce système à deux cartes peut-il se programmer lui-même avec ChatGPT ? Suivez les instructions pas à pas et découvrez comment les cartes communiquent et comment fonctionne l'API ChatGPT.

Dans le domaine des systèmes embarqués et de l'Internet des objets (IoT), le microcontrôleur ESP32 d'Espressif a gagné en popularité grâce à sa polyvalence et à ses capacités robustes. Grâce à sa connectivité Wifi intégrée et à ses puissantes capacités de traitement, il ouvre un éventail de possibilités pour la construction d'appareils intelligents et connectés. Dans cet article, nous examinons un projet de démonstration qui utilise l'API ChatGPT, un modèle de langage d'IA créé par OpenAI, pour atteindre de nouveaux sommets avec deux cartes Arduino Nano ESP32.

L'objectif de ce projet est de créer un système de communication transparent entre les cartes Nano ESP32, l'une fonctionnant avec l'EDI Arduino et l'autre avec l'environnement MicroPython. En exploitant l'API ChatGPT, nous permettons à ces cartes de s'engager dans des conversations engageantes et interactives. Imaginez les possibilités d'envoyer à vos microcontrôleurs des extraits de code ou de recevoir des solutions créatives à vos questions.

Tout au long de cet article, nous aborderons les aspects techniques de la configuration du matériel et du logiciel et de l'établissement de la communication entre les cartes Nano ESP32. Nous explorerons les avantages et les limites de chaque *framework* (EDI Arduino et MicroPython) en mettant en lumière leurs caractéristiques uniques et leurs cas d'utilisation. En outre, nous fournirons des extraits de code pratiques, des explications et des idées pour vous aider à comprendre le fonctionnement interne de ce projet.

À la fin de cet article, vous comprendrez clairement comment construire votre propre système de communication basé sur le Nano ESP32, en exploitant la puissance de l'IA et de l'IdO. Alors, plongeons dans cette aventure passionnante et commençons à créer un environnement intelligent et interactif avec le Nano ESP32 et ChatGPT !

Matériel

Pour commencer ce projet, nous aurons besoin de quelques composants essentiels. Examinons de plus près le matériel exigé :

1. **Modules Arduino Nano ESP32 (×2)**
   Le microcontrôleur ESP32 est au cœur de notre projet. Pour la communication, nous aurons besoin de deux cartes Arduino Nano ESP32. Ces cartes sont très répandues et offrent de nombreuses fonctionnalités telles que la connectivité Wifi, une grande puissance de traitement et des broches GPIO pour l'interfaçage avec des composants externes.
2. **Câbles USB Type-C (×2)**
   Les câbles USB Type-C sont nécessaires pour alimenter et programmer les cartes Nano ESP32. Ces câbles nous permettent d'établir une connexion entre les cartes et notre ordinateur, facilitant ainsi la programmation et le transfert de données.
3. **Réseau Wifi**
   Un réseau Wifi stable est essentiel pour établir la communication entre les deux cartes et l'API ChatGPT. Assurez-vous d'avoir accès à un réseau Wifi fiable avec une connectivité Internet.

Cela permettra à la première carte Nano ESP32 (appelée ici « Nano ESP32-1 ») de se connecter à l'API ChatGPT et d'échanger des messages.

Maintenant que nous connaissons les composants nécessaires, passons aux aspects logiciels et de programmation du projet.

Logiciel et programmation

Pour mettre en place le système de communication et programmer les cartes, nous aurons besoin des outils logiciels suivants :

1. **EDI Arduino**
   L'environnement de développement intégré (EDI) Arduino est le plus largement utilisé pour la programmation des microcontrôleurs basés sur le firmware Arduino. Désormais, avec sa nouvelle version 2.0, il offre une interface plus conviviale, un langage de programmation simplifié et une bibliothèque étendue de fonctions préconstruites qui facilitent le travail avec les cartes Nano ESP32.
2. **Firmware MicroPython**
   MicroPython est une version légère du langage de programmation Python optimisée pour les microcontrôleurs. Il offre une expérience de programmation plus flexible et interactive par rapport à l'EDI Arduino. Pour programmer la deuxième Nano ESP32 (ESP32-2) en MicroPython, nous devrons installer le micrologiciel MicroPython à bord. Dans la **figure 1,** vous pouvez voir les deux cartes connectées. La carte de gauche fonctionne avec le *framework* Arduino et celle de droite fonctionne avec MicroPython.
3. **Accès à l'API ChatGPT**
   Pour connecter la carte Nano ESP32-1 à l'API ChatGPT, vous devez avoir accès à l'API. Pour créer votre clé API, vous devrez vous rendre sur [1], créer un compte Open AI, puis vous rendre sur [2] et cliquer sur *Create New Secret Key* ; vous pouvez également le faire en cliquant sur l'icône de profil dans le coin supérieur droit de la page Web et en sélectionnant *View API Keys* dans le menu déroulant. Une fois que vous avez créé la clé API, sauvegardez-la dans un endroit sûr car vous ne pourrez pas la recopier.
4. **Programmation**
   Nous allons commencer avec le Nano ESP32-1 fonctionnant avec l'EDI Arduino. Tout d'abord, nous ajoutons les bibliothèques appropriées à notre code. La bibliothèque *WiFi.h* est nécessaire pour que nous puissions ajouter la fonction de connexion du Nano ESP32 au réseau Wifi. Ensuite, *HTTPClient.h* est utilisé pour envoyer nos données, c'est-à-dire la réponse de ChatGPT à la seconde carte Nano ESP32. Enfin, nous utilisons *ArduinoJson.h* (la bibliothèque permettant d'utiliser JSON sur pratiquement toutes les cartes existantes). Elle nous permet la sérialisation et la désérialisation JSON, que nous utiliserons pour accéder à l'interface API ChatGPT, envoyer des requêtes et recevoir des réponses. Les codes des deux cartes sont également disponibles sur GitHub [3].

```
// Load Wi-Fi library
#include <WiFi.h>
#include <HTTPClient.h>
#include <ArduinoJson.h>

// Replace with your network credentials
const char* ssid = "WIFI_SSID";
const char* password = "WIFI_PWD";
//chatgpt api key
const char* apiKey =
  "sk-xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx";
```

Nous allons maintenant ajouter l'adresse IP de la deuxième carte Nano ESP32 (cette adresse IP sera affichée dans le moniteur série de l'EDI, lorsque la deuxième carte Nano ESP32 sera connectée au réseau Wifi).

```
const char* serverIP = "192.168.1.82";
  // IP address of the second Nano ESP32
const uint16_t serverPort = 80;
  // Port number on the second Nano ESP32
```

Une fonction a été créée spécifiquement pour se connecter à l'API ChatGPT et communiquer avec elle, voir **listage 1**.
Passons maintenant au code de la deuxième carte Nano ESP32. Avant de commencer la programmation, expliquons comment exécuter MicroPython sur le Nano ESP32 : nous utiliserons *The Arduino Lab*

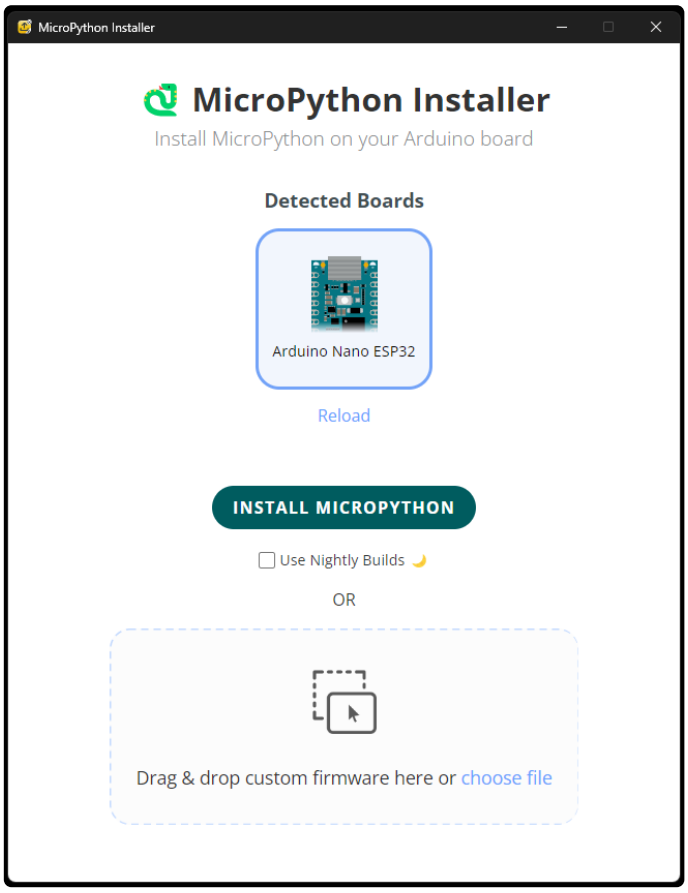


Figure 2. Outil MicroPython Firmware par Arduino Labs.

Listage1. Code Arduino pour envoyer des requêtes à ChatGPT.

```
String sendChatGPTRequest(String prompt) {
 String received_response= "";
 String apiUrl = "https://api.openai.com/v1/completions";
 String payload = "{\"prompt\":\"" + prompt +
                  "\",\"max_tokens\":100, \"model\": \"text-davinci-003\"}";
 HTTPClient http;
 http.begin(apiUrl);
 http.addHeader("Content-Type", "application/json");
 http.addHeader("Authorization", "Bearer " + String(apiKey));

 int httpResponseCode = http.POST(payload);
 if (httpResponseCode == 200) {
 String response = http.getString();

 // Parse JSON response
 DynamicJsonDocument jsonDoc(1024);
 deserializeJson(jsonDoc, response);
 String outputText = jsonDoc["choices"][0]["text"];
 received_response = outputText;
 //Serial.println(outputText);
 } else {
 Serial.printf("Error %i \n", httpResponseCode);
 }
 return received_response;
}
```

pour MicroPython. Pour commencer, nous allons d'abord nous rendre sur [4] et télécharger l'outil *Arduino MicroPython firmware* pour flasher l'Arduino Nano ESP32, comme le montre la **figure 2** ci-dessous.
Connectez votre Arduino Nano ESP32 à votre ordinateur, une fois la carte détectée, cliquez sur *Install Micropython* et après quelques secondes, votre carte sera flashée avec le dernier firmware, et vous serez capable d'utiliser MicroPython sur le Nano ESP32.
Nous allons maintenant télécharger *Arduino Labs for MicroPython IDE* depuis son site officiel [5] et télécharger la dernière version. Après avoir extrait le fichier, lancez le logiciel en exécutant le fichier *Arduino Labs for Micropython.exe*.
Sur la deuxième carte Nano ESP32, le code est court et simple ; nous allons d'abord importer les bibliothèques. Nous utiliserons les bibliothèques *network* et *socket* pour nous connecter au réseau Wifi et pour créer notre port serveur où nous pourrons recevoir le code ou les réponses du Nano ESP32-1. De plus, les bibliothèques *machine* et *time* sont nécessaires pour utiliser les GPIO et la fonction *timer* du Nano ESP32.

```
import network
import socket
import machine
import time

# Wi-Fi credentials
wifi_ssid = "WIFI_SSID"
wifi_password = "WIFI_PWD"

# Connect to Wi-Fi
wifi = network.WLAN(network.STA_IF)
wifi.active(True)
wifi.connect(wifi_ssid, wifi_password)

# Wait until connected to Wi-Fi
while not wifi.isconnected():
 pass
# Print the Wi-Fi connection details
print("Connected to Wi-Fi")
print("IP Address:", wifi.ifconfig()[0])
```

Ensuite, un serveur socket est créé pour recevoir les données du Nano ESP32-1, puis le microcontrôleur entre dans une boucle `while` où il attend de recevoir d'éventuelles réponses ; lorsqu'un code ou une réponse est reçu, le Nano ESP32 exécute le code puis recommence à attendre d'autres instructions (voir **listage 2**).
Dans l'encadré **EDI Arduino, MicroPython et plus encore**, nous discutons des avantages et des inconvénients des deux environnements et comprenons pourquoi l'un peut être plus approprié que l'autre selon le scénario du cas d'utilisation.

Communication entre les cartes Nano ESP32

Entrons maintenant dans les détails techniques de la communication entre les deux cartes Nano ESP3 via Wifi dans le cadre de ce projet. Cette section vise à fournir une explication plus détaillée du processus de communication pour une meilleure compréhension.

Nano ESP32-1 (EDI Arduino)

Le Nano ESP32-1, qui fonctionne avec l'EDI Arduino, établit une connexion Wifi pour communiquer avec des services et des appareils externes. Il se connecte à un réseau Wifi spécifique avec des informations d'identification fournies. Une fois connecté, le Nano ESP32-1 attend une entrée de l'utilisateur sur le moniteur série de l'EDI Arduino.

Lorsque l'utilisateur tape GPT et appuie sur Entrée, le Nano ESP32-1 l'invite à saisir un message. Le message de l'utilisateur est ensuite envoyé par le Nano ESP32-1 à l'API ChatGPT pour traitement. Cette transmission s'effectue via la connexion Wifi établie, ce qui permet au Nano ESP32-1 de communiquer avec l'API externe. Dans la **figure 3**, vous pouvez voir l'ensemble de la communication et de la boucle de connexion entre les deux cartes et l'API ChatGPT.

L'API ChatGPT, une interface avec le modèle de langage d'IA, reçoit le message du Nano ESP32-1. À l'aide de techniques avancées de traitement du langage naturel et d'algorithmes d'apprentissage automatique, l'API analyse le message pour en comprendre le contexte et générer une réponse intelligente ou un extrait de code. Le modèle ChatGPT au sein de l'API exploite ses vastes données d'apprentissage et ses capacités de compréhension du langage pour fournir une réponse pertinente et captivante. Nano ESP32-1 reçoit l'extrait de code généré par l'API ChatGPT et le stocke en mémoire. Cet extrait de code représente la réponse générée par l'IA à la requête de l'utilisateur. Nano ESP32-1 affiche ensuite l'extrait de code reçu sur le moniteur série, ce qui permet aux utilisateurs de revoir les instructions générées par l'IA. Dans la **figure 4**, vous pouvez voir la sortie du moniteur série des deux cartes, où le Nano ESP32-1 envoie le code au Nano ESP32-2 après avoir reçu la réponse de ChatGPT.

Nano ESP32-2 (Framework MicroPython)

D'autre part, le Nano ESP32-2 fonctionne avec le *framework* MicroPython. Tout comme le Nano ESP32-1, le Nano ESP32-2 établit une connexion Wif sur le même réseau avec des informations d'identification fournies. Cela lui permet de recevoir des instructions sans fil de la part de Nano ESP32-1.

Nano ESP32-2 établit une connexion socket et écoute sur un port spécifique, typiquement le port 80. Un socket est un point d'extrémité logiciel qui permet la communication entre deux appareils sur un réseau. En écoutant sur un port spécifique, le Nano ESP32-2 est prêt à recevoir des données entrantes du Nano ESP32-1.

Lorsque le Nano ESP32-1 souhaite envoyer l'extrait de code généré, il établit une connexion avec le Nano ESP32-2 via la connexion socket. L'extrait de code est alors transmis au Nano ESP32-2, où il est reçu et traité.

Nano ESP32-2 traite l'extrait de code reçu et en extrait les instructions. Ces instructions définissent des tâches spécifiques à exécuter par le Nano ESP32-2. Par exemple, l'extrait de code peut demander au Nano ESP32-2 de faire

Listage 2. Recevoir et exécuter du code sur la deuxième carte (Python).

```
# Create a socket server
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.bind(('', 80))
server.listen(1)

# Accept and handle incoming connections
while True:
 print("Waiting for connection...")
 client, addr = server.accept()
 print("Client connected:", addr)

 # Receive the code snippet from the first ESP32
 code = ""
 while True:
 data = client.recv(1024)
 if not data:
 break
 code += data.decode()

 # Execute the received code
 try:
 exec(code)
 response = "Code executed successfully"
 print (response)
 except Exception as e:
 response = "Error executing code: " + str(e)

 # Send the response back to the first ESP32
 client.sendall(response.encode())
 # Close the connection
 client.close()
 print("Client disconnected")
```

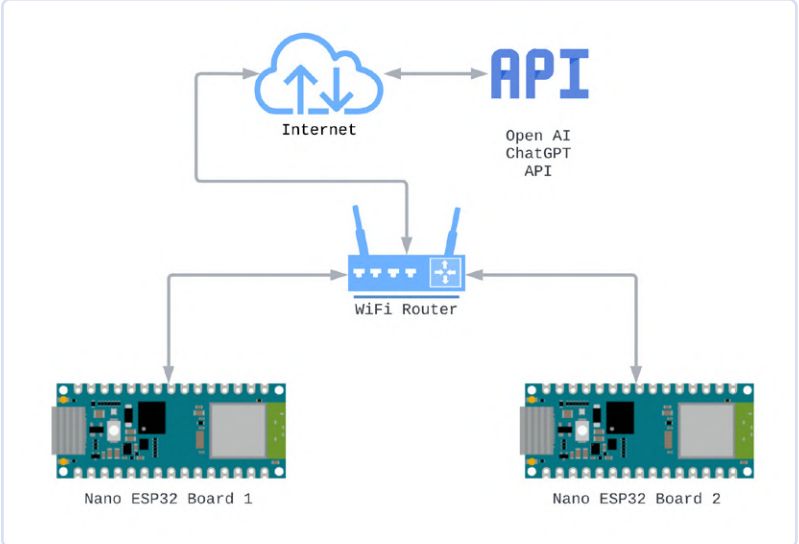


Figure 3. Communication entre les deux cartes Arduino Nano ESP32 et l'API ChatGPT.

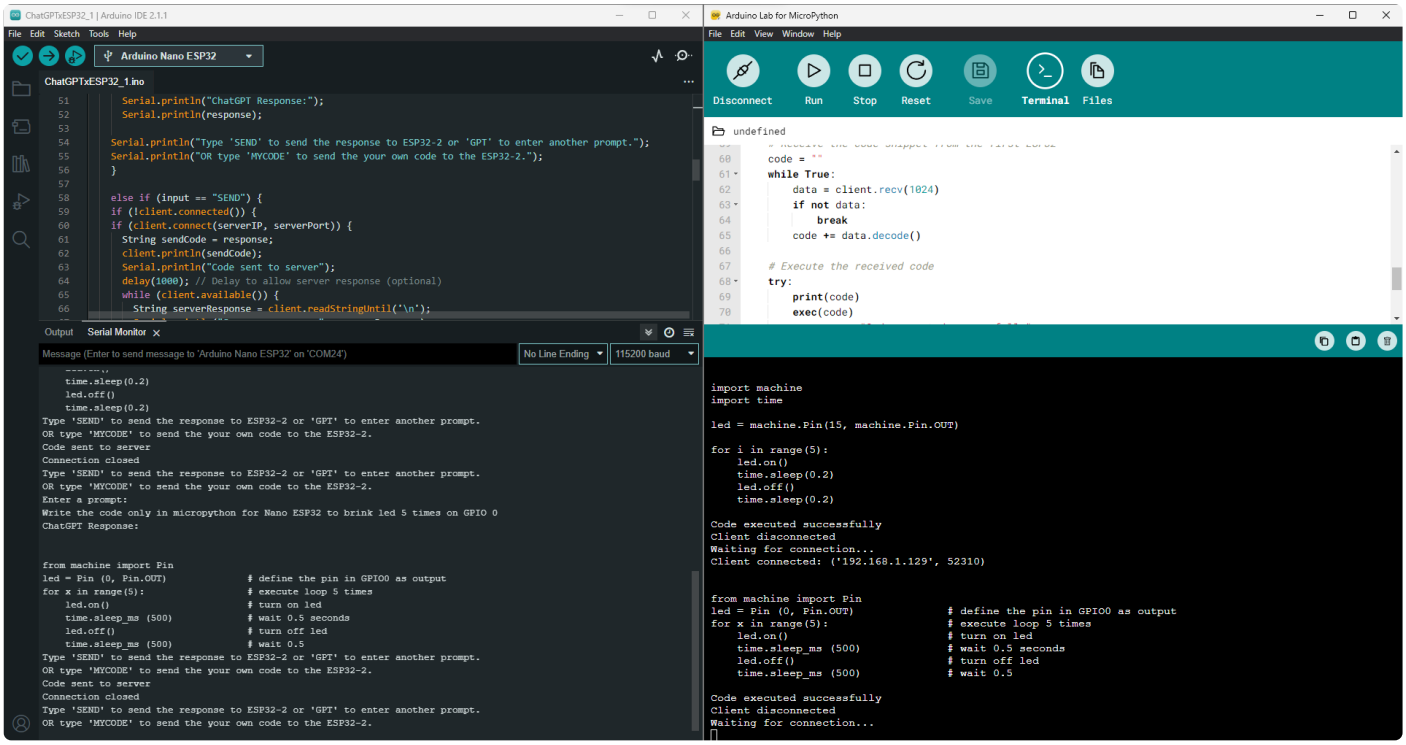


Figure 4. Communication entre les deux Nano ESP32, EDI Arduino (à gauche) et Arduino Lab pour MicroPython (à droite).

clignoter une LED pendant une certaine durée ou d'effectuer d'autres actions en fonction de la réponse générée par l'IA.

Après avoir exécuté les instructions, le Nano ESP32-2 envoie une réponse au Nano ESP32-1 via la connexion socket établie. Cette réponse sert à confirmer que l'extrait de code reçu a été exécuté avec succès. Elle garantit que le Nano ESP32-1 est au courant de l'achèvement de la tâche demandée.

En suivant ce processus de communication, les deux cartes Nano ESP32 peuvent échanger des messages et communiquer efficacement. Le Nano ESP32-1 interagit avec l'API ChatGPT, en tirant parti de ses capacités d'IA, tandis que le Nano ESP32-2 agit en tant que destinataire et exécuteur des instructions générées par l'IA.

Limites de la programmation avec ChatGPT

Bien que ChatGPT soit un modèle de langage d'IA remarquable, il présente certaines limites lorsqu'il s'agit de générer du code fonctionnel, en particulier dans des scénarios de programmation plus complexes. Lors des tests, on a constaté que ChatGPT n'était en mesure de fournir un code fonctionnel pour un simple croquis de clignotement que dans 60 % des cas. Cela indique que la capacité du modèle à générer des extraits de code précis et fonctionnels n'est pas toujours garantie.

Un problème se pose lorsque l'on demande des extraits de code à ChatGPT. Parfois, la réponse inclut du texte explicatif avant et après le code, ce qui peut poser des problèmes lorsque l'on tente d'envoyer l'intégralité de la réponse à la seconde carte Nano ESP32. Ce texte peut manquer de formatage ou de syntaxe, ce qui le rend incompatible lors d'une exécution directe. Dans la **figure 5**, vous pouvez voir la réponse de ChatGPT après lui avoir demandé de fournir un code pour vérifier la valeur du capteur sur la broche 36 de l'ESP32.

Pour surmonter cette limite , nous avons intégré une fonction supplémentaire dans le code, qui permet aux utilisateurs de saisir eux-mêmes le code et d'utiliser ChatGPT comme référence ou guide pour optimiser et réduire le nombre de lignes de code. Cette approche permet de mieux contrôler le code généré et de résoudre les problèmes de formatage et de syntaxe rencontrés lorsque l'on se fie uniquement à la réponse de ChatGPT.

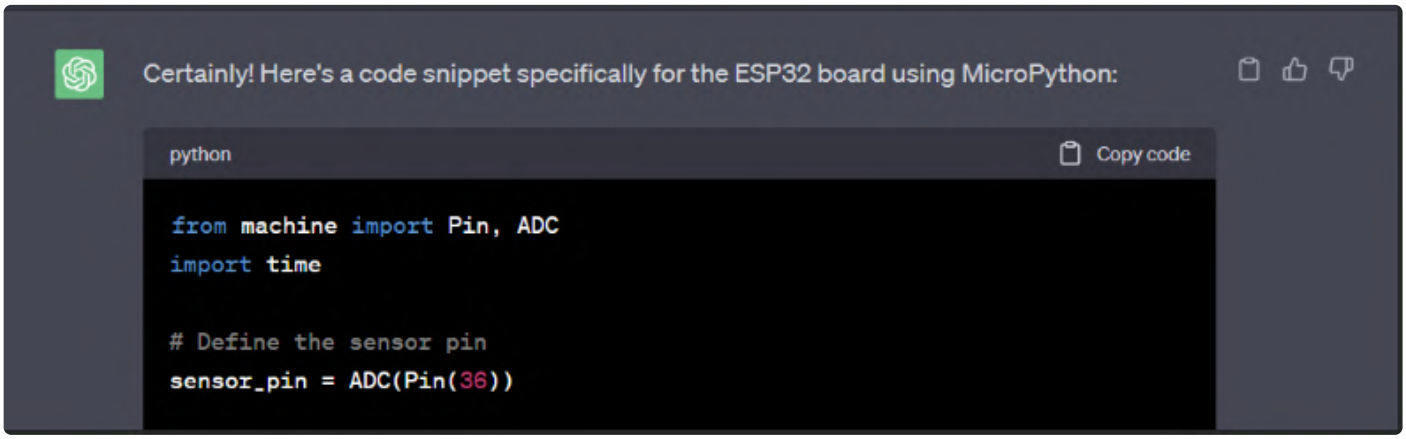


Figure 5. Réponse de ChatGPT après l'avoir invité à écrire un extrait de code.

Applications et cas d'utilisation

Le projet d'intégration des cartes Nano ESP32 avec l'API ChatGPT permet de réaliser toute une série d'applications et de cas d'utilisation intéressants. Voici quelques exemples notables :

- Domotique intelligente : en exploitant l'API ChatGPT, les utilisateurs peuvent interagir avec leurs systèmes domotiques intelligents en utilisant des messages en langage naturel : ils peuvent contrôler les lumières, ajuster les paramètres de température ou même demander des conseils sur les pratiques d'économie d'énergie.
- Prototypage et développement rapide : la combinaison des cartes Nano ESP32 et des extraits de code générés par l'IA permet un prototypage rapide des projets IdO. Les utilisateurs peuvent rapidement tester et développer leurs idées en recevant des suggestions de code instantanées pour différentes fonctions.
- Outil pédagogique : le projet constitue un outil pédagogique précieux pour les débutants en programmation. Les étudiants peuvent participer à des conversations interactives sur la programmation avec l'IA, recevoir des conseils et apprendre de nouvelles techniques.
- Assistant personnel et recherche d'informations : l'intégration de ChatGPT peut être utilisée pour développer une application d'assistant personnel. Les utilisateurs peuvent poser des questions, demander des recommandations ou obtenir des informations sur divers sujets, le tout grâce à des réponses pilotées par l'IA.
- Inspiration en matière de codage : le projet sert de source d'inspiration pour le codage créatif. Il peut générer des idées uniques et innovantes pour des animations, des visualisations ou des projets interactifs, en stimulant la créativité des développeurs.

Il est important de souligner que si ChatGPT peut fournir des informations et des suggestions précieuses, il peut encore s'améliorer dans le domaine de la programmation. L'utilisation de ChatGPT comme référence ou pour l'optimisation du code peut améliorer le de développement, mais elle doit être complétée par une expertise humaine et une révision approfondie du code.

À mesure que les modèles de langage de l'IA continuent d'évoluer, nous pouvons nous attendre à des améliorations dans leur capacité à générer un code plus précis et plus fiable. Ces progrès ouvriront de nouveaux horizons pour utiliser l'IA dans la programmation et permettront une collaboration encore plus étroite entre les humains et les machines. Pour tirer le meilleur parti de ChatGPT, il est important de l'utiliser comme un outil et de ne pas se fier uniquement à ses résultats. La combinaison de l'expertise et du jugement humains avec les réponses générées par l'IA peut conduire à des résultats plus précis et plus fiables. En comprenant et en tenant compte de ces limites, les utilisateurs peuvent utiliser efficacement ChatGPT tout en atténuant les risques et les défis potentiels.

Explorer d'autres approches

En plus du projet décrit précédemment, il existe d'autres méthodes et approches qui pourraient être utilisées pour réaliser la communication Nano ESP32 et l'intégration de l'IA. Explorons les différentes façons dont ce projet peut être implémenté sur la plateforme Nano ESP32 :

1. **Protocole MQTT**
   Le protocole MQTT (*Message Queuing Telemetry Transport*) est un protocole de messagerie léger et efficace couramment utilisé dans les applications IdO. Au lieu de recourir au Wifi et aux connexions socket, les cartes Nano ESP32 peuvent communiquer entre elles grâce au protocole MQTT. Il est possible de configurer des broker MQTT pour faciliter la communication entre les cartes, ce qui permet un échange de données transparent et une intégration de l'IA.
2. **Connexion directe par Web Socket**
   Une autre approche consiste à établir une connexion Web Socket directe entre les cartes Nano ESP32. Web Socket est un protocole de communication qui permet une communication en duplex intégral sur une seule connexion TCP. En implémentant Web Socket sur les cartes Nano ESP32, il est possible d'établir une connexion continue et bidirectionnelle, ce qui permet une communication en temps réel et une intégration de l'IA.
3. **Protocole ESP-NOW**
   ESP-NOW est un protocole de communication spécialement conçu pour les appareils à faible consommation, qui permet une transmission rapide et fiable des données entre les cartes Nano ESP32. Grâce à ce protocole, il est possible d'établir une communication directe entre les cartes, sans avoir recours à des connexions Wifi. En intégrant des capacités d'IA sur une carte et en utilisant ESP-NOW pour la communication, les réponses en temps réel et les extraits de code peuvent être échangés efficacement.
4. **Intégration de l'IA sur l'appareil**
   Au lieu de dépendre d'API externes, il est possible de déployer des modèles d'IA directement sur les cartes Nano ESP32. TensorFlow Lite, par exemple, permet de déployer et d'exécuter localement des modèles d'IA sur des microcontrôleurs. En entraînant ou en optimisant un modèle pour des tâches spécifiques, telles que la génération d'extraits de code, les cartes Nano ESP32 peuvent fournir des réponses pilotées par l'IA sans nécessiter de connexions externes.

Ces méthodes alternatives offrent des avantages et des considérations différents en fonction des exigences et des contraintes spécifiques du projet. Il est important de prendre en compte des facteurs tels que la consommation d'énergie, la latence, la complexité et l'évolutivité lors du choix de l'approche la plus appropriée.

En explorant ces méthodes alternatives, les développeurs peuvent étendre les capacités des cartes Nano ESP32, intégrer l'IA à différents niveaux et adapter le système de communication à leurs propres besoins. Le projet de communication Nano ESP32 et d'intégration de l'IA n'est pas limité à une seule approche. En envisageant des méthodes alternatives telles que MQTT, Web Socket, ESP-NOW ou l'intégration de l'IA sur l'appareil, les développeurs peuvent personnaliser le projet pour répondre à leurs exigences uniques et tirer parti de tout le potentiel de la plate-forme Nano ESP32. Alors, libérez votre créativité, expérimentez différentes méthodes et construisez des systèmes IdO innovants qui combinent l'IA et le Nano ESP32.

EDI Arduino, MicroPython, et plus encore

Arduino IDE Framework

L'EDI Arduino est un choix populaire pour les débutants et les amateurs en raison de sa simplicité et de sa facilité d'utilisation. Il fournit un environnement de programmation convivial avec une version simplifiée du langage de programmation C++. Voici quelques avantages et inconvénients de l'utilisation de l'EDI Arduino avec la Nano ESP32 :

Avantages

- Facile à apprendre : l'EDI Arduino offre une facilité d'apprentissage, ce qui le rend accessible aux débutants en programmation ou en microcontrôleurs.
- Grande communauté et support de bibliothèque : Arduino a une vaste communauté de passionnés, des ressources en ligne étendues et une large gamme de bibliothèques et d'exemples disponibles, ce qui simplifie le développement.
- Prototypage rapide : l'EDI Arduino permet un prototypage rapide grâce à sa syntaxe simplifiée et à la prise en charge des bibliothèques, ce qui accélère les cycles de développement.

Inconvénients

- Langage limité : l'EDI Arduino dispose d'une version simplifiée de C++, ce qui peut limiter l'utilisation de fonctions de programmation avancées par rapport à d'autres langages.
- Mémoire et performance : l'EDI Arduino est parfois moins performant en termes d'utilisation de la mémoire et de performances que d'autres frameworks.
- Moins de flexibilité : l'EDI Arduino impose certaines conventions et limitations qui peuvent restreindre le plein potentiel du Nano ESP32.

Framework MicroPython

MicroPython est une implémentation légère du langage de programmation Python conçue pour les microcontrôleurs. Il offre une expérience de programmation plus flexible et interactive. Voici quelques avantages et inconvénients de l'utilisation du framework MicroPython avec le Nano ESP32 :

Avantages

- Langage Python : MicroPython permet aux développeurs d'exploiter le langage Python puissant, ce qui facilite l'écriture de codes et d'algorithmes complexes.
- REPL interactif : MicroPython fournit un environnement Read-Eval-Print Loop (REPL), permettant aux développeurs de tester et d'expérimenter le code de manière interactive directement sur la carte Nano ESP32.
- Utilisation efficace de la mémoire : MicroPython est conçu pour utiliser efficacement la mémoire, ce qui le rend adapté aux appareils à ressources limitées tels que le Nano ESP32.

Inconvénients

- Courbe d'apprentissage : MicroPython peut être plus difficile à apprendre pour les débutants qui ne sont pas familiers avec le langage Python.
- Support de bibliothèques limité : bien que MicroPython dispose d'une collection en constante évolution de bibliothèques, il peut avoir moins d'options que l'écosystème étendu de bibliothèques Arduino.
- Compromis de performance : bien que MicroPython offre une grande flexibilité, la nature interprétée du langage peut entraîner une exécution légèrement plus lente que les langages compilés tels que C++.

Le croisement fascinant de l'IA et de l'IdO

L'intégration des cartes Nano ESP32 avec l'API ChatGPT représente un croisement fascinant entre l'IA, l'IdO et la programmation. En permettant aux cartes Nano ESP32 de communiquer et d'interagir avec un modèle de langage d'IA, nous créons un éventail de possibilités. De la domotique au prototypage rapide et aux outils éducatifs, les applications sont vastes (voir l'encadré **Cas d'utilisation**).

Toutefois, il est essentiel de garder à l'esprit les limites de ChatGPT et de toujours rester prudent lorsque l'on travaille avec des réponses générées par l'IA, en particulier dans les scénarios de programmation et de codage. La combinaison de l'expertise et du jugement humains avec le contenu généré par l'IA produira de meilleurs résultats.

Avec ce projet, les utilisateurs peuvent libérer leur créativité, explorer de nouveaux horizons dans le développement IdO et améliorer leurs compétences en programmation. Alors, prenez vos cartes Nano ESP32, plongez dans le monde des interactions alimentées par l'IA et profitez du potentiel illimité de cette intégration passionnante !

230485-04

Questions ou commentaires ?

Contactez Elektor (redaction@elektor.fr).

À propos de l'auteur

Saad Imtiaz est un ingénieur expérimenté dans les systèmes embarqués, les systèmes mécatroniques et le développement de produits. Il a collaboré avec plus de 200 entreprises, allant des startups aux entreprises mondiales, sur le prototypage et le développement de produits. Saad a également travaillé dans l'industrie aéronautique et a dirigé une startup technologique. Il a récemment rejoint Elektor en 2023 et participe au développement de projets logiciels et matériels.

Autres environnements

L'un des avantages notables du Nano ESP32 est sa flexibilité à fonctionner avec différents frameworks, outre l'EDI Arduino, MicroPython, il y a PlatformIO, ESP-IDF, JavaScript (Node.js), et les frameworks Lua. Le Nano ESP32 est compatible avec plusieurs autres frameworks, ce qui accroît encore sa polyvalence. Cette flexibilité permet aux développeurs de choisir le framework qui correspond le mieux aux exigences de leur projet et à leur familiarité avec les langages de programmation. Nous allons explorer quelques autres frameworks qui peuvent être utilisés avec le Nano ESP32 et leurs avantages :

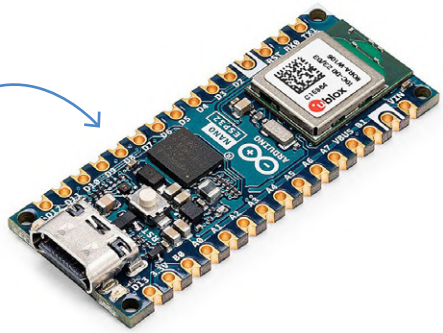
Mongoose OS : Mongoose OS est un système d'exploitation open-source pour microcontrôleurs, comme le Nano ESP32. Il fournit un environnement indépendant de la plateforme avec une connectivité cloud intégrée et prend en charge de nombreux langages de programmation tels que JavaScript, C et C++. Mongoose OS simplifie le processus de développement en offrant une interface de ligne de commande facile à utiliser, de riches bibliothèques et des capacités de contrôle à distance des appareils.

Zephyr RTOS : Zephyr RTOS est un système d'exploitation en temps réel conçu pour les systèmes à ressources limitées, y compris le Nano ESP32. Il offre une architecture évolutive et modulaire, ce qui le rend adapté aux projets qui nécessitent du multitâche, du traitement en temps réel et une gestion avancée des périphériques. Le large support matériel de Zephyr et son ensemble étendu de pilotes et de bibliothèques permettent aux développeurs de créer facilement des applications IdO complexes.

ESPHome : ESPHome, conçu pour les appareils ESP32 et ESP8266, est un framework IdO polyvalent doté d'une architecture modulaire proche du RTOS Zephyr. Il offre un large support matériel, un ensemble de pilotes et de bibliothèques pour simplifier le développement de systèmes à ressources limitées. Ce cadre facilite le traitement en temps réel, le multitâche et la gestion avancée des périphériques, ce qui en fait un excellent choix pour les projets IdO de domotique.

Le choix du bon framework dépend de facteurs tels que les exigences du projet, la familiarité avec le langage de programmation, les bibliothèques disponibles et le soutien de la communauté, ainsi que le niveau de contrôle souhaité sur le matériel et les performances. Chaque framework présente ses avantages et ses inconvénients, ce qui permet aux développeurs de choisir celui qui convient le mieux à leur cas d'utilisation spécifique.

Grâce à la compatibilité du Nano ESP32 avec différents framework, les développeurs ont la liberté d'explorer différents langages de programmation, écosystèmes et approches de développement. Cette flexibilité leur permet de créer des solutions IdO innovantes, d'exploiter les outils et les bibliothèques existants et d'utiliser efficacement les capacités du Nano ESP32.



Produits

- **Arduino Nano ESP32**
  www.elektor.fr/20562
- **Arduino Nano ESP32 avec connecteurs www.elektor.fr/20529**
- **Dogan Ibrahim and Ahmet Ibrahim,** *The Official ESP32 Book* **(E-book) (Elektor 2017)**
  www.elektor.fr/18330
- **Günter Spanner,** *MicroPython for Microcontrollers* **(Elektor 2021)**
  www.elektor.fr/19736

LIENS

[1] Open AI : https://platform.openai.com
[2] Open AI - API Keys : https://platform.openai.com/account/api-keys
[3] Dépôt GitHub de ce projet : https://github.com/elektor-labs/ChatGPTxESP32
[4] Arduino Labs - MicroPython Installer : https://labs.arduino.cc/en/labs/micropython-installer
[5] Arduino Labs - EDI MicroPython : https://labs.arduino.cc/en/labs/micropython

talkie-walkie avec ESP-NOW

pas tout à fait Wi-Fi, pas tout à fait Bluetooth non plus…

Clemens Valens (Elektor)

Imaginez que votre projet sans fil nécessite à la fois des temps de réponse rapides et une longue portée ? Le Wi-Fi et le Bluetooth ne conviennent pas dans ces cas. L'ESP-NOW est peut-être une bonne alternative ? Les connexions sont établies presque instantanément et des portées de plusieurs centaines de mètres sont possibles. Dans cet article, nous l'essayons dans une application simple de talkie-walkie ou d'interphone sans fil.

L'ESP32 d'Espressif est souvent utilisé pour ses compétences Wi-Fi et Bluetooth, des domaines dans lesquels il excelle. Ce sont d'excellents protocoles pour toutes sortes d'applications sans fil, mais ils ont leurs limites.

L'un des inconvénients du Wi-Fi est le temps nécessaire pour établir une connexion. En outre, il ne permet pas une communication directe (pair-à-pair, **figure 1**) entre les appareils. Un routeur est toujours impliqué. Pour cette raison, ce n'est pas vraiment adapté aux simples télécommandes à faible latence permettant d'ouvrir une porte de garage ou d'allumer et d'éteindre une lumière. Ces tâches nécessitent une réponse immédiate. Pour contourner ce problème, les applications Wi-Fi ont tendance à être allumées et connectées en permanence. Elles consomment donc beaucoup d'énergie, même lorsqu'elles sont inactives.

Le Bluetooth, quant à lui, se caractérise par un établissement rapide de la connexion et une communication de pair-à-pair (aussi poste-à-poste ou *peer-to-peer* en anglais, ou encore P2P), et est excellent pour les télécommandes à faible latence. Toutefois, il est destiné à des applications à courte portée, avec des appareils communicants espacés d'une dizaine de mètres. Il est vrai que le Bluetooth à longue portée existe, mais il n'est pas encore très répandu.

La solution : ESP-NOW

Le protocole sans fil ESP-NOW [1] d'Espressif est une solution pour les situations qui nécessitent à la fois des temps de réponse rapides et une longue portée tout en utilisant la même bande de fréquence que le Wi-Fi et le Bluetooth.

Le protocole combine les avantages du Wi-Fi et du Bluetooth. L'ESP-NOW est destiné à la domotique et à la maison intelligente. Comme il permet des topologies un-à-plusieurs (*one-to-many*) et plusieurs-à-plusieurs (*many-to-many*) (**figure 2**), il n'a besoin ni de routeur, ni de passerelle, voire pire, de cloud.

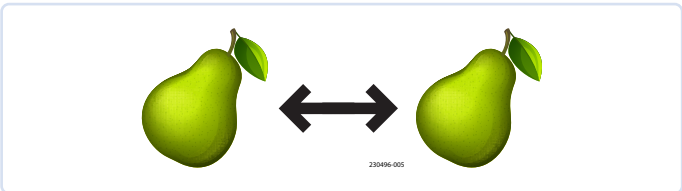


Figure 1. La communication de pair-à-pair telle qu'illustrée ici n'est pas possible avec le Wi-Fi ; un routeur est toujours nécessaire entre les deux nœuds.

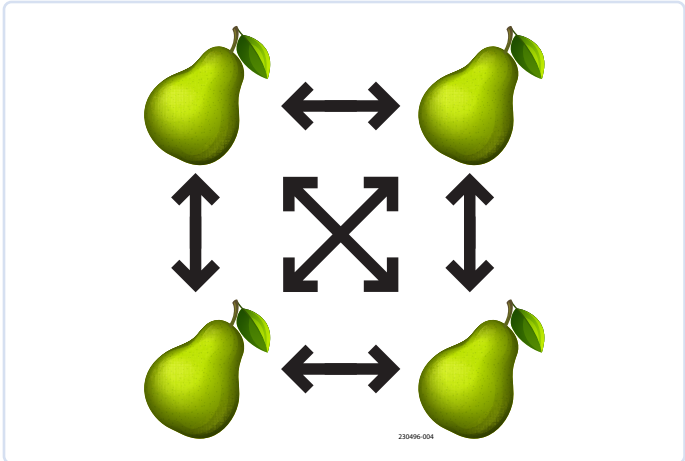


Figure 2. L'ESP-NOW prend en charge les réseaux de plusieurs-à-plusieurs. Dans un tel réseau, chaque nœud peut communiquer directement avec les autres nœuds sans passer par un routeur.

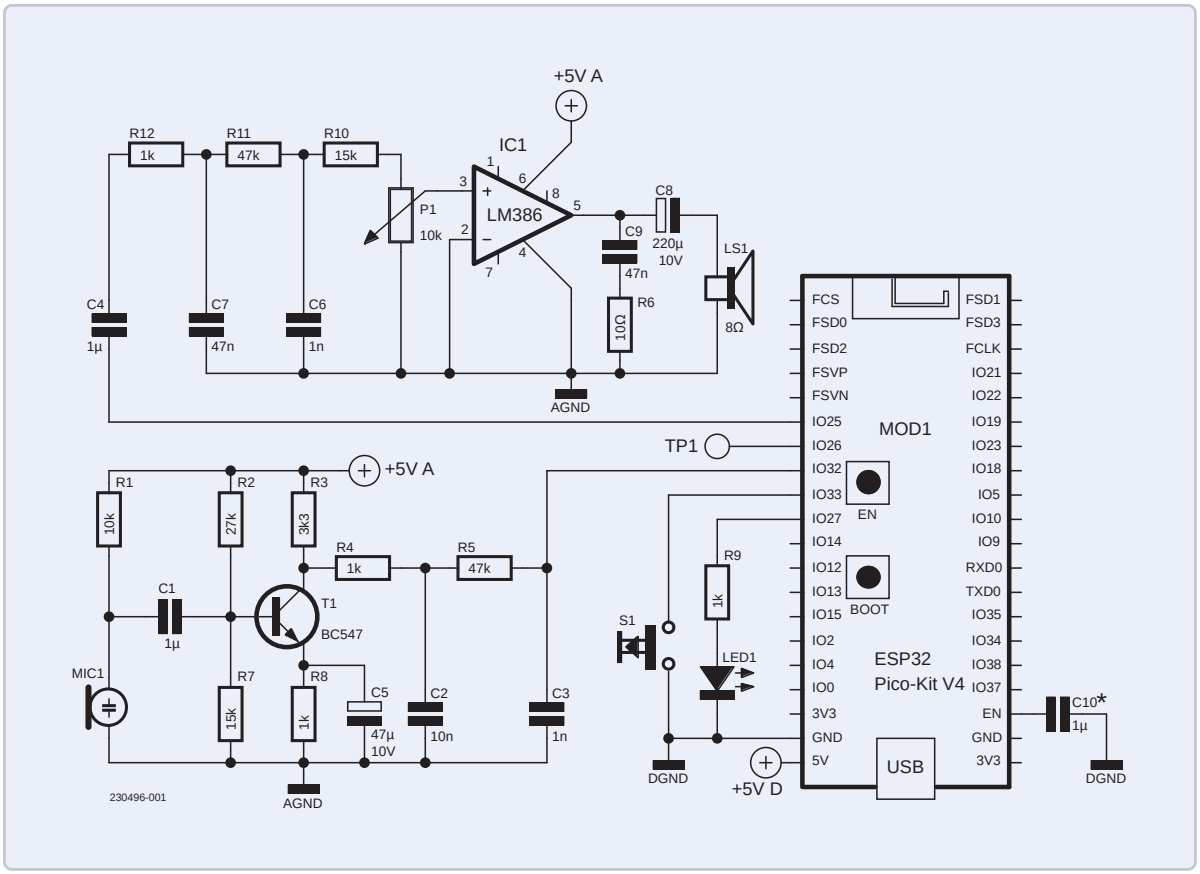


Figure 3. Un simple préamplificateur de microphone à l'entrée et un LM386 classique comme amplificateur de puissance à la sortie. Notez comment les alimentations des parties analogiques et numériques sont séparées.

L'ESP-NOW ne met pas en œuvre de connexion fantaisiste ou de protocoles de communication complexes. L'adressage est basé sur l'adresse MAC Ethernet du nœud, et une étape d'appariement est nécessaire pour que les nœuds puissent communiquer entre eux. De plus, il n'est pas garanti que les paquets de données arrivent dans l'ordre. Pour des applications simples de commande à distance, tout cela suffit.

Le débit de données de l'ESP-NOW est de 1 Mbit/s par défaut (configurable), et un paquet de données peut avoir une charge utile allant jusqu'à 250 octets. Avec les octets d'en-tête et de somme de contrôle, etc., cela donne une taille maximale de paquet de 255 octets.

Construisons un talkie-walkie

Mon objectif était de créer un appareil de type talkie-walkie ou un interphone basé sur l'ESP-NOW. Un rapide coup d'œil aux spécifications de l'ESP32 montre qu'il intègre tout ce qu'il faut pour cela : un convertisseur analogique-numérique (CAN), un convertisseur numérique-analogique (CNA), beaucoup de puissance de calcul et, bien sûr, tous les éléments de la radio. Dans la pratique, cependant, les choses sont un peu moins roses.

Le CAN de 12 bits s'avère plutôt lent, j'ai mesuré une fréquence d'échantillonnage maximale d'environ 20 kHz. Quelque part sur internet, il a été mentionné que sa bande passante analogique n'est que de 6 kHz. Le CNA a une largeur de huit bits (mais il y en a deux), ce qui limite encore plus la qualité audio possible.

Cependant, un talkie-walkie peut s'en sortir avec ces chiffres si la bande passante audio est limitée à la bande passante standard de téléphonie de 3,5 kHz. Une fréquence d'échantillonnage de 8 kHz donne un débit de données de (8 000 / 250) × 255 × 8 = 65 280 bit/s (n'oubliez pas que la taille maximale de la charge utile est de 250 octets). C'est bien inférieur au débit par défaut de 1 Mbit/s. Ces spécifications ne nous permettront pas d'obtenir un son très fidèle, mais ce n'est de toute façon pas notre objectif. L'intelligibilité est plus importante.

Le circuit

Pour garder les choses simples, j'ai utilisé un préamplificateur de microphone à condensateur à bande limitée à un transistor comme entrée audio et j'ai ajouté un amplificateur classique à base de LM386 comme sortie audio. Le schéma est montré dans la **figure 3**. La bande passante d'entrée est limitée à l'extrémité basse par C1 et C5, qui sont légèrement sous-dimensionnés. L'extrémité haute est limitée par les filtres passe-bas R4/C2 et R5/C3. Des filtres passe-bas similaires sont placés à la sortie du CNA. Le signal sur le côté chaud de P1 ne doit pas être supérieur à 400 mV_{PP}.

Comme module ESP32, j'ai opté pour l'ESP32-PICO-KIT. Il existe de nombreux autres modules, mais ils n'exposent pas toutes les sorties CNA sur GPIO25 et GPIO26. Nous avons également besoin d'une entrée CAN. J'ai utilisé GPIO32 pour cela, qui correspond à ADC1, canal 4. Le point de test TP1 sur GPIO26 (la deuxième sortie CNA) est fourni comme une sortie de contrôle pour le signal du microphone. Un bouton poussoir sur GPIO33 fournit la fonctionnalité *push-to-talk* (PTT, appuyer pour parler en français), et la LED sur GPIO27 est la LED multifonction obligatoire d'un circuit à microcontrôleur.

Notez que l'alimentation est divisée en une partie analogique et une partie numérique. La raison n'est pas d'éviter le couplage du bruit de commutation numérique à haute vitesse dans l'entrée audio, mais d'éviter un cliquetis dans la sortie. Apparemment, une tâche exécutée sur l'ESP32 produit des perturbations périodiques qui peuvent devenir audibles lorsque le circuit n'est pas câblé avec soin. La meilleure façon que j'ai trouvée pour éviter cela est d'utiliser deux alimentations séparées (**figure 4**). Le module ESP32 doit être traité comme un composant qui a besoin d'une alimentation (tout comme le LM386), et non comme un module qui peut aussi fournir de l'énergie au reste du circuit ; dans cette application, il ne le peut pas. Gardez à l'esprit que le LM386 a une plage d'alimentation de 4 V à 12 V.

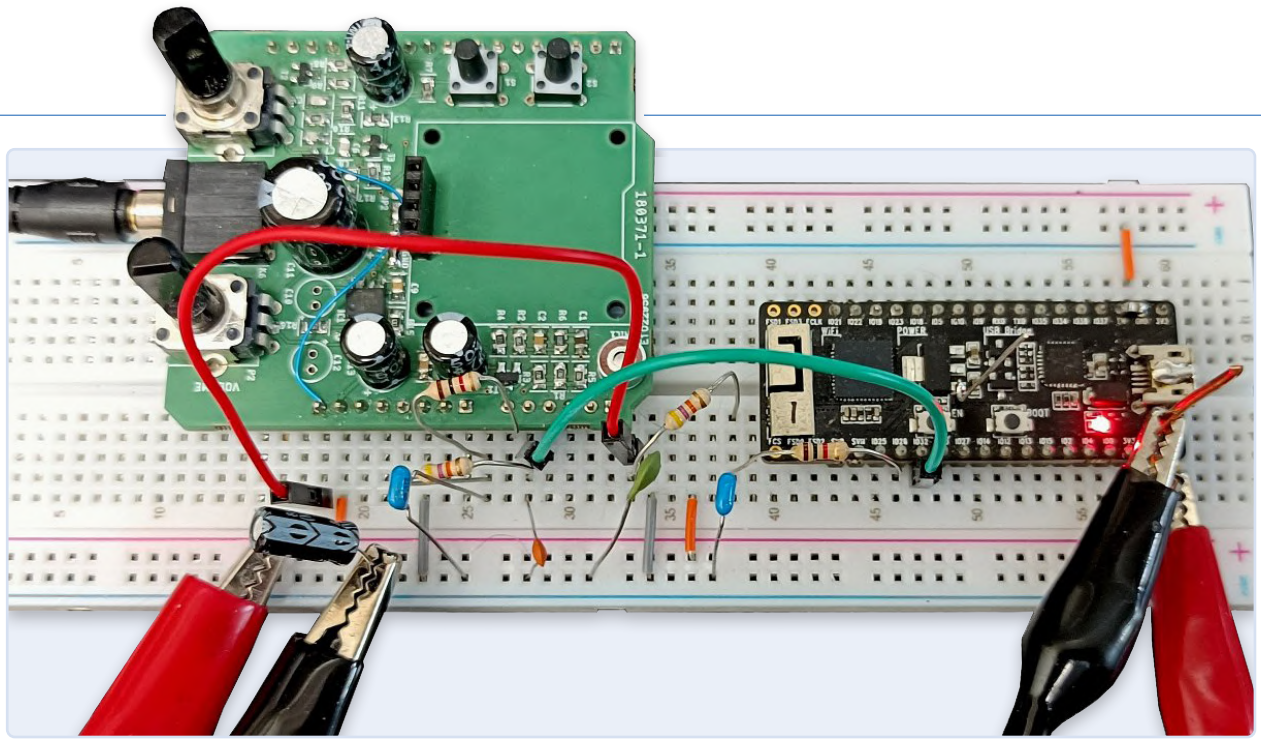


Figure 4 . Une preuve de concept construite sur une carte avec un ESP32-PICO-KIT et un Elektor Snore Shield [2] légèrement modifié pour les amplificateurs d'entrée et de sortie. Notez les deux paires de pinces crocodiles qui fournissent les alimentations analogiques et numériques séparées.

Le condensateur C10 est optionnel et n'est nécessaire que dans quelques rares cas de modules ESP32 anciens qui ne démarrent pas correctement lorsqu'ils ne sont pas connectés à un ordinateur (ou autre). Il se trouve que j'ai quelques-uns de ces premiers modules, et j'ai donc inclus le C10 dans mon design.

Le logiciel

J'ai basé le programme du talkie-walkie sur l'exemple `ESPNow_Basic_Master` fourni dans la bibliothèque d'Espressif pour Arduino. Après l'avoir adapté à mes besoins, j'y ai ajouté l'échantillonnage et la lecture audio. Il y a plusieurs choses que vous voudrez peut-être savoir à propos du programme.

L'échantillonnage et la lecture audio sont contrôlés par une interruption de minuterie fonctionnant à 8 kHz. Pour l'échantillonnage, la routine de service d'interruption (ISR) de la minuterie lève uniquement un drapeau pour signaler qu'un nouvel échantillon doit être acquis. La fonction `loop()` surveille ce drapeau et prend les mesures nécessaires. En effet, le CAN ne doit pas être lu à l'intérieur d'un ISR lors de l'utilisation de la bibliothèque CAN fournie par Espressif. La fonction `adc1_get_raw()` utilisée ici appelle toutes sortes d'autres fonctions qui peuvent faire des choses sur lesquelles vous n'avez aucun contrôle. Comme le logiciel ESP32 fonctionne dans un environnement multi-tâche, il est important d'assurer la sécurité des fils d'exécution. Lorsque vous utilisez Arduino pour la programmation de l'ESP32, beaucoup de choses sont gérées pour vous, mais si vous envisagez de porter mon programme sur l'ESP-IDF, vous devrez peut-être être plus prudent.

La lecture audio est facile, car l'ISR de la minuterie d'échantillonnage écrit simplement un échantillon sur le CNA s'il y en a un de disponible. Si ce n'est pas le cas, il fixe la sortie du CNA à la moitié de l'alimentation de l'ESP32, soit 1,65 V. La seule chose à savoir ici est qu'un tampon dit ping-pong est utilisé pour rationaliser la réception audio numérique (**figure 5**). Un tel tampon se compose de deux tampons, dont l'un est rempli pendant que l'autre est lu. Cela permet un chevauchement. En théorie, cela ne devrait pas se produire puisque l'émetteur et le récepteur utilisent la même fréquence d'échantillonnage et la même logique de synchronisation, mais en réalité cela se produit en raison des tolérances de synchronisation. Un tampon double ou ping-pong permet d'éviter des clics gênants pendant la lecture. Notez que la réception de paquets de données dans le désordre n'est pas gérée.

Couplage

Le micrologiciel du talkie-walkie est un système maître-esclave. Le maître fonctionne en mode station Wi-Fi (STA), tandis que l'esclave est en mode point d'accès (AP). Le maître se connecte immédiatement à un esclave lorsqu'il en détecte un, et il peut commencer à envoyer des données immédiatement. Toutefois, lorsque le maître se connecte à l'esclave, ce dernier ne se connecte pas au maître. L'esclave ne peut pas envoyer de données au maître et la communication bidirectionnel n'est pas possible (du moins, je n'y suis pas parvenu ; si vous savez mieux, faites-le moi savoir).

Pour que l'esclave se connecte au maître, il est possible d'utiliser la fonction de rappel de réception de données. Lorsque des données sont reçues, l'adresse de l'expéditeur est transmise à cette fonction en même temps que les données. Ainsi, dès que l'esclave reçoit quelque chose, il peut se connecter à l'expéditeur des données. Pour ce faire, j'ai utilisé les mêmes fonctions et procédures que celles utilisées par

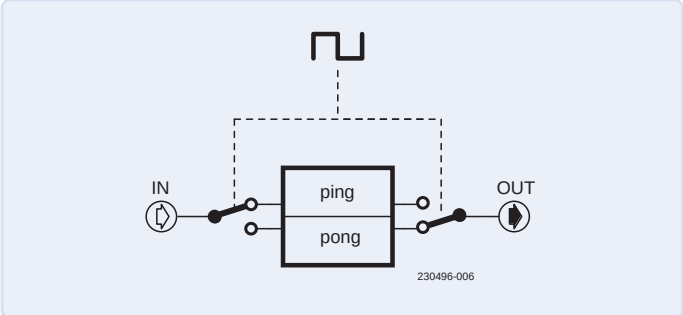


Figure 5. La mise en mémoire tampon double ou ping-pong permet d'éviter les discontinuités dans un flux de données.

le maître pour se connecter à l'esclave. Il y a cependant une subtilité qui n'est pas très bien (voire pas du tout) documentée : l'esclave doit définir son champ d'interface Wi-Fi sur `ESP_IF_WIFI_AP`, sinon il ne fonctionnera pas. Ce champ prend par défaut la valeur `ESP_IF_WIFI_STA`, selon les besoins du maître, et il n'a donc pas besoin de le définir explicitement. Par conséquent, ce champ n'apparaît nulle part dans les exemples et nous, les utilisateurs, ignorons qu'il existe.

Push-to-Talk

Lorsque l'ESP-NOW émet en continu, le microcontrôleur chauffe beaucoup. Dans l'application talkie-walkie, il n'y a aucune raison de diffuser en continu, et j'ai donc ajouté un bouton *push-to-talk* (alias PTT, « appuyer pour parler ») (S1). Appuyez sur ce bouton et maintenez-le enfoncé pendant la conversation. Si l'émetteur est apparié au récepteur, la LED s'allume. Du côté du récepteur, la LED s'allume également, indiquant qu'un appel est en cours. Pour éviter tout retour audio, la sortie audio du côté de l'émetteur est coupée lorsque le bouton PTT est enfoncé. Par conséquent, même si la communication est en principe full-duplex, les deux interlocuteurs ne doivent pas essayer de parler en même temps. C'est une excellente occasion de terminer toutes vos phrases avec « à vous ».

Un seul programme pour tous

Le programme consiste en un fichier Arduino *.ino* (croquis ou *sketch*). En dehors de la bibliothèque ESP32 d'Espressif, aucune autre bibliothèque n'est nécessaire.

Le talkie-walkie a besoin d'un appareil maître et d'un appareil esclave. Pour compiler le programme pour un dispositif maître, mettez en commentaire la ligne 12, qui indique `NODE_TYPE_SLAVE`. Pour le dispositif esclave, cette macro doit être définie. Vous pouvez également reconfigurer d'autres paramètres si vous le souhaitez. Il est également possible de compiler sans support d'entrée (`AUDIO_SOURCE`) et/ou de sortie (`AUDIO_SINK`) audio. Ceci est pratique pour le débogage ou pour une application qui ne nécessite qu'une communication unidirectionnelle. Le code source peut être téléchargé ici [3].

Une plus grande fidélité ?

Il ne devrait pas être trop compliqué de transmettre des données audio de bonne qualité avec le protocole ESP-NOW si, au lieu d'utiliser le simple amplificateur de microphone et les convertisseurs CAN et CNA intégrés de l'ESP32, vous passez à l'I²S. Cela rend le circuit et le programme un peu plus complexes, mais permettrait, en théorie, de diffuser des données audio de 16 bits à une fréquence d'échantillonnage de 48 kHz. Toutefois, la réception éventuelle de paquets dans le désordre doit être gérée correctement. Mais bon, le Bluetooth n'a-t-il pas été conçu pour cela ?

Test de portée

Pour voir si l'ESP-NOW permet une communication à longue distance, j'ai écrit un programme simple pour envoyer un message ping à l'esclave une fois par seconde. L'esclave n'est rien d'autre qu'un ESP32-PICO-KIT avec une LED connectée à un GPIO27, alimenté par une power bank USB. Chaque fois qu'un ping est reçu, la LED clignote brièvement (100 ms).

Avec l'émetteur placé à l'extérieur à 1 m au-dessus du sol, j'ai obtenu une distance de communication en visibilité directe (LOS) d'environ 150 m. À cette distance, la réception est devenue intermittente et l'esclave a dû être tenu en hauteur (environ 2 m au-dessus du sol). Cette situation peut probablement être améliorée en positionnant (et en concevant) soigneusement les deux modules.

VF : Maxime Valens — 230496-04

Questions ou commentaires ?

Vous avez des questions techniques ou des commentaires sur cet article ? Envoyez un courriel à l'auteur à l'adresse clemens.valens@elektor.com ou contactez Elektor à l'adresse redaction@elektor.fr.

À propos de l'auteur

Après une carrière dans l'électronique marine et industrielle, Clemens Valens a commencé à travailler pour Elektor en 2008 en tant que rédacteur en chef d'Elektor France. Il a occupé différents postes depuis lors et a récemment rejoint le département de développement de produits. Ses principaux centres d'intérêt sont le traitement du signal et la génération de sons.

Produits

- **ESP32-PICO-KIT V4**
  www.elektor.fr/esp32-pico-kit-v4
- **Elektor ESP32 Smart Kit Bundle**
  www.elektor.fr/elektor-esp32-smart-kit-bundle

LIENS

[1] En savoir plus sur l'ESP-NOW : https://espressif.com/en/solutions/low-power-solutions/esp-now
[2] Le shield anti-ronflement d'Elektor : https://www.elektormagazine.fr/magazine/elektor-71/42319
[3] Téléchargements pour cet article : https://www.elektormagazine.fr/230496-04

de l'idée au circuit avec l'ESP32-S3

construire un prototype avec les puces Espressif

Liu Jing Hui, Espressif

De nombreux produits et appareils – en particulier les exemplaires uniques et ceux produits en faible quantité – qui intègrent des puces Espressif sont initialement construits avec des cartes de développement. Les développeurs passent en suite à l'utilisation d'un module ou d'une puce nue s'ils ont besoin de produire un plus grand nombre de produits ou si le produit doit simplement être plus esthétique. Plusieurs facteurs doivent être pris en compte lors de cette migration. Dans cet article, nous abordons les risques les plus courants.

Nous prenons l'ESP32-S3 comme exemple de conception avec les puces Espressif. Cette puce contient une solution intégrée de système sur puce (SoC) à faible consommation, WiFi 2,4 GHz + Bluetooth LE (5). Elle dispose de deux cœurs rapides, d'une mémoire vive extensible en externe et d'une multitude d'interfaces qui prennent en charge de nombreuses applications. Il est important de noter que le SoC dispose également d'un matériel d'étalonnage intégré permettant de compenser les légers défauts de la configuration radio. Il est donc relativement facile de faire fonctionner votre circuit et il n'est pas nécessaire de recourir à un équipement de test spécialisé. L'ESP32-S3 est un choix idéal pour un large éventail d'applications d'intelligence artificielle (IA) et d'IA des objets (AIoT). L'ESP32-S3 est disponible sous forme de puce nue ou de module (comme l'ESP32-S3-WROOM-1) qui comprend l'ESP32-S3 ainsi que le quartz nécessaire, la mémoire flash, la commutation haute fréquence et, en option, une antenne sur circuit imprimé ou une connexion pour une antenne externe. Dans cet article, nous aborderons l'implémentation de la puce nue et du module.

Si vous souhaitez utiliser cette puce dans un circuit, il y a quatre éléments importants à prendre en compte :

- la conception du circuit et le dessin du schéma
- l'agencement du circuit imprimé
- le réglage RF et du quartz
- le téléchargement du micrologiciel et le dépannage.

Conception de circuits et dessin de schémas

Pour commencer, il est conseillé d'étudier attentivement la documentation de la puce ou du module à utiliser. Plusieurs documents sont disponibles en ligne [1]. Il s'agit notamment de la fiche technique, des directives de conception matérielle et des conceptions de référence des modules, qui indiquent ce qu'il faut faire pour que la puce fonctionne de manière optimale.

Pour ceux qui utilisent une puce, nous présentons un circuit de base pour la puce ESP32-S3 avec tous les composants nécessaires dans la **figure 1**. Notez que contrairement aux puces sans radio intégrée, un découplage approprié est essentiel ici. Nous spécifions le nombre et les valeurs des condensateurs de découplage pour chaque broche d'alimentation ; il est important de les utiliser et de n'omettre aucun condensateur. En particulier, aux broches 2 et 3, le réseau de filtres CLC (C8, L1 et C9 dans le schéma) doit être utilisé. Sur la plupart des puces Espressif, des broches d'alimentation spécifiques sont affectées à la partie RF, où un CLC ou un filtre CCL doit toujours être utilisé pour supprimer les harmoniques élevées.

Un autre point à noter concernant l'alimentation de l'ESP32-S3 est qu'elle doit être capable de fournir au moins 500 mA. Cela s'applique fondamentalement à toutes les puces Espressif commercialisées jusqu'à présent. (La seule exception est l'ESP32-H2, qui se contente de 350 mA.) Une alimentation qui fournit un courant insuffisant conduira souvent à des réinitialisations de brownout et à d'autres phénomènes gênants, qui se produiront généralement lors de l'initialisation des connexions Wifi.

L'ESP32-S3 nécessite un quartz comme source de signal d'horloge pour l'ensemble du système. Utilisez toujours une bobine en série avec la borne XTAL_P pour supprimer les harmoniques. Les valeurs des condensateurs de découplage C1 et C4 dépendent du quartz, mais aussi de l'impédance parasite des pistes et des pastilles du circuit imprimé. Adafruit propose un bon guide pour déterminer les valeurs initiales de ces composants [2], mais pour obtenir la valeur optimale

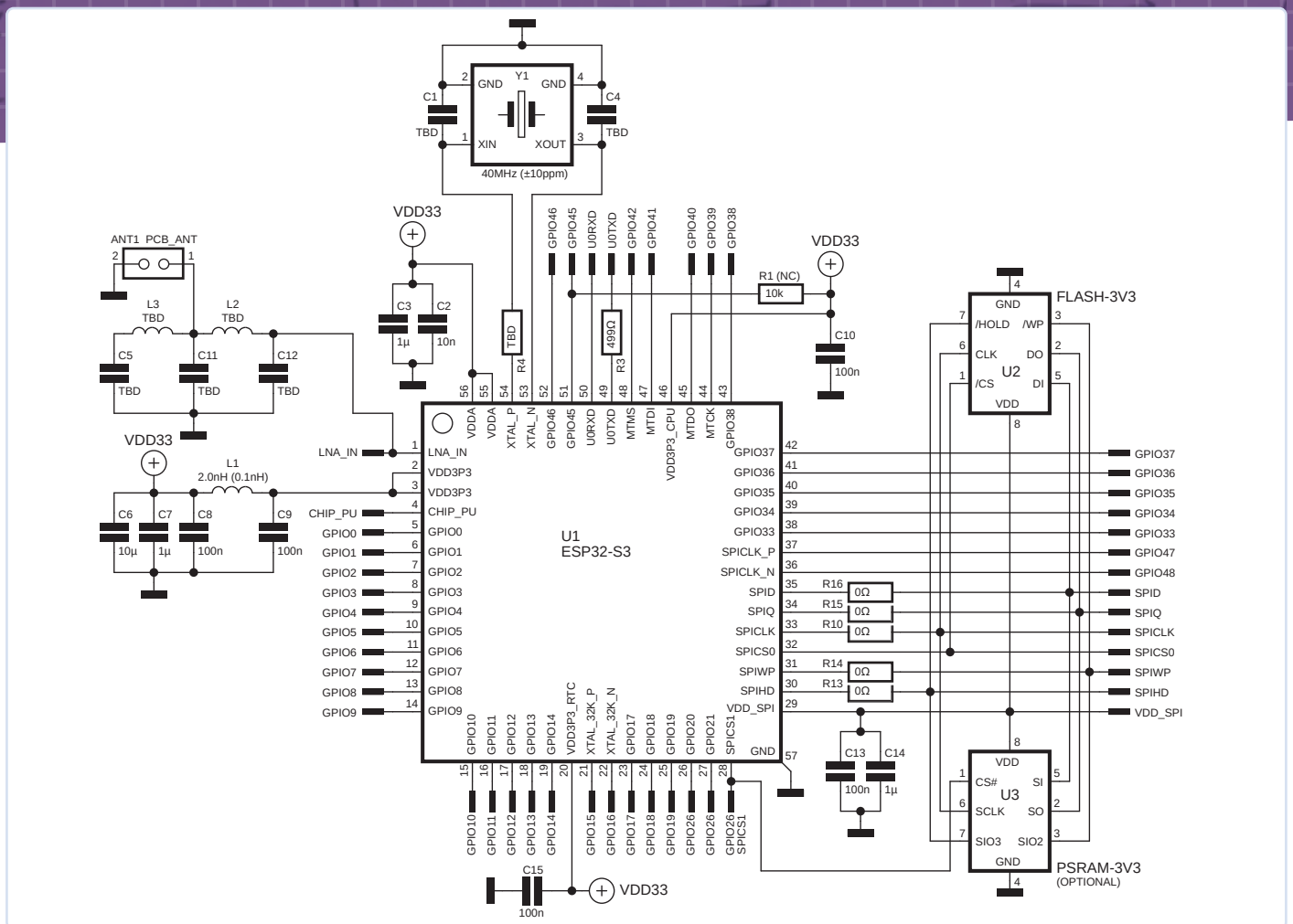


Figure 1. La puce ESP32-S3 et tous les composants qu'elle nécessite généralement. Le découplage est très important ici.

et certifiée, vous devrez les mesurer et les modifier une fois que vous aurez accès au circuit imprimé réel. Plus d'informations à ce sujet dans la suite de cet article.

L'ESP32-S3 nécessite une mémoire flash pour stocker son logiciel ; de la PSRAM peut éventuellement être ajoutée si davantage de mémoire est nécessaire. Certaines variantes de l'ESP32-S3 disposent d'une mémoire flash et/ou PSRAM dans le SoC. Dans ce cas, il est possible d'omettre U2 et/ou U3. Cela signifie qu'il n'est pas nécessaire de réserver de l'espace sur le circuit imprimé pour ces puces, ce qui permet de réduire la taille du circuit imprimé. Dans ce cas, il faut connecter des condensateurs de découplage à VDD_SPI, car ils sont nécessaires pour découpler l'alimentation de la mémoire flash/PSRAM.

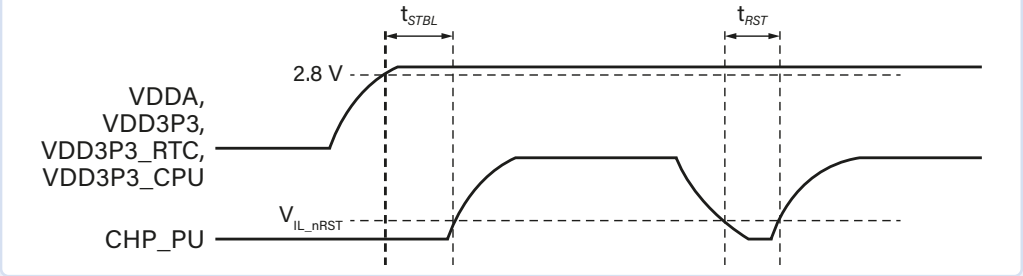
Pour obtenir la meilleure portée radio et vous assurer que le circuit est conforme aux exigences de certification CEM, vous devez vous assurer que les impédances de la piste entre l'ESP32-S3 et l'antenne sont compatibles. En général, la ligne de transmission entre l'ESP32 et l'antenne a une impédance de 50 Ω, mais la pastille ESP32-S3 LNA_IN n'a pas cette impédance et l'antenne que vous utilisez peut ne pas l'avoir non plus. En d'autres termes : la connexion de l'ESP32 à la ligne de transmission nécessite un circuit d'adaptation π-CLC à proximité de l'ESP32-S3 pour adapter son impédance à 50 Ω. La connexion d'une antenne à la piste peut également nécessiter un circuit d'adaptation CLC à proximité, mais vous pouvez l'omettre si vous pouvez confirmer, par exemple par simulation, que l'impédance de l'antenne est de 50 Ω. Notez que dans le schéma, le réseau CLC (composé de C11, L2 et C12) n'a pas de valeurs assignées. En effet, les valeurs nécessaires dépendent du circuit et du matériau du circuit imprimé et doivent être déterminées après la phase de conception. Vous trouverez plus de détails à ce sujet dans la suite de cet article.

Si vous utilisez un module, vous n'avez pas à vous préoccuper des points susmentionnés, car le module comprend déjà les composants et les valeurs corrects.

Si vous utilisez un module ou une puce : La broche CHIP_PU (marquée EN sur le module) sert à la fois de broche d'activation et de réinitialisation ; elle ne doit pas être laissée flottante. Afin de démarrer de manière fiable la puce ESP32-S3, la broche CHIP_PU ne doit être activée que lorsque l'alimentation électrique est stable. (Voir l'encadré **Paramètres de temporisation pour la mise sous tension et la réinitialisation**.) Généralement, un circuit RC est ajouté à cette broche pour générer un délai. Certains produits doivent fonctionner dans un scénario où la rampe d'alimentation est très lente, ou nécessite des mises sous tension et hors tension fréquentes, ou encore lorsque l'alimentation est instable. (Dans ce cas, utiliser uniquement un circuit RC peut être insuffisant pour respecter les délais de la séquence de mise sous tension et hors tension, ce qui peut conduire à un échec de démarrage de la puce. Dans ce cas, nous suggérons d'utiliser d'autres moyens pour répondre aux exigences, tels que l'utilisation d'une puce externe de supervision de l'alimentation ou d'une puce chien de garde. Si votre circuit le permet, vous pouvez également contrôler la broche CHIP_PU par un autre microcontrôleur, ou simplement ajouter un bouton de réinitialisation.

La dernière étape pour faire fonctionner l'ESP32-S3 avec succès est de faire attention aux broches de strapping. À chaque démarrage ou réinitialisation, les puces Espressif ont besoin de certains paramètres de configuration initiaux, tels que le mode de démarrage de la puce, la tension de la mémoire flash, etc. Ces paramètres sont sélectionnés via les broches de strapping. Après la réinitialisation, les broches de strapping fonctionnent comme des broches d'E/S normales. Vous trouverez tous les détails dans la fiche technique de la puce.

Paramètres temporels pour le démarrage et la réinitialisation



Visualisation des paramètres temporels pour le démarrage et la réinitialisation. (Source [5])

Paramètre	Description	Min (µs)
t_{STBL}	Temps réservé à la stabilisation des rails 3,3 V avant que la broche CHIP_PU ne soit tirée vers le haut pour activer la puce.	50
t_{RST}	Temps réservé à CHIP_PU pour rester inférieur à VIL_nRST afin de réinitialiser la puce	50

À ce stade, notre ESP32-S3 peut démarrer avec succès. Cependant, vous voudrez probablement la connecter à d'autres puces, capteurs et actionneurs. Quels GPIOs pouvez-vous utiliser pour cela ? Bonne nouvelle : toutes les puces Espressif disposent d'une fonction appelée GPIO Matrix. Elle facilite grandement la connexion à l'ESP32-S3, car elle permet d'affecter TOUT GPIO disponible à n'importe quel signal de la plupart des périphériques généraux (par exemple, I2C, SPI, SDIO, etc.). Bien sûr, certains périphériques utilisent encore des GPIO fixes (par exemple, ADC). Reportez-vous à la fiche technique de la puce pour plus d'informations si nécessaire. D'une manière générale, n'oubliez pas de lire d'abord les documents techniques – il vaut mieux relire des informations que vous connaissez déjà que de devoir refaire un circuit imprimé à cause d'une erreur évitable.

Disposition du circuit imprimé

Si vous utilisez une puce, vous devriez avoir un schéma parfait pour dessiner la disposition. Pour les produits Wi-Fi et Bluetooth, l'agencement du circuit imprimé est essentiel pour obtenir des performances RF qualifiées. Même si vous n'utilisez qu'une puce Espressif comme microcontrôleur, sans ajouter d'antenne ou utiliser la radio, les directives ci-dessous vous permettront d'obtenir un système stable et une fiabilité accrue. Notez que tous les détails exacts sont disponibles dans les lignes directrices pour la conception du matériel. Nous nous contenterons ici de généraliser les points importants.

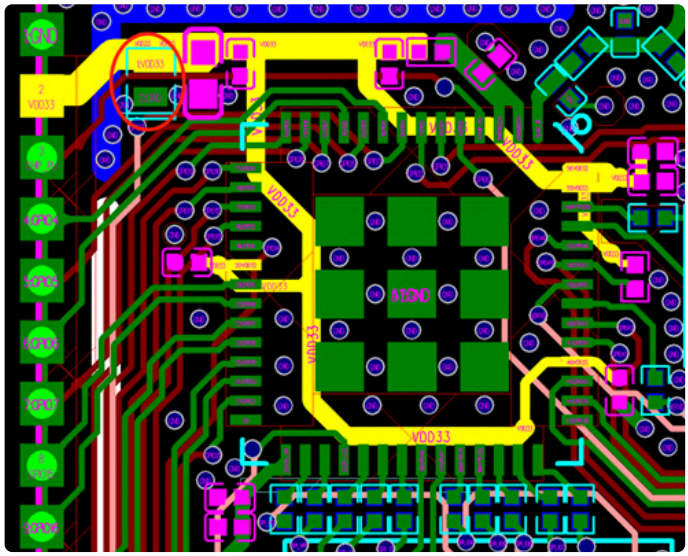


Figure 2. Chaque condensateur de découplage doit être aussi proche que possible des broches d'alimentation.

Commençons par l'empilement des couches. Quatre couches ou plus sont recommandées car, ainsi, nous pouvons avoir un plan de masse large et continu adjacent à la couche de la puce comme couche de masse de référence. Si vous souhaitez vraiment utiliser deux couches, n'oubliez pas de vous assurer que vous disposez toujours d'un bon plan de masse.

Le deuxième point important est le routage des pistes d'alimentation. Nous suggérons de les router en étoile sur une couche intérieure. Chaque condensateur de découplage et circuit de filtrage CLC doit être aussi proche que possible des broches d'alimentation (**figure 2**).

Le troisième point consiste à prendre soin de vos tracés de pistes RF. Vous devez demander à l'usine d'avoir une impédance contrôlée de 50 Ω pour tous les pistes RF. Cela commence au stade de la disposition du circuit imprimé, où vous êtes censé calculer la largeur des pistes et l'écart par rapport à la masse sur la base de votre empilage. L'usine peut ensuite les ajuster légèrement. Si les composants sont dans le format 0201 SMD, veuillez utiliser un stub dans la conception du circuit RF adapté à proximité de la puce.

Vient ensuite le quartz. La bobine en série doit être très proche de la puce ESP32-S3, et les deux condensateurs de charge doivent être placés de deux côtés du quartz (**figure 3**).

Une dernière chose à laquelle il faut prêter attention sont les pistes de la flash et de l'UART, ainsi que les pistes qui vont vers d'autres pastilles GPIO périphériques. Ces pistes peuvent générer des harmoniques à haute fréquence, il est donc conseillé de les acheminer sur les couches internes et de les entourer de masse autant que possible.

Si vous utilisez un module, le point critique est de savoir comment le placer pour obtenir les meilleures performances RF. Les lignes directrices suivantes ne concernent que les modules dotés d'une antenne sur le circuit imprimé ; si vous utilisez un module doté d'une antenne externe, vous disposez d'une plus grande marge de manœuvre quant à l'emplacement de celle-ci sur le circuit imprimé.

Comme indiqué dans la **figure 4**, nous vous suggérons de placer le module dans un coin de la carte. (Notez également que le point d'alimentation ne se trouve pas du même côté pour tous les types de modules ; là encore, vérifiez la fiche technique avant de concevoir le circuit imprimé). Nous recommandons que l'antenne soit complètement à l'extérieur de la carte. Si cela n'est pas possible, assurez-vous qu'il y a un espace libre sous et au moins 15 mm autour de l'antenne :

cela signifie qu'il ne doit pas y avoir de pistes, de plans, de vias, ou de composants à cet emplacement (**figure 5**).
Bien qu'il soit assez facile de souder la plupart des connexions de nos modules Wroom avec un fer à souder standard, il y a une connexion de masse en dessous qui est normalement inaccessible à la main. Si vous avez la possibilité de le faire (par exemple, en utilisant un four à refusion ou une station de soudage à air chaud), nous vous conseillons de la connecter à un plan de masse, car cela permet de dissiper la chaleur et d'obtenir une meilleure mise à la masse pour la radio, mais dans des circonstances normales, le module fonctionnera également sans cette connexion.

Réglage RF et du quartz

Si vous utilisez une puce : si vous avez suivi tous les critères ci-dessus et que vous avez reçu un circuit imprimé fonctionnel de l'usine, pour une meilleure fiabilité et pour passer la certification CEM, vous aurez malheureusement besoin d'équipements professionnels pour régler les valeurs d'adaptation et les condensateurs du quartz. Si vous avez accès à ces équipements (ou si vous disposez de produits peu coûteux comme le MicroVNA), nous allons vous expliquer en quoi consiste cette procédure.
Vous avez probablement choisi les valeurs des condensateurs de charge en fonction de la capacité parasite estimée des pistes. Il est maintenant temps de les ajuster : parce que le 2,4 GHz principal est généré par ce quartz, l'optimisation de son fonctionnement vous permettra d'obtenir une meilleure portée et des harmoniques plus faibles. Nous allons optimiser cela en utilisant l'outil ESP RF Test que vous pouvez télécharger sur notre site [3].

1. Sélectionnez le mode de *TX tone* avec l'outil *Certification and Test*.
2. Observez le signal de 2,4 GHz avec un analyseur de communication radio ou un analyseur de spectre et démodulez-le pour obtenir le décalage de fréquence réel.
3. Réglez le décalage de fréquence à ±10 ppm (recommandé) en ajustant la valeur de la capacité de charge externe.

- Lorsque le décalage de la fréquence centrale est positif, cela signifie que la capacité de charge équivalente est trop faible et que la capacité de charge externe doit être augmentée.
- Lorsque le décalage de la fréquence centrale est négatif, cela signifie que la capacité de charge équivalente est trop grande et que la capacité de charge externe doit être réduite.
- Les condensateurs de charge externe des deux côtés sont généralement égaux, mais dans des cas particuliers, ils peuvent avoir des valeurs légèrement différentes.

Une fois les valeurs des condensateurs de charge du quartz réglées , nous passons au réseau d'adaptation de l'antenne. Dans le circuit d'adaptation, nous définissons le port proche de la puce comme Port 1 et le port proche de l'antenne comme Port 2. S11 décrit le rapport entre la puissance du signal réfléchi par le Port 1 et la puissance du signal d'entrée, et S21 est utilisé pour décrire la perte de transmission du signal du Port 1 au Port 2. Pour la série de puces ESP32-S3, si S11 est inférieur ou égal à -10 dB et S21 est inférieur ou égal à -35 dB lors de la transmission des signaux 4,8 GHz et 7,2 GHz, le circuit d'adaptation peut satisfaire aux exigences de transmission.
Connectez les deux extrémités du circuit d'adaptation à l'analyseur de réseau (voir l'encadré « Circuit d'adaptation ») et testez le paramètre de

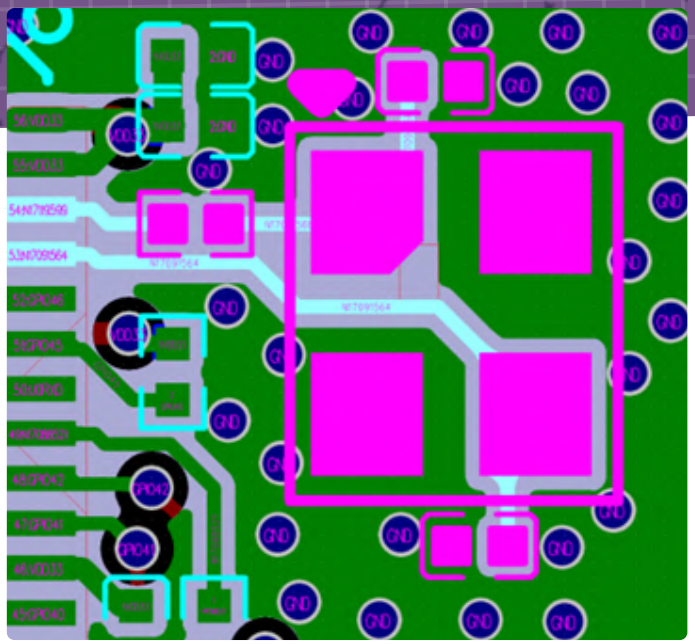


Figure 3. Les deux condensateurs de charge doivent être placés de part et d'autre du quartz.

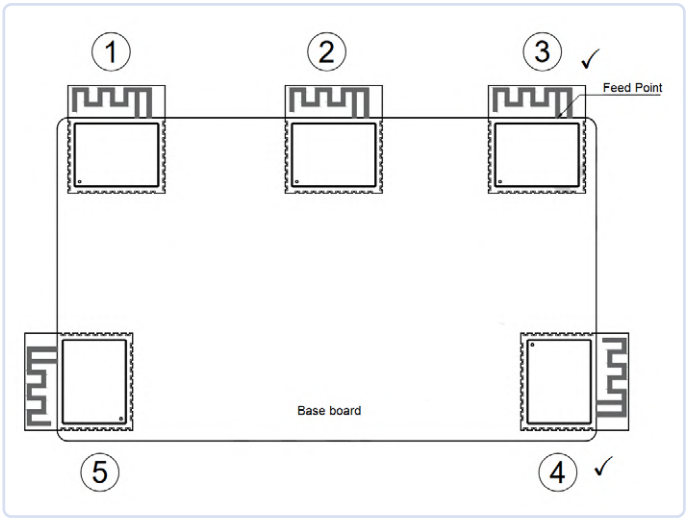


Figure 4. Montage d'un module sur une carte de base. (Source : [6])

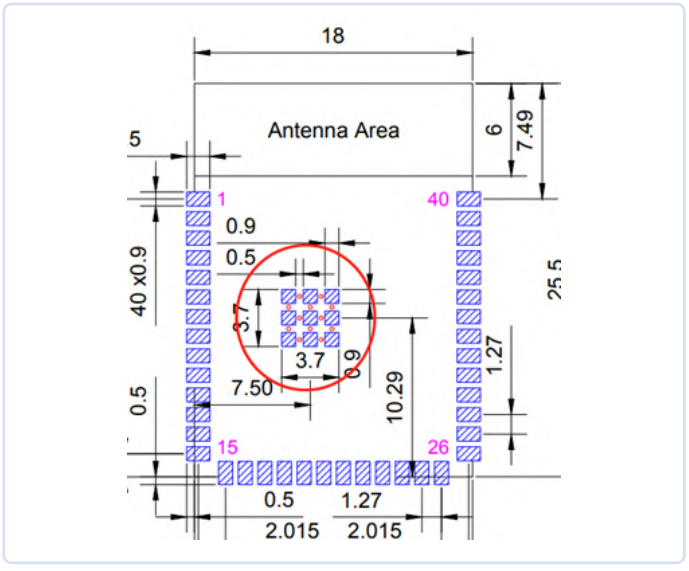
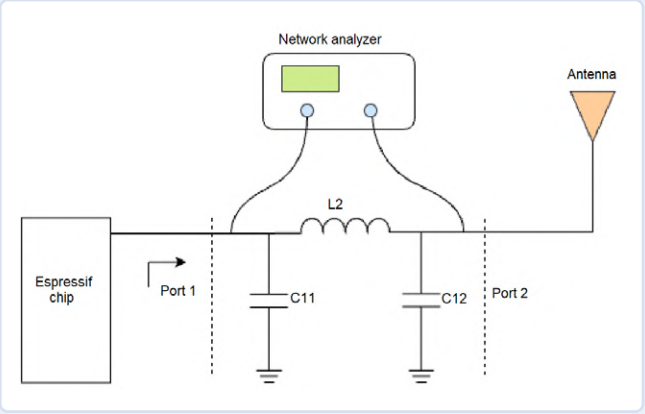


Figure 5. Circuit imprimé recommandé pour l'ESP32-S3-WROOM-1. (Source : [6])

Matching Circuit



Source [7]

Indicateur de référence	Valeur recommandée	Numéro de série
C11	1.2 ~ 1.8 pF	GRM0335C1H1RXBA01D
L2	2.4 ~ 3.0 nH	LQP03TN2NXB02D
C12	1.8 ~ 1.2 pF	GRM0335C1H1RXBA01D

réflexion S11 et le paramètre de transmission S21 du signal. Ajustez les valeurs des composants du circuit jusqu'à ce que S11 et S21 répondent aux exigences. Si la conception du circuit imprimé de la puce respecte strictement les directives de conception du circuit imprimé et que l'antenne est bien conçue, vous pouvez vous référer aux plages de valeurs dans l'encadré pour déboguer le circuit d'adaptation.

Alors, que faire lorsque vous voulez intégrer une puce ESP32-S3 dans votre carte, mais que vous ne disposez pas des équipements sophistiqués pour le faire correctement ? Si vous avez suivi tous les autres points de cet article, vous pourrez peut-être utiliser les valeurs estimées des condensateurs de charge et ne pas utiliser de réseau CLC. (Si vous en avez quand même conçu un dans votre circuit imprimé, vous pouvez supprimer C11/C12 et remplacer L2 par une résistance de zéro ohm. Ainsi, vous pourrez ajouter un réseau CLC plus tard sans avoir à modifier la disposition du circuit imprimé). D'après notre expérience, l'absence d'un réseau d'adaptation d'impédance n'affecte pas trop la portée ; sa principale utilité est de réduire l'EMV (Error Magnitude Vector) des appareils, ce qui est nécessaire pour satisfaire aux exigences de certification.

Si vous utilisez un module : aucune des opérations susmentionnées n'est nécessaire, car votre module est livré avec tous ses composants pré-réglés pour des performances optimales.

Téléchargement du micrologiciel et résolutions de problèmes

Si vous utilisez un module ou une puce : avant que votre ESP32-S3 ne réalise quoi que ce soit, vous devrez charger un micrologiciel dans sa mémoire flash (ce qu'on appelle la programmation ou le téléchargement d'un micrologiciel). Pour toutes les puces Espressif, le processus de téléchargement est le suivant : réinitialiser la puce en mode téléchargement – télécharger – réinitialiser et faire fonctionner la puce en mode SPI Boot.

Vous pouvez programmer les puces Espressif généralement de deux façons : toutes les puces Espressif supporteront le téléchargement de micrologiciel via l'UART, mais les puces les plus récentes ont généralement aussi un périphérique USB qui vous permet de télécharger le micrologiciel directement via une connexion USB à votre ordinateur. Vous trouverez ci-dessous les étapes détaillées pour l'ESP32-S3.

Pour télécharger via UART :

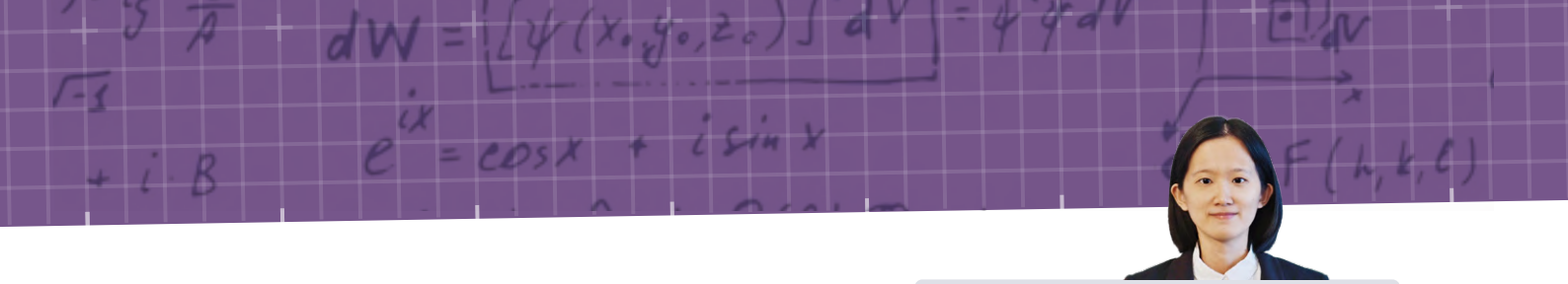
1. Avant le téléchargement, assurez-vous de mettre la puce ou le module en mode *Download Boot*, c'est-à-dire que la broche de strapping GPIO0 (tirée vers le haut par défaut) est tirée vers le bas et que la broche GPIO46 (tirée vers le bas par défaut) est laissée flottante ou tirée vers le bas. Configurez la broche GPIO45 de manière appropriée selon le tableau des broches de strapping, ou laissez-la flottante si vous utilisez un module.
2. Mettez la puce ou le module sous tension et vérifiez s'il est entré en mode UART Download via le port série UART0. Si le journal indique « *waiting for download* «, la puce ou le module est entré en mode *Download Boot*.
3. Téléchargez votre micrologiciel dans la mémoire flash via UART. Vous pouvez utiliser n'importe quelle option de programmation que votre environnement de programmation (Arduino, ESP-IDF, VSCode...) vous donne ; ou vous pouvez utiliser l'outil autonome *Flash Download Tool*.
4. Une fois le micrologiciel téléchargé, tirez IO0 vers le haut ou laissez-la flottante pour vous assurer que la puce ou le module entre en mode *SPI Boot*.
5. Alimentez à nouveau le module. La puce lira et exécutera le nouveau micrologiciel pendant l'initialisation.

Notez que la plupart des cartes de développement ont un schéma qui permet au logiciel de faire passer automatiquement la puce ESP32 en mode téléchargement et de la faire sortir de ce mode. Reportez-vous, par exemple, aux schémas de l'ESP32-S3-Devkit-C-1 si vous voulez savoir comment implémenter cela [4].

Pour télécharger via USB :

1. Dans la plupart des cas, lorsque le micrologiciel de la puce fonctionne, l'outil de flashage devrait être en mesure de mettre la puce en mode *USB download* via la connexion USB. Dans ce cas, vous pouvez passer à l'étape suivante. Si ce n'est pas le cas, vous devez mettre la puce ou le module en mode *Download Boot*, c'est-à-dire vous assurer que la broche de strapping GPIO0 (tirée vers le haut par défaut) est tirée vers le bas et que la broche GPIO46 (tirée vers le bas par défaut) est laissée flottante ou tirée vers le bas. Configurez la broche GPIO45 comme indiqué dans le tableau des broches de connexion, ou laissez-la flottante si vous utilisez un module.
2. Allumez la puce ou le module et vérifiez s'il est entré en mode *UART Download* via le port série USB. Si le journal indique «*waiting for download*», la puce ou le module est entré en *mode Download Boot*.
3. Téléchargez votre firmware dans la mémoire flash via UART. Vous pouvez utiliser n'importe quelle option de programmation que votre environnement de programmation (Arduino, ESP-IDF, VSCode...) vous donne ou utilisez l'outil *Flash Download*.
4. Une fois que le micrologiciel est téléchargé et que la puce est entré automatiquement dans le mode *USB Download*, la procédure est terminée. Si ce n'est pas le cas, tirez IO0 vers le haut ou laissez-la flottante pour vous assurer que la puce ou le module entre en mode SPI Boot. Réinitialisez ou mettez sous tension la puce ou le module, et il lira et exécutera le nouveau micrologiciel pendant l'initialisation.

Si vous ne parvenez pas à télécharger le micrologiciel via l'UART, vérifiez le journal de l'UART0 avec un outil de terminal série. Le message de démarrage de l'ESP32-S3 vous indiquera la valeur réelle



À propos de l'auteur

Liu Jinghui, diplômée en ingénierie électronique, a rejoint Espressif et s'est consacrée à la résolution des problèmes de matériel. Pendant cinq ans, elle a été la représentante des produits Espressif et de leurs utilisations. Elle a une bonne compréhension des caractéristiques matérielles des produits et des problèmes qui peuvent survenir lors de leur utilisation par les clients.

de verrouillage des broches de strapping, ce qui vous permettra de déterminer quelle valeur est erronée. La valeur après `boot:` montre les valeurs des broches de strapping en hexadécimal : bit 2 = GPIO46, bit 3 = GPIO0, bit 4 = GPIO45, bit 5 = GPIO3 :

```
ets Jun 8 2016 00:22:57
rst:0x1 (POWERON_RESET),boot:0x3 (DOWNLOAD_BOOT(UART0/
UART1/SDIO_REI_REO_V2))
```

Si vous ne reconnaissez pas le périphérique USB ou si la connexion USB est instable, essayez d'abord de passer en mode téléchargement, puis de télécharger. Rappelez-vous également que du côté de l'ordinateur, un seul programme doit ouvrir le port série à la fois : essayer de flasher tout en ayant un terminal série ouvert sur le même port est voué à l'échec.

Notez que si le passage en mode téléchargement par USB est généralement fiable, il existe quelques scénarios dans lesquels il peut échouer :

- USB PHY est désactivé par l'application ;
- Le port USB est reconfiguré pour d'autres tâches, par exemple, hôte USB, appareil standard USB ;
- Les GPIO USB sont reconfigurés, par exemple comme sorties pour LEDC, SPI, ou comme GPIO génériques.

Dans ce cas, il est nécessaire d'entrer manuellement en mode téléchargement. C'est pourquoi, au moins pendant le développement du micrologiciel, nous suggérons d'avoir toujours un moyen (boutons, cavaliers, ...) de configurer les broches de strapping pour forcer un démarrage en mode téléchargement.

Une autre chose à noter : parfois, certains constatent que leur carte ne démarre pas correctement (c'est-à-dire qu'elle ne commence pas à exécuter leur programme en flash) à moins qu'ils ne la réinitialisent manuellement. Cela peut se produire lorsqu'un condensateur est placé sur une broche de strapping (par exemple, GPIO0 sur l'ESP32-S3). Un tel condensateur est parfois placé pour amortir un bouton, mais lorsque l'appareil est mis sous tension, le niveau du signal du GPIO0 augmente lentement, ainsi, l'ESP32-S3 le considère comme une valeur basse.

Les modules font la magie

Comme vous pouvez le voir dans cet article, bien qu'il soit possible de réaliser un projet fonctionnel, concevoir une carte intégrant un ESP32-S3 avec des performances optimisées et une certification réussie n'est pas trivial et nécessite des connaissances approfondies ainsi que des outils. Ainsi, si vous disposez de l'espace nécessaire, nous vous suggérons d'utiliser plutôt un module ; toute la magie RF est déjà faite par Espressif, ce qui rend l'intégration d'un module dans votre projet beaucoup plus facile.

Si tout cela vous semble encore trop compliqué, comme indiqué au début de l'article, Espressif propose également une large gamme de cartes de développement, des plus simples qui déploient toutes les GPIO vers des connecteurs faciles à utiliser comme l'ESP32-S3-DevkitC-1, aux plus puissantes cartes avec la reconnaissance vocale intégrée comme l'ESP32-S3-Box-3 et l'ESP32-S3-Eye, dotée d'un appareil photo. En général, les schémas et les designs de ces cartes de développement sont disponibles, donc même si vous ne prévoyez pas de les utiliser dans votre produit, ils peuvent constituer un excellent point de départ pour votre projet.

230563-04

Questions ou commentaires ?

Envoyez un courriel à l'auteure (liujinghui@espressif.com) ou contactez Elektor redaction@elektor.fr).

Produits

- **ESP32-S3-WROOM-1**
  www.elektor.fr/20696

LIENS

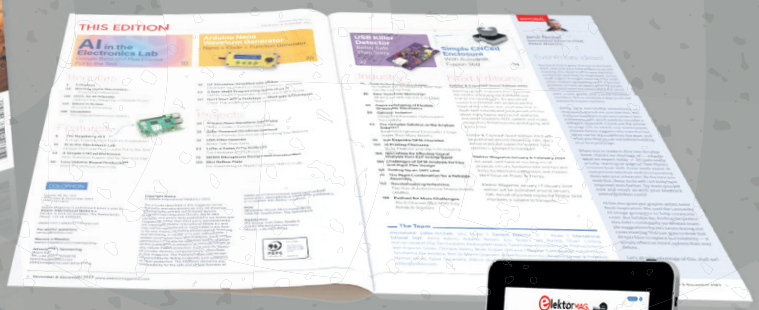
[1] Documents techniques d'Espressif :
https://espressif.com/en/support/documents/technical-documents?keys=&field_type_tid%5B%5D=842
[2] Adafruit, "Choosing the Right Crystal and Caps for your Design," 2012 :
https://blog.adafruit.com/2012/01/24/choosing-the-right-crystal-and-caps-for-your-design
[3] ESP RF Test Tool : https://espressif.com/en/support/download/other-tools
[4] Schéma de l'ESP32-S3-DevKitC-1 : https://dl.espressif.com/dl/schematics/SCH_ESP32-S3-DevKitC-1_V1.1_20221130.pdf
[5] Fiche technique de la série ESP32 : https://espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf
[6] ESP32-H2-Series Hardware Design Guidelines :
https://espressif.com/sites/default/files/documentation/esp32-h2_hardware_design_guidelines_en.pdf
[7] Fiche technique de l'ESP32-S3-WROOM-1 :
https://espressif.com/sites/default/files/documentation/esp32-s3-wroom-1_wroom-1u_datasheet_en.pdf

20% de réduction
sur la première année de votre abonnement

Rejoignez la communauté Elektor

Devenez membre **maintenant !**

elektor GOLD membership



- accès à l'archive numérique depuis 1978 !
- 8x magazine imprimé Elektor
- 8x magazine numérique (PDF)
- 10 % de remise dans l'e-choppe et des offres exclusives pourles membres
- accès à plus de 5000 fichiers Gerber
- Livraison gratuite en France

www.elektormagazine.fr/gold-member

Utilisez le code promo :

ESPRESSIF20

ESPRESSIF × elektor

Q&A avec Teo Swee-Ann

(PDG d'Espressif)

innovation des puces AIoT

entretien avec Teo Swee-Ann, PDG d'Espressif

Comment s'y prendre pour mettre sur le marché une solution électronique innovante à une vitesse révolutionnaire ? Teo Swee-Ann, directeur général d'Espressif, nous fait part de ses réflexions. Nous découvrons son parcours professionnel et comment il en est venu à concevoir et à déployer des puces, des modules et des solutions AIoT.

Elektor : quand avez-vous commencé à vous intéresser à l'électronique ? Avez-vous été inspiré par un parent ou un professeur ? Ou peut-être êtes-vous devenu curieux en faisant de la rétro-ingénierie et en bricolant des produits électroniques ?

Teo Swee-Ann : j'ai commencé à m'intéresser à l'électronique dès mon plus jeune âge. En grandissant, j'ai toujours été fascinée par les gadgets et les machines. Nous faisions des choses habituelles, comme démonter les radios et les téléviseurs, pour essayer de comprendre comment ils fonctionnaient. À l'époque, il ne s'agissait pas vraiment de rétro-ingénierie, mais plutôt d'une simple curiosité pour voir les coulisses. Comme d'habitude, le fait de démonter les choses n'a répondu à aucune question, mais a renforcé ma curiosité.

Aucun membre de ma famille proche n'était très impliqué dans l'électronique, et mon inspiration n'est donc pas venue directement d'un parent. Mais il semble que j'aie des facilités pour les logiciels et les mathématiques. Mon mémoire portait sur l'électromagnétisme informatique. J'ai inventé quelques techniques utiles pour calculer très efficacement les fonctions de Green pour les milieux stratifiés. Il s'est avéré que cette technique, ainsi qu'une autre appelée « circuit équivalent à éléments partiels », pouvait être utile pour la caractérisation des inducteurs dans les puces, en particulier pour tenir compte des pertes de substrat. Je pense que c'est ainsi que j'ai trouvé un emploi dans l'industrie des semiconducteurs. Lorsque j'ai eu la chance de travailler sur la conception de circuits, je me suis immédiatement procuré quelques livres classiques de conception de circuits intégrés de Paul Gray, Philip Allen, Ken Martin, et je les ai pratiquement appris par cœur. Il n'y a pas eu de regard en arrière par la suite.

Elektor : l'ingénierie et l'esprit d'entreprise ont-ils toujours fait partie de vos projets ? Lorsque vous avez commencé à fréquenter l'université, avez-vous songé à lancer et à gérer une entreprise ?

Teo Swee-Ann : je n'ai pas pensé à l'aspect commercial des choses. Je suis plus idéaliste ; je crois que c'est la connaissance qui transformera le monde, pas les affaires. Mais, bien sûr, mon expérience acquise m'a appris qu'il faut parfois en faire plus, s'adapter et créer une plateforme commerciale pour la technologie.

Elektor : parlez-nous de vos intérêts techniques. Sur quoi avez-vous axé votre maîtrise à l'Université nationale de Singapour ?

Teo Swee-Ann : à l'université, je préférais les mathématiques et les logiciels et je fuyais tout ce qui concernait le matériel. Mon mémoire de maîtrise portait donc sur la caractérisation des éléments passifs hyperfréquences dans les couches de matériaux à l'aide de l'électromagnétisme informatique. Un support de couche peut être un objet tel qu'un circuit imprimé ou un circuit intégré, qui est fabriqué couche par couche et dont le matériau varie d'une couche à l'autre. L'objectif était de calculer les caractéristiques à haute fréquence des composants passifs à micro-ondes, tels que les inductances, dans de tels milieux.

Nous étions alors confrontés à de nombreux problèmes : la difficulté de créer une seule fonction de Green pour représenter à la fois le champ proche et le champ lointain, la manière d'extraire les pôles des fonctions et la manière d'effectuer les transformations de Sommerfeld des fonctions du domaine des fréquences au domaine spatial. Mon travail a donc été divisé en deux parties.

La première partie consistait à montrer comment nous pouvions créer une fonction unique pour représenter à la fois le champ proche et le champ lointain et disposer d'une méthode de calcul efficace pour calculer leurs transformées de Sommerfeld (quelque chose comme la transformée de Laplace, mais avec un noyau de Bessel ou de Hankel). Nous savons que ces fonctions présentent des singularités et des coupures de branches. C'est donc comme si nous marchions sur un champ de mines mathématique avec quelques barrières.

Il s'agissait d'abord d'extraire les singularités de ces fonctions et ensuite de créer des « pôles synthétiques » qui ont des solutions de forme fermée dans le calcul suivant, afin d'accélérer la convergence du calcul (un problème qui s'est posé en raison de l'inclusion des pôles pour l'extraction des singularités). La petite partie restante des fonctions qui n'ont pas été prises en compte sera alors calculable et aura une convergence rapide.

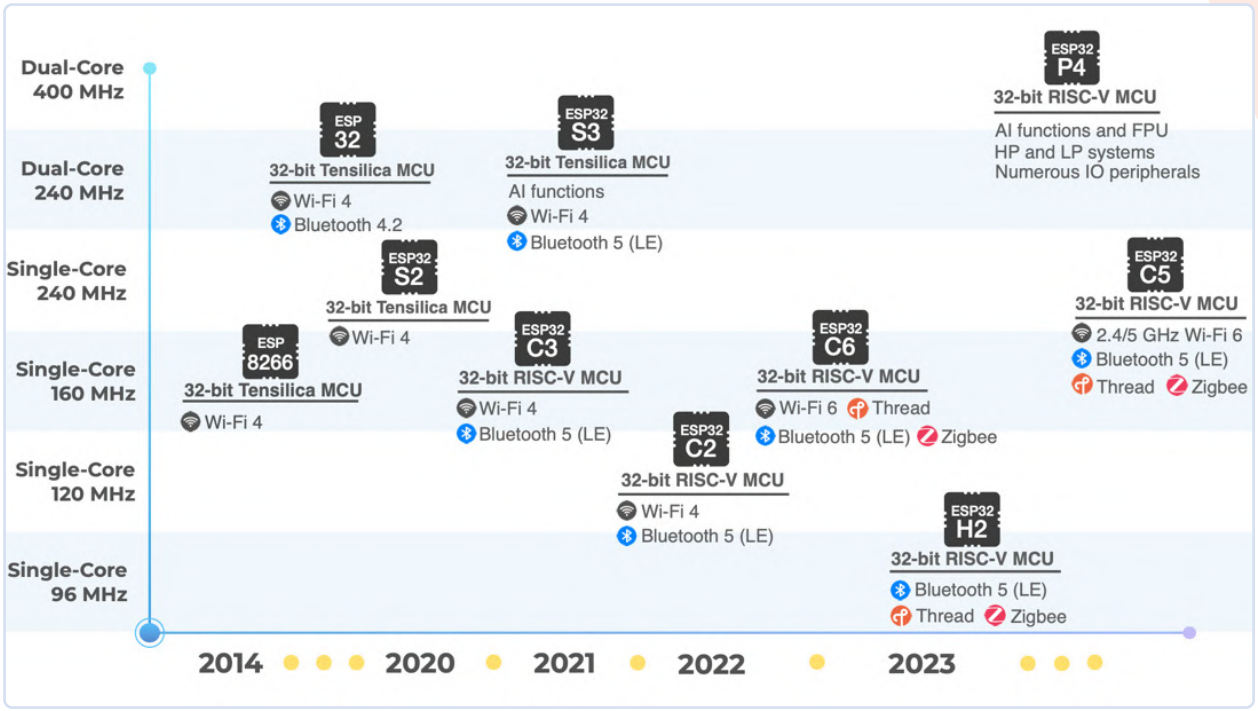
Enfin, le problème restant était de savoir comment extraire les singularités dans le domaine complexe. J'ai créé une technique alternative, que je nomme « Tomographie de Cauchy » pour extraire ces singularités, en utilisant le théorème du résidu de Cauchy et en récupérant leurs emplacements via la technique du crayon de fonction généralisée (GPOF). Par un heureux hasard, il m'arrive encore aujourd'hui d'utiliser cette technique dans la conception de circuits pour calculer la fonction de transfert d'un système sur la base de sa réponse transitoire. Il s'avère que la réponse transitoire est une sorte d'intégration dans le domaine temporel de la réponse en fréquence du système. En convertissant les deux domaines, on peut en quelque sorte voir comment les pôles du système créent un certain type de réponse transitoire et vice versa.

Elektor : votre réponse à l'invitation d'Elektor à devenir le prochain rédacteur en chef invité a été un « oui » ferme et spontané. Quand avez-vous fait la connaissance d'Elektor ? Elektor vous a-t-il influencé ou affecté, vous et Espressif, d'une manière ou d'une autre ?

Teo Swee-Ann : c'est en 2015 que nous avons découvert Elektor. Le magazine se distinguait par sa capacité à identifier et à mettre en lumière les tendances émergentes dans l'industrie. En fait, je crois qu'Elektor a été parmi les pionniers à présenter les puces d'Espressif, en commençant notamment par l'ESP8266, jouant ainsi un rôle important dans leur popularisation. Nous nous tournons régulièrement vers Elektor comme source d'inspiration, toujours désireux de mesurer l'évolution des besoins et des intérêts de son lectorat. En outre, Elektor joue un rôle inestimable en servant de source de connaissances pour ceux qui aspirent à entrer dans le secteur de l'électronique. Il souligne une obligation commune que nous ressentons dans l'industrie : diffuser nos connaissances et, ce faisant, veiller à ce que le domaine de l'électronique continue d'attirer et de cultiver de nouveaux talents.

Elektor : quels sont les avantages et les inconvénients de votre formation d'ingénieur dans votre rôle actuel de PDG d'Espressif ?

Teo Swee-Ann : je n'y vois aucun inconvénient. J'aime être en première ligne et je continue à travailler sur les circuits. Le fait de travailler aux côtés de l'équipe me permet de mieux comprendre les problèmes auxquels nous sommes confrontés et les difficultés liées à la mise en place de chaque fonction. Cela me permet de mieux répartir les ressources, d'identifier les problèmes clés, de communiquer avec l'équipe et de parvenir à un consensus.



L'une de nos décisions initiales était qu'une puce IdO devait idéalement avoir 600 DMIPS. Ce niveau de performance offre un équilibre entre la puissance, surpassant de nombreux microcontrôleurs, et la rentabilité, grâce à son architecture double cœur.

Elektor : quels sont vos centres d'intérêt en dehors de l'ingénierie et de l'électronique ?
Teo Swee-Ann : je joue beaucoup de musique lorsque je réfléchis à des problèmes. Cela débloque en quelque sorte le cerveau. J'avais l'habitude de jouer de la guitare classique et j'ai récemment repris le violoncelle. J'étudie actuellement les suites pour violoncelle de Bach, petit à petit, et je m'efforce de m'améliorer.

Elektor : vous avez lancé Espressif en 2008. Votre premier produit a été l'ESP8086, sorti en 2013. Etait-il l'objectif principal de l'entreprise pendant ces cinq années, ou fournissiez-vous également d'autres services ?
Teo Swee-Ann : au cours des premières années, il s'agissait d'une très petite structure. Je travaillais sur un logiciel afin de créer un langage pour décrire les circuits. D'où le nom « Espressif », qui signifiait exprimer nos idées sur les circuits. Cependant, il est difficile de vendre des logiciels, et nous avons bifurqué pour devenir une société de conseil, construisant des IP analogiques pour nos clients. Finalement, nous avons été inspirés par le concept d'une singularité technologique imminente et avons décidé de construire des puces IdO, qui agiraient comme le système nerveux d'un tel appareil intelligent.

Elektor : en 2014, Espressif a lancé son premier SoC IdO, l'ESP8266EX. Quels types de défis techniques visiez-vous avec cette solution ?
Teo Swee-Ann : à cette époque, l'idée de créer une puce Wifi bon marché circulait. Nous avons décidé de relever ce défi en intégrant tous les composants externes RF dans la puce. Jusqu'à présent, les modules étaient encombrés de 50 à 100 composants. Nous avons révolutionné cette conception en intégrant le balun, le commutateur d'antenne, l'amplificateur de puissance et l'amplificateur faible bruit (LNA) directement dans la puce. Le principal défi consistait à gérer la puissance de crête de 27 dBm de l'amplificateur de puissance. Sans un commutateur suffisamment robuste, le LNA risquait d'être endommagé.
Nous avons également modifié la façon dont les amplificateurs de puissance étaient pilotés en interne. Au lieu d'inductances, nous avons utilisé des baluns, ce qui a amélioré la stabilité du système. Par ailleurs, dans le passé, la plupart des puces Wifi chargeaient les images logicielles dans la mémoire de manière plutôt statique. Nous avons adopté une conception de cache afin que le logiciel puisse être lu en continu à partir de la mémoire flash. C'est beaucoup plus lent, mais cela fonctionne. Nous avons également intégré une grande partie de la logique du système dans le logiciel, plutôt que dans le matériel. Cela a facilité le développement.
La convergence de ces innovations progressives a conduit à la création d'une conception très compacte. Notre produit final coûtait entre un quart et un tiers de ce que proposaient nos concurrents. Aujourd'hui, nous sommes fiers de reconnaître que bon nombre des techniques que nous avons mises au point sont devenues des pratiques courantes dans l'industrie, contribuant de manière significative à la réduction des coûts associés aux puces IdO.

Elektor : 2016 a été une année importante pour Espressif. Parlez-nous du développement de l'ESP32 et de sa sortie.
Teo Swee-Ann : avant l'ESP32, notre approche était quelque peu spontanée et moins structurée. L'ESP32 a marqué un changement important dans notre état d'esprit, nous amenant à aborder le marketing de manière plus systématique. L'une de nos décisions initiales était qu'une puce IdO devait idéalement avoir 600 DMIPS. Ce niveau de performance offre un équilibre entre la puissance, surpassant de nombreux microcontrôleurs, et la rentabilité, grâce à son architecture double cœur et à son système de cache efficace. Le processus pour façonner l'ESP32 a nécessité de nombreuses discussions, délibérations et révisions.
C'est à ce moment-là que nous avons formellement établi la structure de notre équipe, affiné notre processus de travail et aligné nos processus de travail sur la conception et les spécifications du produit. Nous nous sommes ainsi éloignés de notre approche précédente, plus « garage » et moins formelle. Parallèlement au développement de l'ESP32, notre équipe logicielle a lancé l'ESP-IDF. Les décisions, telles que celles concernant nos politiques en matière de logiciels libres, ont été consolidées au cours de cette période, établissant une base solide pour les progrès futurs. En résumé, l'évolution de l'ESP32 a mis en évidence l'importance croissante des logiciels, plutôt que de se concentrer uniquement sur le matériel. Cette transition a été enrichie par l'engagement actif et le retour d'information inestimable de notre communauté de développeurs dévoués.

Elektor : parlez-nous des solutions d'Espressif compatibles avec Matter, qui semble être une priorité en 2023 et en 2024, n'est-ce pas ?
Teo Swee-Ann : jusqu'à présent, le secteur de la maison intelligente n'a pas atteint son potentiel et il est encore difficile pour les consommateurs d'utiliser les produits de manière efficace. Les cas d'utilisation sont limités et l'expérience offerte est fragmentée. Matter tente d'apporter un changement en faisant en sorte que les appareils parlent le même langage, et il est très encourageant de voir la plupart des grands acteurs s'unir pour résoudre ces problèmes. Plus important encore, nous constatons des réponses positives de la part des clients. Il s'agit donc d'une priorité naturelle pour nous. L'approche d'Espressif pour construire des solutions Matter ne se limite pas au matériel. Bien que nous ayons différentes puces supportant les protocoles Matter over Thread et Wifi, nous allons

plus loin en comprenant les problèmes spécifiques des clients dans son adoption et en essayant de créer une solution pour ceux-ci. Nos modules ESP ZeroCode, notre service de fourniture de certificats et notre service d'assistance à la certification sont de bons exemples de cette approche.

Elektor : comment s'est déroulée la création de l'ESP RainMaker ?
Teo Swee-Ann : ESP RainMaker a également été conçu pour répondre aux problèmes rencontrés par les clients. Auparavant, nous les voyions réinventer la roue lorsqu'il s'agissait de construire une plateforme cloud IdO ou de fabriquer des produits basés sur des plateformes cloud tierces avec une différenciation et un contrôle minimal sur les données. D'autre part, l'architecture cloud sans serveur était tellement prometteuse pour construire un tel cloud sans se soucier de la gestion de l'infrastructure. Il est cependant encore difficile pour les fabricants d'appareils de construire une telle plateforme à partir de zéro. C'est pourquoi nous avons créé ESP RainMaker avec la vision de construire une plateforme cloud pour les clients qui leur donne un contrôle total et une personnalisation pour construire leur propre cloud IdO avec beaucoup moins d'investissement.

Elektor : les systèmes et produits AIoT transmettent beaucoup de données. Cela suscite bien sûr des inquiétudes en matière de protection des données et de la vie privée. Espressif pense-t-elle que le marché des produits AIoT connaîtra un « boom » une fois que des normes internationales en matière de protection des données et de la vie privée auront été adoptées ?
Teo Swee-Ann : dans le paysage actuel de l'AIoT, les questions de confidentialité des données et de réglementation sont certainement d'une grande signification, et nous fournissons à nos clients de multiples outils pour les éviter. Tout d'abord, nous encourageons l'IA en périphérie en intégrant sa prise en charge dans notre matériel : par exemple, l'ESP32-S3 et le prochain ESP32-P4 ont des instructions IA pour le traitement rapide des modèles d'apprentissage profond, de sorte que les appareils peuvent traiter les données localement plutôt que d'avoir à les envoyer à des serveurs. Pour les données qui sont stockées sur des serveurs, nous fournissons aux clients des outils qui le font d'une manière conforme à la réglementation. Par exemple, comme l'a examiné le TÜV Rheinland, l'implémentation cloud ESP RainMaker d'Espressif fournit toutes les fonctions nécessaires pour atteindre la conformité GDPR lorsque les clients construisent leur propre cloud IdO basé sur ESP RainMaker. La sécurité joue également un rôle : De nos jours, il n'est pas rare que des acteurs malveillants s'introduisent dans un système et s'emparent de toutes les données. Nous nous sommes engagés à fournir des composants matériels et logiciels sécurisés conformes au programme américain *Cybersecurity Labeling* et à d'autres initiatives internationales, telles que le *Cybersecurity Labelling Scheme* de Singapour. Les dispositifs certifiés dans le cadre de tels programmes peuvent apporter aux clients une certaine tranquillité d'esprit en ce qui concerne leur sécurité.
Un défi plus pressant que nous avons identifié est le manque de développeurs de logiciels embarqués. En outre, la fragmentation du marché complique encore la situation, le complexifiant encore. Chez Espressif, nous relevons activement ces défis. Nous avons introduit de nombreuses solutions. Par exemple, nous avons ESP-ZeroCode pour développer des applications Matter, ESP-SkaiNet pour la reconnaissance vocale locale de l'IA, ESP-RainMaker pour créer votre propre plateforme cloud, ESP-ADF pour créer votre propre système audio Wifi, etc.
Notre objectif est de doter nos clients des outils dont ils ont besoin pour créer des solutions sans partir de zéro, afin de garantir l'efficacité et la rapidité du développement. Pour une évolution positive significative du paysage de l'AIoT, nous pensons qu'il est essentiel de réduire la fragmentation. Il est impératif que les différentes plateformes collaborent et favorisent une norme unifiée. Si Matter est prometteuse dans cette direction, son plein potentiel et son impact à long terme restent à vérifier.

Elektor : la pénurie mondiale de puces a-t-elle favorisé l'utilisation de la technologie Espressif dans des produits commerciaux ? Si oui, comment ces produits peuvent-ils contribuer à asseoir la marque Espressif ?
Teo Swee-Ann : pendant cette période difficile de pénurie mondiale de puces, Espressif a mis en œuvre une approche stratégique en rationnant les ventes de puces à nos clients estimés. Cette décision a été appréciée par nombre de nos clients. Tout d'abord, elle leur a permis de continuer à recevoir un approvisionnement régulier en puces, même si les quantités étaient limitées. Deuxièmement, cette stratégie a empêché nos clients de surstocker à la hâte en réponse à la crise. Enfin, et c'est important, pendant toute cette période, nous nous sommes engagés à ne pas augmenter nos prix, de sorte que la volatilité du marché n'a pas eu d'incidence négative sur notre structure tarifaire. Je pense que notre stabilité et notre croissance modeste en 2023 reflètent l'efficacité et la prévoyance de ces politiques.

Elektor : dans une interview accordée à Elektor il y a plusieurs années, vous avez déclaré que l'IA serait la prochaine grande tendance, ce qui s'est avéré exact. Selon vous, quelle sera la prochaine grande révolution dans le domaine de l'électronique ?
Teo Swee-Ann : dans le domaine de l'électronique, nous devons tenir compte de deux trajectoires distinctes. Tout d'abord, il y a celle menant à un écosystème virtuel ou de réalité mixte plus immersif. Imaginez un avenir où, dans les prochaines décennies, nous pourrions posséder des implants qui non seulement augmenteraient nos cinq sens, tels que la vision et l'audition, mais introduiraient également de nouvelles expériences sensorielles ou des voies cognitives jusqu'alors inexplorées par l'évolution naturelle. Cela signifierait que nos émotions, nos capacités cognitives et nos expériences sensorielles seraient considérablement améliorées ou modulées par l'IA. La réalisation de cette vision exige une innovation technologique importante, en particulier pour atteindre des capacités de calcul plus élevées avec une consommation d'énergie et des tensions réduites, avec les facteurs de forme les plus compacts. D'autre part, la deuxième orientation met l'accent sur l'application

Pour l'avenir, notre objectif principal reste la création de solutions pionnières qui donnent la priorité à l'efficacité énergétique et à des facteurs de forme plus petits.

À propos de Teo Swee-Ann

Teo Swee-Ann est le fondateur et le PDG de la société de semiconducteurs Espressif Systems, cotée à Shanghai. Il est titulaire d'une maîtrise en génie électrique de l'université nationale de Singapour. Teo a travaillé pour les fabricants de puces américains Transilica et Marvell Technology Group, ainsi que pour la société chinoise Montage Technology, avant de fonder sa propre entreprise en 2008.

de l'IA à la conception de l'électronique. À titre de comparaison, si l'IA est capable de faire fonctionner des véhicules de manière autonome, elle possède certainement le potentiel nécessaire pour concevoir des circuits sophistiqués. Cette évolution permet de relever un défi important dans notre domaine, à savoir la tendance à la suringénierie des conceptions. L'intégration de l'IA dans le processus de conception offre la possibilité de les affiner, de les rationaliser et de les rendre plus efficaces, accélérant ainsi la réalisation de la première trajectoire.

Elektor : quelle est votre vision pour Espressif ? Où voyez-vous l'entreprise dans, disons, cinq ans ?

Teo Swee-Ann : chez Espressif, nous nous positionnons avant tout comme une entreprise leader dans le domaine des puces AIoT. Cela dit, nos capacités vont bien au-delà du simple matériel. Nous avons un fort ancrage dans le développement logiciel, avec notre système logiciel propriétaire, ESP-IDF, et de vastes cadres de solutions répondant à l'intégration du cloud, à l'IA périphérique, au réseau maillé, aux applications audio et de caméra, et plus encore. Ce qui nous distingue vraiment, c'est notre communauté et notre écosystème florissants, où professionnels et amateurs se réunissent pour partager et développer des idées révolutionnaires. Pour l'avenir, notre objectif principal reste la création de solutions pionnières qui donnent la priorité à l'efficacité énergétique et à des facteurs de forme plus petits. En somme, nous espérons qu'Espressif représente une éthique distinctive et une attitude progressiste à l'égard de l'ingénierie, de l'innovation et de la promotion d'un esprit communautaire collaboratif.

VF : Maxime Valens — 230614-04

LIEN

[1] C. Valens, "The Reason Behind the Hugely Popular ESP8266?: Interview with Espressif Founder and CEO Teo Swee Ann on the new ESP32," Elektor Business, 1/2018.: https://elektormagazine.com/magazine/elektor-63

ESPRESSIF

SDK

démarrer avec ESP-DL
le SDK d'apprentissage profond

L'ESP32-S3 intègre l'accélération de l'IA qui vous permet de l'utiliser pour diverses applications d'apprentissage automatique. ESP-DL est le SDK d'apprentissage profond d'Espressif, qui vous permet de tirer parti des instructions d'accélération de l'IA pour construire des modèles d'apprentissage automatique optimisés. Ce SDK fournit des exemples de reconnaissance faciale d'humains et de chats. ESP-DL prend désormais en charge le compilateur TVM qui permet d'utiliser les modèles TensorFlow, PyTorch ou ONNX. Espressif propose également un portage du SDK TensorFlow Lite Micro de Google avec le support de l'accélération ESP32-S3, vous permettant d'utiliser des modèles TensorFlow Lite standard sur l'ESP32-S3.

https://github.com/espressif/esp-dl
https://github.com/espressif/tflite-micro-esp-examples

simuler l'ESP32 avec Wokwi

le jumeau numérique de votre projet

Uri Shaked, Wokwi

Wokwi est un simulateur pour les systèmes embarqués et les appareils IdO. Il fournit un environnement en ligne où vous pouvez configurer, déboguer et partager des projets ESP32 sans avoir besoin de matériel. Le simulateur fonctionne simplement dans un navigateur web et peut simuler toutes les puces de la famille ESP32 : l'ESP32 classique, ainsi que les versions S2, S3, C3, C6 et H2. Il est également possible de simuler d'autres familles de microcontrôleurs.

Wokwi vous permet de créer un jumeau numérique de votre projet : vous pouvez simuler différents appareils d'entrée et de sortie [1], tels que des écrans LCD, des capteurs, des moteurs, des LED, des boutons-poussoirs, des haut-parleurs et des potentiomètres, et vous pouvez même créer vos propres modèles de simulation pour de nouveaux appareils. Avec le simulateur, vous pouvez modifier et développer votre micrologiciel plus rapidement même si vous ne disposez pas du matériel, vous pouvez utiliser des outils de débogage puissants, partager vos projets et collaborer facilement avec d'autres développeurs.

Commencez votre aventure Wokwi

Vous pouvez utiliser votre langage de programmation préféré avec Wokwi. Si vous êtes débutant, MicroPython ou Arduino Core est probablement un bon choix. Les professionnels et les utilisateurs expérimentés peuvent utiliser ESP-IDF, Rust et des RTOS embarqués tels que Zephyr ou NuttX.
Il est recommandé de consulter quelques exemples existants pour avoir une idée de ce que vous pouvez réaliser avec Wokwi :

- MicroPython [2]
- ESP32 avec Arduino Core [3]
- Rust [4]
- DeviceScript (TypeScript pour ESP32) [5]

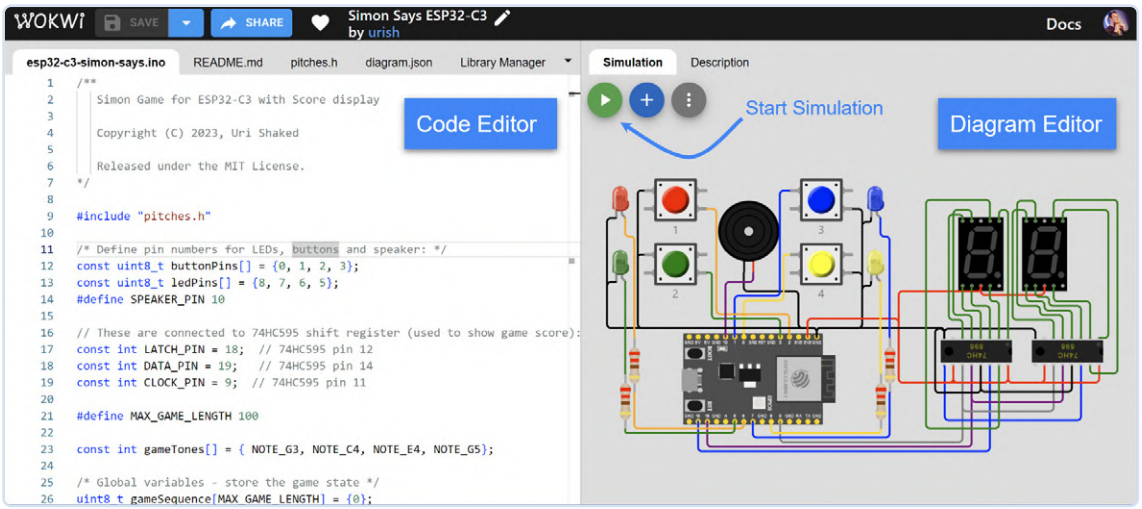


Figure 1. Le jeu « Jacques a dit » sur l'interface utilisateur de Wokwi.

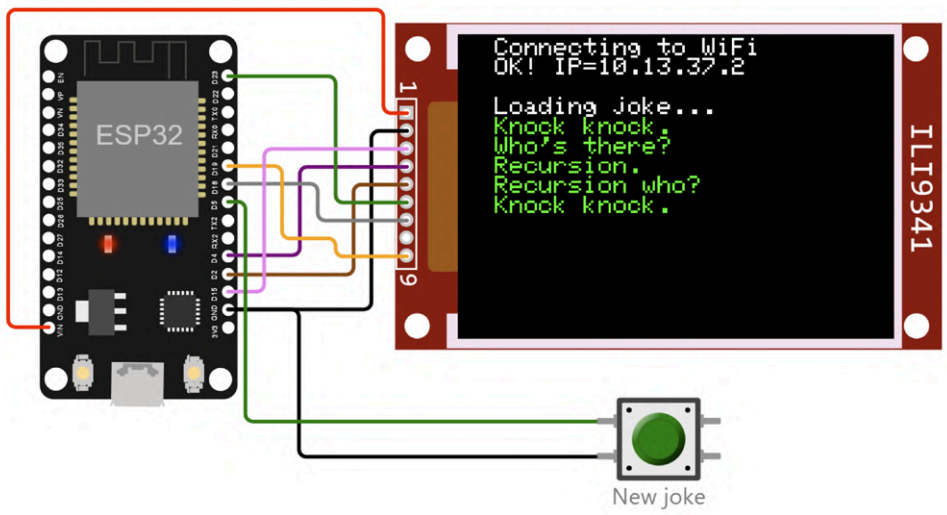


Figure 2. Exemple de simulation Wifi : client [8].

Ouvrez un exemple et appuyez sur le bouton vert *play* pour lancer la simulation et commencer à utiliser le projet. Certains projets disposent également d'un fichier `README` contenant plus d'informations sur le projet et sur la manière de l'utiliser. Pour créer un nouveau projet, rendez-vous sur [6]. Choisissez ensuite un microcontrôleur et un langage de programmation et cliquez sur *Start*.

Interface utilisateur de Wokwi

La version web de Wokwi se compose de deux parties : sur le panneau de gauche, vous pouvez éditer le code source (l'éditeur de code), et à droite, vous pouvez dessiner le schéma en ajoutant différents appareils/composants et en les connectant au microcontrôleur (*Diagram Editor*), voir la **figure 1**.

Diagram Editor

Ajoutez des éléments au diagramme en cliquant sur le bouton bleu + et en choisissant le composant souhaité dans la liste. Faites glisser les composants jusqu'à la position souhaitée. Pour dessiner un fil, cliquez sur la broche où le fil commence, puis sur la broche cible. Si vous voulez donner au fil un tracé spécifique, après avoir cliqué sur la broche de départ, cliquez sur l'endroit où vous voulez le connecter.
Pour des résultats plus précis, vous pouvez activer la grille en appuyant sur *G*. Cela aligne tous les composants et les fils sur une grille de 0,1 pouce (2,54 mm) - qui est l'espacement standard des broches d'un connecteur. Quelques raccourcis utiles : *D* duplique le composant sélectionné, *R* le fait pivoter, les chiffres *0* à *9* changent la couleur d'un fil, d'une LED ou d'un bouton-poussoir et si vous appuyez sur *Shift*, vous pouvez utiliser la souris pour sélectionner plusieurs composants en même temps. Pour plus de raccourcis, voir le manuel de l'éditeur de schémas [7].

Gestionnaire de bibliothèques

Utilisez le gestionnaire de bibliothèques (Library Manager) pour ajouter rapidement des bibliothèques Arduino standard à un projet. Les utilisateurs payants peuvent également télécharger leurs propres bibliothèques Arduino et les inclure dans leur projet.
Pour les autres langages de programmation, vous pouvez inclure des bibliothèques en modifiant les fichiers de configuration du projet (par exemple *Cargo.toml* pour Rust), ou en incluant le code source de la bibliothèque directement dans votre projet (par exemple pour MicroPython).

Simulation Wifi

L'ESP32 dispose d'une fonction Wifi intégrée, c'est pourquoi il est populaire dans les projets IdO. Wokwi simule la fonction Wifi de la puce. Le simulateur (**figure 2**) crée un point d'accès virtuel appelé *Wokwi-GUEST*. C'est un point d'accès ouvert (c'est-à-dire sans mot de passe). Le point d'accès est connecté à internet, vous pouvez donc connecter les projets simulés à des serveurs HTTP, des brokers MQTT et d'autres services cloud tels que Firebase, ThingsSpeak, Blynk et Azure IoT (**figure 3**).

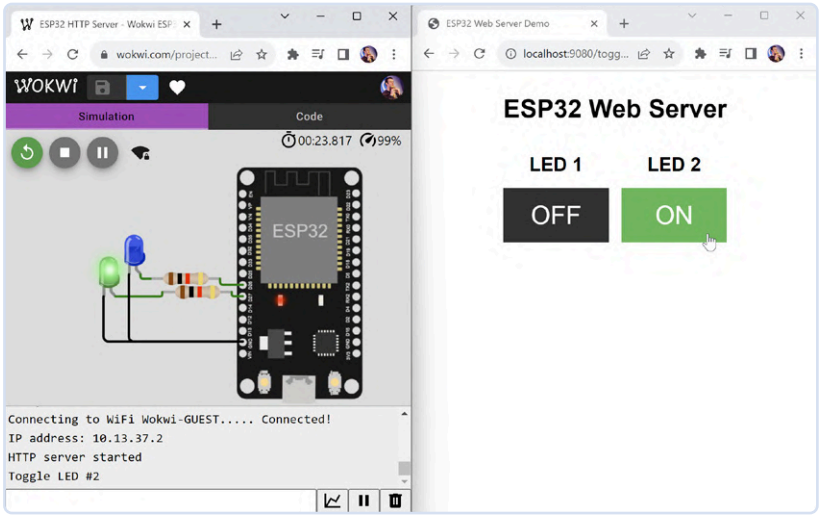


Figure 3. Exemple de simulation Wifi : serveur web [9].

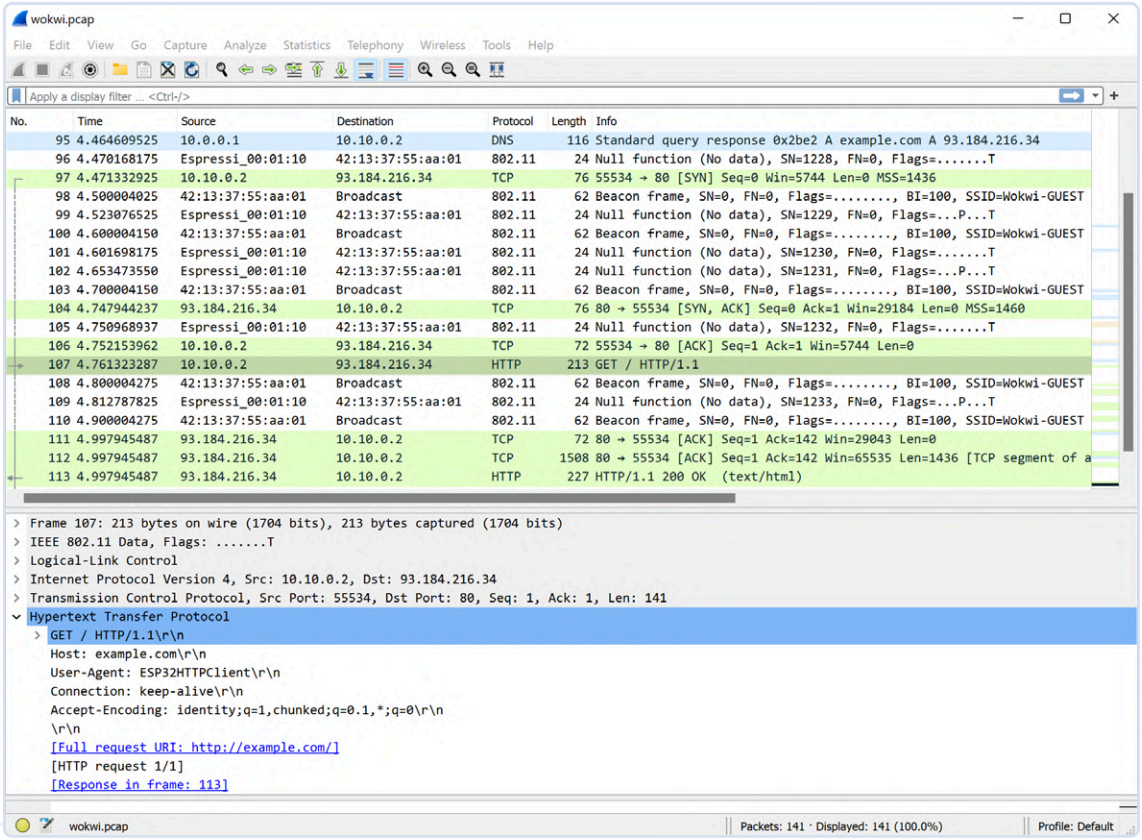


Figure 4. Visualisation du trafic de données Wifi.

Les utilisateurs payants peuvent également utiliser une passerelle IdO spéciale [10] sur leur ordinateur, qui leur permet de se connecter à des serveurs locaux depuis la simulation, et de connecter leur ordinateur à un serveur HTTP (ou tout autre type de serveur) fonctionnant dans le simulateur. Sous le capot, Wokwi simule une pile réseau complète : cela va de la couche MAC 802.11 la plus basse, en passant par les couches IP et TCP/UDP, jusqu'aux protocoles tels que DNS, HTTP, MQTT, CoAP, etc. Nous pouvons visualiser les données brutes du réseau [11] dans un analyseur de protocole réseau tel que Wireshark (**figure 4**).

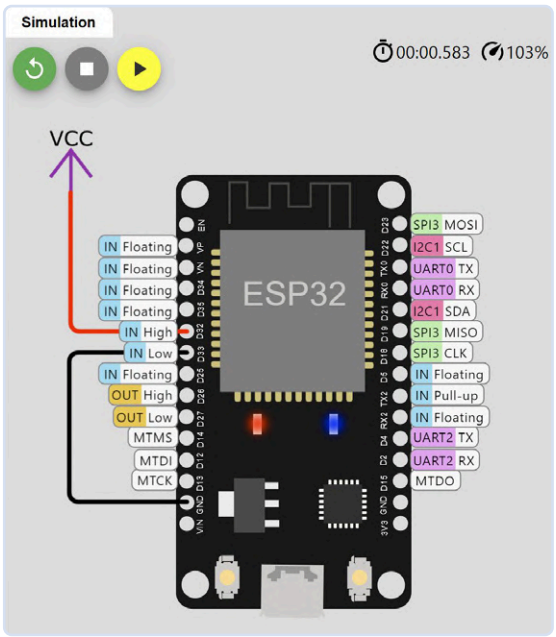


Figure 5. Simulation en pause : on peut voir l'état de chaque broche.

Pin Function Debugger

L'ESP32 offre une configuration flexible des broches : le logiciel peut choisir quel périphérique est connecté à chacune des broches en configurant IO MUX, GPIO Matrix et les périphériques basse consommation/RTC. Définir la configuration des broches au niveau logiciel peut être très pratique, mais cela complique la tâche de savoir quelle est la configuration réelle des broches dans le micrologiciel. Heureusement, dans Wokwi, nous pouvons simplement pauser la simulation et voir immédiatement la fonction et l'état actuels de chaque broche (**figure 5**).

Intégration avec Visual Studio Code

L'utilisation de Wokwi dans un navigateur web est très pratique pour apprendre le prototypage rapide, et pour partager des projets. Mais pour les projets plus complexes, il est recommandé d'utiliser Wokwi en combinaison avec votre EDI et outils de développement habituels. Wokwi s'intègre parfaitement à Visual Studio Code. Il suffit d'installer l'extension *Wokwi For VS Code* [12] et de créer un simple fichier de configuration [13] qui indique à Wokwi où se trouve votre micrologiciel compilé.

Vous pouvez également intégrer Wokwi au débogueur intégré de VS Code en ajoutant quelques lignes à sa configuration [14]. Contrairement au matériel réel, Wokwi dispose d'un nombre illimité de points d'arrêt, ce qui permet un débogage efficace (**figure 6**).

Pour les projets IdO, vous pouvez utiliser le transfert de port TCP [15]. Cela vous permet de connecter votre ordinateur à un serveur web (ou tout autre type de serveur TCP) fonctionnant sur la puce ESP32 simulée.

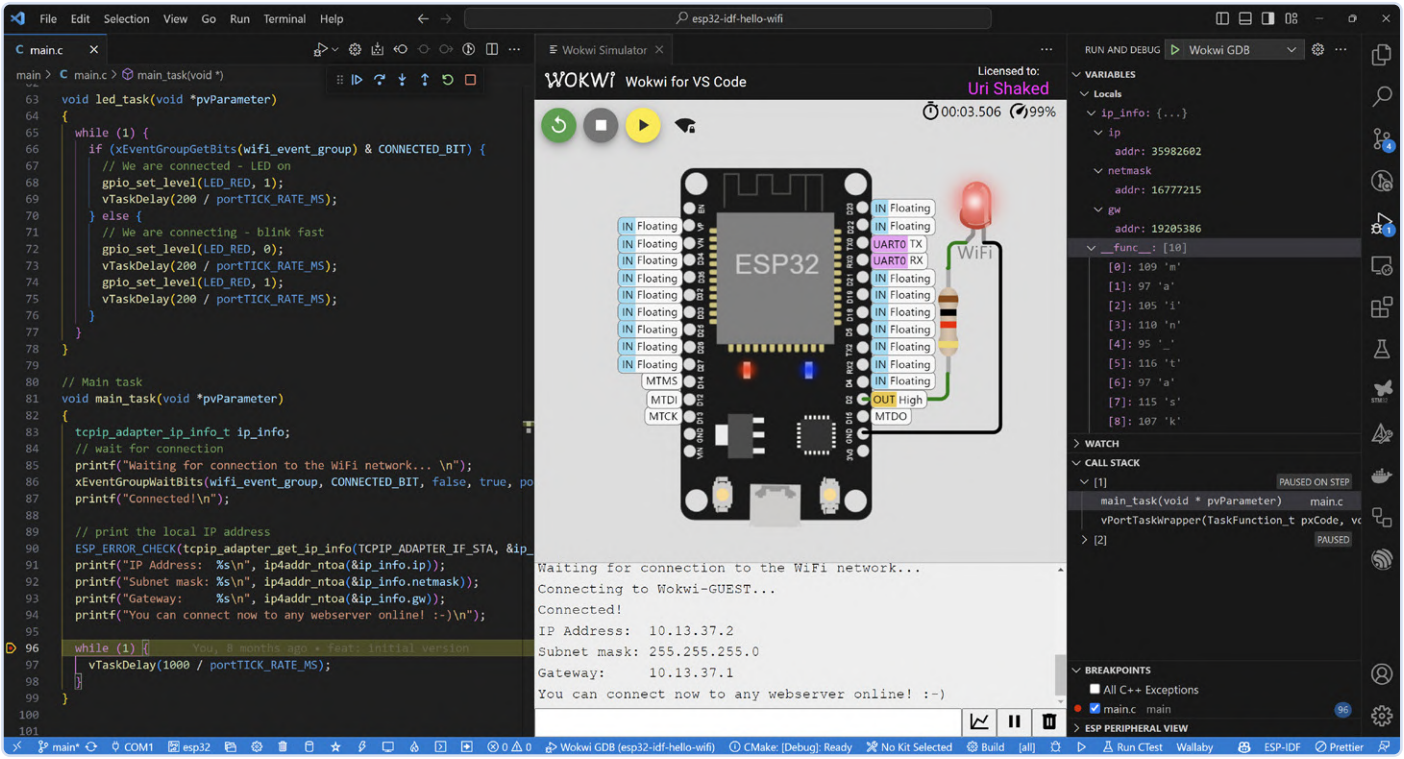


Figure 6. Débogage d'un projet ESP-IDF avec VS Code et Wokwi.

Intégration avec l'EDI Espressif

Les adeptes de l'EDI Espressif peuvent également créer une configuration de démarrage pour exécuter un projet dans le simulateur. La sortie (UART) du programme est alors envoyée vers la console de l'EDI. Voir Comment utiliser le simulateur Wokwi avec Espressif-IDE [16] pour plus de détails. [16].

Custom Chips

Custom Chips nous permettent d'étendre les fonctions de Wokwi avec de nouveaux composants. Nous pouvons simuler des capteurs, des écrans, des mémoires, des équipements de test (**figure 7**) et même du matériel personnalisé.

Pour créer notre propre circuit intégré, nous définissons son brochage dans un fichier JSON et nous écrivons du code pour implémenter la logique de la puce. Le code est généralement écrit en C, et l'API ressemble aux couches d'abstraction matérielle (HAL) courantes pour les puces intégrées, de sorte que les développeurs de micrologiciels seront en mesure de l'utiliser. D'autres langages tels que Rust, AssemblyScript et Verilog font également l'objet d'une prise en charge expérimentale. Pour plus d'informations et d'exemples, voir la documentation *API Chips* [17].

Wokwi pour les circuits intégrés

La simulation peut faire gagner beaucoup de temps lors du prototypage, mais elle peut aussi aider à la recherche de défauts pendant le développement. L'intégration continue (CI) est une approche moderne du développement logiciel : construire et tester le projet après chaque modification du code. Mais les tests sur du matériel réel prennent du temps et sont difficiles à automatiser. La situation est encore plus délicate si l'on souhaite une intégration complète du système. Pensez aux tests automatisés du Wifi ou à la capture de l'image sur un écran LCD. Il est très difficile d'y parvenir de manière stable et fiable.

Wokwi pour les CI résout tout cela. Si vous utilisez GitHub Actions, `wokwi-ci-action` [18] vous permet d'intégrer la simulation dans votre flux de travail existant avec seulement quelques lignes de code. Pour les autres environnements de CI, `wokwi-cli` [19] fournit une interface de ligne de commande simple pour exécuter et tester des micrologiciels par simulation. Le serveur de simulation de CI est disponible en tant que service cloud géré et peut également être déployé sur votre poste de travail.

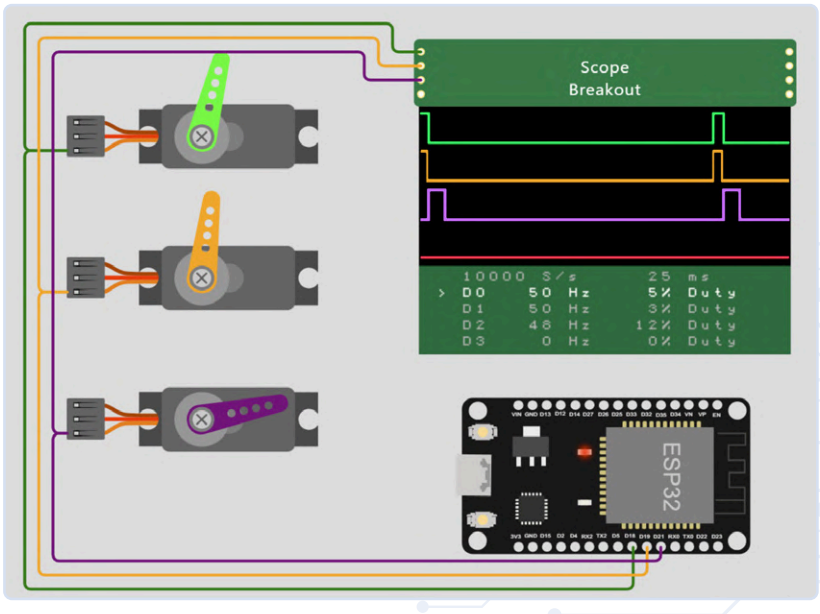


Figure 7. Un scope numérique créé par un utilisateur avec l'API Custom Chips.

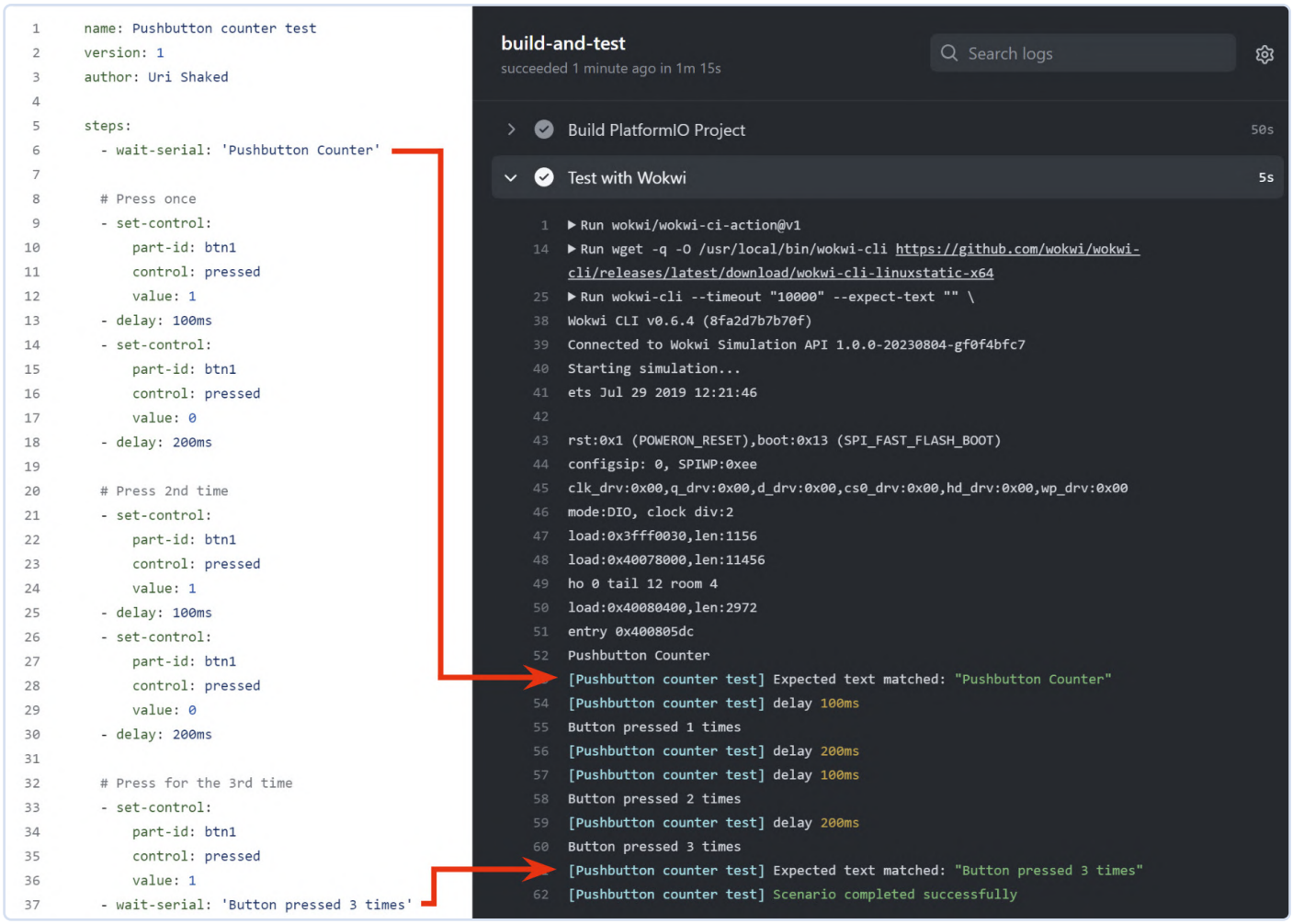


Figure 8. Le code d'exemple d'automatisation [20] (à gauche) avec la sortie de GitHub Action correspondante (à droite).

Scénarios d'automatisation

Les scénarios d'automatisation (**figure 8**) vous permettent d'effectuer des tests et des vérifications complexes sur le micrologiciel. Ils sont faciles à écrire : chaque scénario consiste en un fichier YAML contenant une série de commandes telles que l'appui sur un bouton, le réglage d'une valeur de capteur de température ou la recherche d'une chaîne de caractères dans la sortie série.

Limites

Wokwi offre de nombreuses fonctions, mais il a aussi quelques limites. L'objectif principal du simulateur est d'exécuter du code embarqué. Le simulateur est principalement numérique et la prise en charge de la simulation analogique est limitée. Wokwi ne vous avertira pas si vous oubliez une résistance de limitation de courant ; il ne dégage pas non plus de fumée magique (mais qui sait ce que l'avenir nous réserve...).

230523-04

Questions ou commentaires ?

Contactez Elektor (redaction@elektor.fr).

À propos de l'auteur

Uri Shaked est un maker chevronné. Il travaille actuellement sur Wokwi, une plateforme en ligne de simulation de systèmes IdO et embarqués, et sur Tiny Tapeout, qui rend la fabrication de vos propres ASIC abordable et accessible.

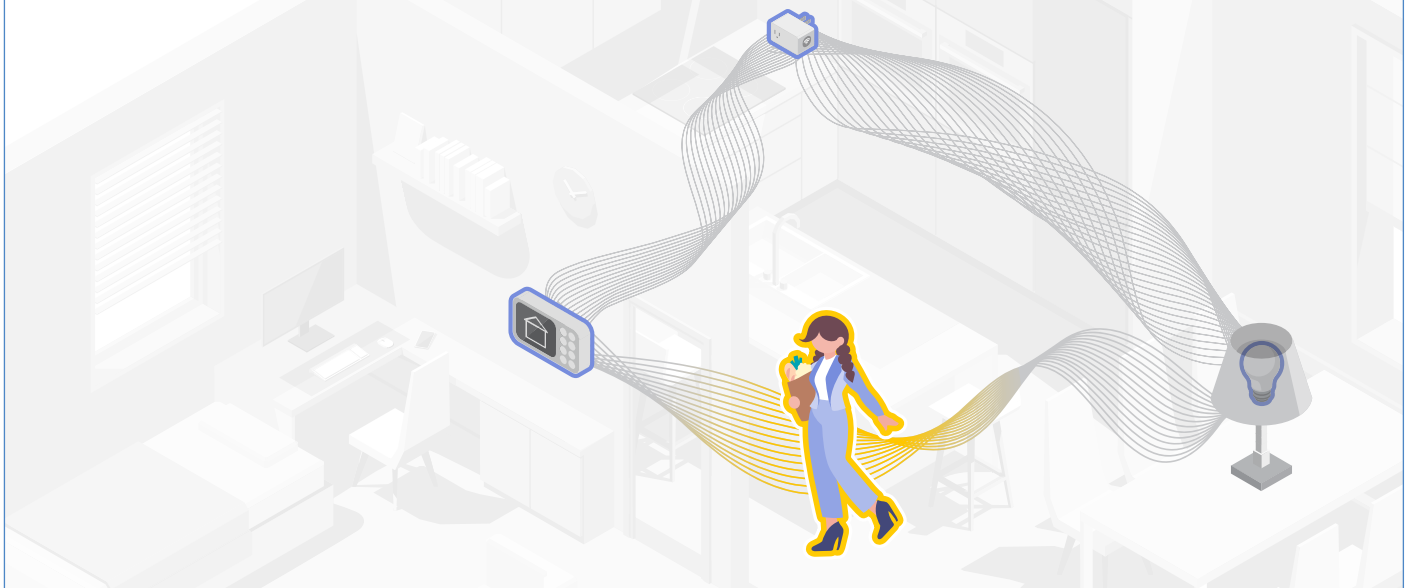
Produits

- **ESP32-C3-DevKitM-1**
  www.elektor.fr/20324
- **Koen Vervloesem,** *Getting Started with ESPHome* **(Elektor, 2021)**
  www.elektor.fr/19738

LIENS

[1] Wokwi Supported Hardware : https://docs.wokwi.com/getting-started/supported-hardware
[2] Wokwi Online Embedded Python Simulator : https://wokwi.com/micropython
[3] Wokwi Online ESP32 Simulator : https://wokwi.com/esp32
[4] Wokwi Online Embedded Rust Simulator : https://wokwi.com/rust
[5] Wokwi Online Devicescript Simulator: https://wokwi.com/devicescript
[6] Page de démarrage d'un nouveau projet : https://wokwi.com/projects/new
[7] Guide du Diagram Editor : https://docs.wokwi.com/guides/diagram-editor#keyboard-shortcuts
[8] Exemple de simulation Wifi : https://wokwi.com/projects/342032431249883731
[9] Exemple de serveur Web ESP32 : https://wokwi.com/projects/320964045035274834
[10] Application de passerelle privée : https://docs.wokwi.com/guides/esp32-wifi#the-private-gateway
[11] Visualisation du trafic de données Wifi avec Wireshark : https://tinyurl.com/wsviewtraffic
[12] Esxtension Wokwi for VS Code : https://tinyurl.com/wokwivsext
[13] Configuring Your Project: https://docs.wokwi.com/vscode/project-config
[14] Débogage du code : https://docs.wokwi.com/vscode/debugging
[15] TCP Port Forwarding : https://docs.wokwi.com/vscode/project-config#iot-gateway-esp32-wifi
[16] Comment utiliser le simulateur Wokwi avec Espressif-IDE : https://tinyurl.com/wokwiespide
[17] Démarrer avec Wokwi Custom Chips C API : https://docs.wokwi.com/chips-api/getting-started
[18] Wokwi CI Action : https://github.com/wokwi/wokwi-ci-action
[19] Wokwi CLI : https://github.com/wokwi/wokwi-cli
[20] Automation Scenario Code : https://tinyurl.com/pushcnttst

Révolutionnez votre gamme de produits avec WiFi Motion™,
désormais disponible sur les puces Wi-Fi Espressif.



Explorez l'avenir de la détection de mouvement.
Contactez-nous dès maintenant sur **www.cognitivesystems.com**

COGNITIVE

Figure 1. Le kit ESP32-S3-BOX-3 déballé. Ne sont pas représentés le câble USB et le minuscule module de LED RVB avec les quatre fils de connexion. La nectarine n'est pas incluse.

essai de l'ESP32-S3-BOX-3

une plateforme de développement AIoT complète

Clemens Valens (Elektor)

L'ESP32-S3-BOX-3 d'Espressif est une plateforme permettant de développer des applications telles que des assistants personnels, des haut-parleurs intelligents et d'autres appareils à commande vocale. Voyons cela de plus près.

Basé sur un ESP32-S3, le kit ESP32-S3-BOX-3 [1] est présenté comme « Votre prochain outil de développement AIoT ». AIoT signifie Artificial Intelligence of Things (intelligence artificielle des objets) et est étroitement lié à l'IdO, qui signifie Internet des objets (comme vous le saviez certainement déjà), mais ne doit pas être confondu avec ce dernier. Le kit a la forme d'une petite console dotée d'un écran tactile. Les applications suggérées pour ce kit sont les chatbots d'IA, Matter (un protocole unificateur pour la domotique), les contrôleurs de robots et les dispositifs de capteurs intelligents.

À l'intérieur de la boîte

L'ESP32-S3-BOX-3 (**figure 1**) se présente sous la forme d'une de ces boîtes qui s'ouvrent lentement (pour améliorer l'effet Wow !, m'a dit quelqu'un qui avait travaillé chez Apple). Lorsqu'on l'ouvre, on voit un module d'affichage (55 mm × 50 mm) avec une sorte de support, et un câble USB-C court (environ 30 cm). Il ne s'agit là que de la partie supérieure. Sous le support s'en cache un second, un peu plus petit. Sous le module d'affichage, vous trouverez encore un autre support plus petit et même un quatrième beaucoup plus petit qui se branche sur une plaque à essai.

Le câble USB est accompagné d'un petit sachet contenant un petit module LED RVB et quatre fils de connexion.

Module d'affichage

Le module d'affichage est étonnamment lourd (60 g) pour un si petit objet (LHP 55 mm × 50 mm × 12 mm). C'est l'unité qui fournit toute la puissance : un module ESP32-S3-WROOM-1-N16R16V doté de Wifi 2,4 GHz (802.11b/g/n) et de Bluetooth 5 (LE). L'écran lui-même est un écran TFT tactile de 2,4 pouces, 300 × 240 pixels. Au cas où vous vous poseriez la question, il n'y a pas de batterie à l'intérieur (**figure 2**).

Sur la face avant se trouvent, outre l'écran, deux minuscules trous (les microphones) et un cercle rouge (le bouton de retour). Il y a deux boutons-poussoirs (Boot & Rst) et un connecteur USB-C sur le côté gauche et, sur

le côté droit, quelque chose qui ressemble à un emplacement pour carte microSD, mais qui est en réalité un haut-parleur. Sur la face supérieure se trouvent un bouton de mise en sourdine et deux diodes électroluminescentes ; un connecteur PCIe dépasse sur la face inférieure. Il n'y a rien à l'arrière du module.

Les supports

Le connecteur PCIe se branche avec une prise sur l'un des supports. Comme indiqué précédemment, il y en a quatre, chacun ayant des fonctions différentes (**figure 3**). Celui qui est représenté sur le dessus de la boîte est le DOCK. Il possède deux prises d'extension Pmod (2 × 6 en-tête femelle) à l'arrière ainsi qu'une prise USB-C pour l'alimentation (in & out). Une prise hôte USB-A est disponible sur le côté gauche. Une étiquette indique le nom des signaux accessibles sur les connecteurs Pmod.

Support de capteur

Le plus grand support est le SENSOR. Il comporte un capteur de température et d'humidité avec un minuscule interrupteur marche/arrêt à l'arrière, un emplacement pour carte microSD (pas de haut-parleur cette fois) à gauche et une prise d'alimentation USB-C à droite. Sur la face avant, on trouve un capteur IR et une LED, et une périphérique vraiment cool à mon avis : un radar. Ce support a de la place à l'intérieur pour une batterie de type 18650.

Le troisième support est le BRACKET, et il peut être fixé à quelque chose d'autre grâce aux deux vis qui dépassent de sa face arrière. Ce support dispose également de deux connecteurs Pmod et d'une prise USB-C. Il n'y a pas d'étiquettes ici, nous supposerons donc que les connecteurs Pmod sont câblés de la même manière que sur le DOCK. Le plus petit support, le BREAD, est un adaptateur PCIe vers plaque à essai à 24 contacts. Les broches qui entrent dans la plaque à essai sont étiquetées.

Première mise en marche

Lorsque vous mettez le module d'affichage en marche pour la première fois, il démarre dans ce qui s'avérera plus tard être le mode d'aide (*Help*). Ce mode vous fait passer par quelques écrans d'aide qui expliquent certaines fonctions de base de l'appareil. Il passe ensuite en mode normal dans lequel vous pouvez faire défiler six fonctions : *Sensor Monitor, Device Control, Network, Media Player, Help* et *About Us*.

Contrôle vocal

D'après les affichages d'aide, il est possible d'activer vocalement certaines choses en disant « ail i es pi » (*Hi ESP* prononcé en anglais lettre par lettre). Comme rien ne s'est produit lorsque j'ai essayé, j'ai téléchargé le guide de l'utilisateur [2]. Pour ce faire, un code QR est imprimé sur le fond de la boîte. D'après le manuel, la commande

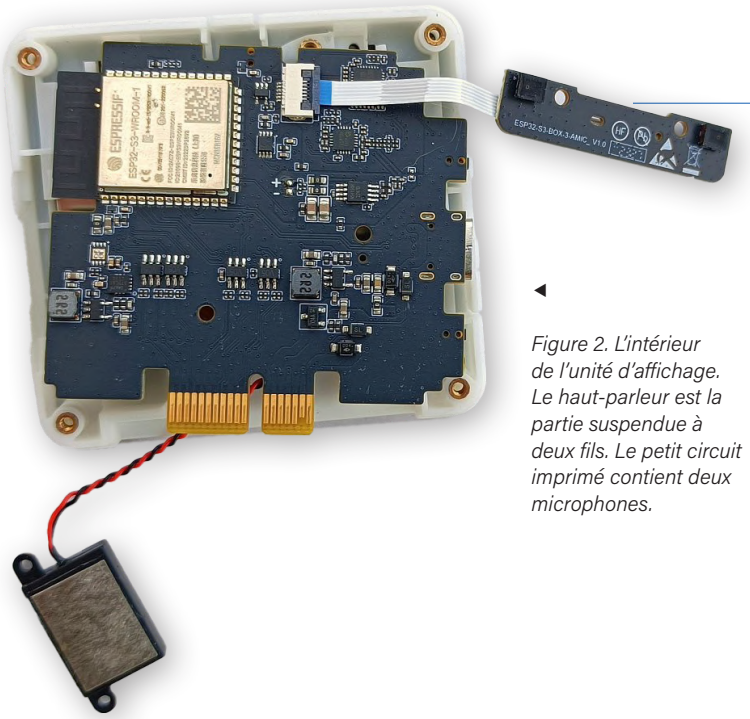


Figure 2. L'intérieur de l'unité d'affichage. Le haut-parleur est la partie suspendue à deux fils. Le petit circuit imprimé contient deux microphones.

vocale fonctionne avec tous les affichages (sauf si le voyant de mise en sourdine est allumé). Sachant cela, j'ai réessayé et j'ai réussi à le faire fonctionner plusieurs fois. Si vous y parvenez, vous entendrez un « ping » et l'écran affichera un microphone. Vous avez maintenant six secondes pour prononcer une commande. Après six secondes d'inactivité, l'appareil dit *Timeout, see you next time* (« trop tard, à la prochaine ») et retourne au mode dans lequel il se trouvait avant d'être activé. J'ai eu beaucoup de mal à réveiller l'appareil, mais une fois réveillé, il a reconnu mes commandes sans problème. Curieux.

Radar

J'ai découvert une fonction intéressante après avoir rempli ma tasse de café. L'écran s'était éteint pendant mon absence et s'est soudainement rallumé. Ce doit être du fait du radar intégré. Si vous ne bougez pas pendant un certain temps, l'écran s'éteindra également.

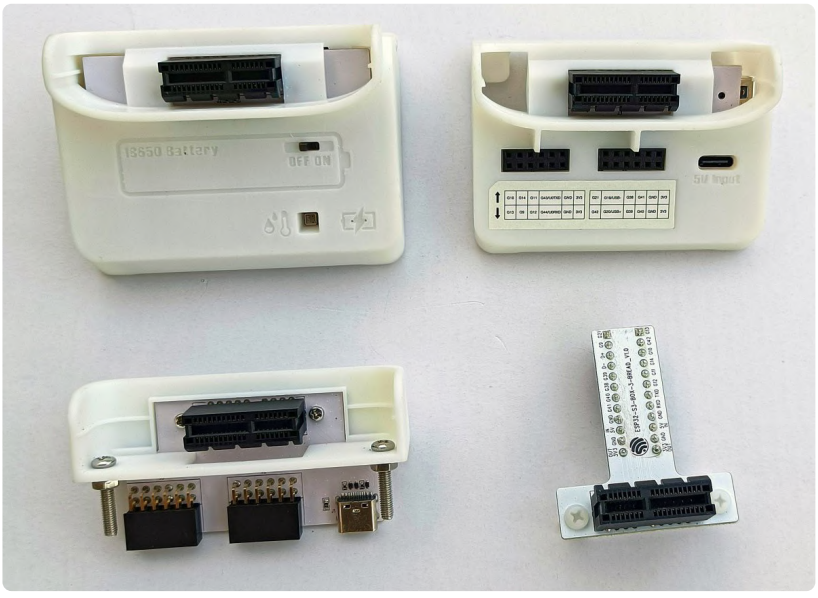
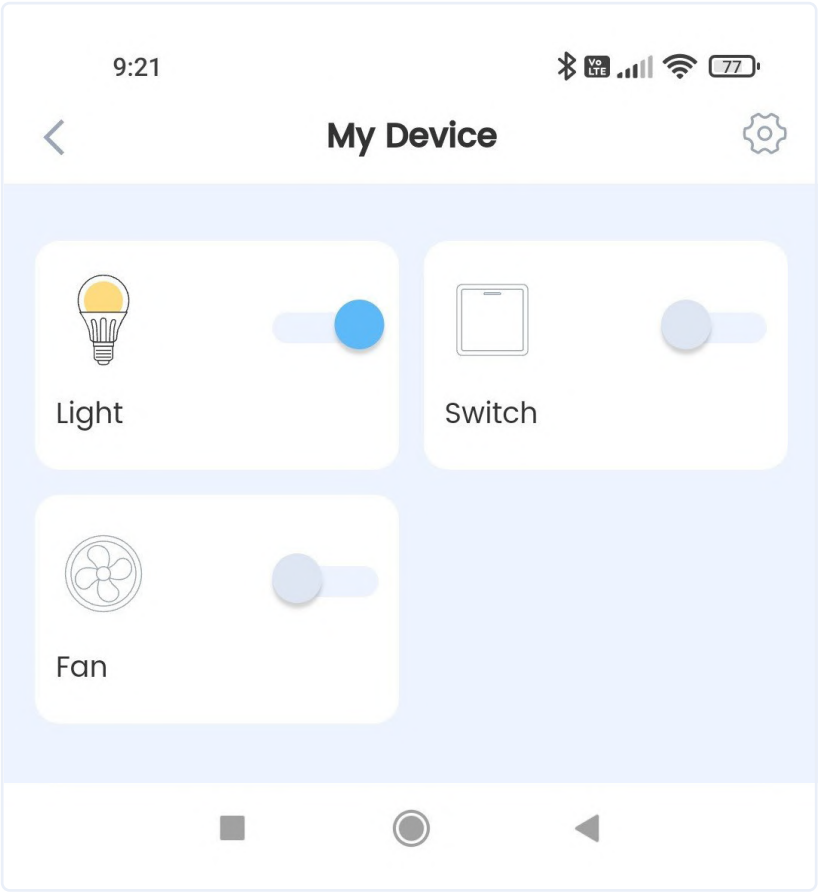


Figure 3. Quatre supports permettent toutes sortes d'applications.



▲

Figure 4. L'appli ESP BOX assure la connexion entre le kit et le service cloud Rainmaker d'Espressif.

L'appareil dans le cloud

Après avoir manipulé un peu les applis intégrées (éteignez puis rallumez l'écran au cas où vous l'auriez branché à chaud sur le support du capteur, sinon il ne fonctionnera pas), je suis passé à l'appli ESP-BOX. Vous pouvez l'obtenir via la page *Network* après avoir appuyé sur le bouton *To install APP* et scanné le code QR.

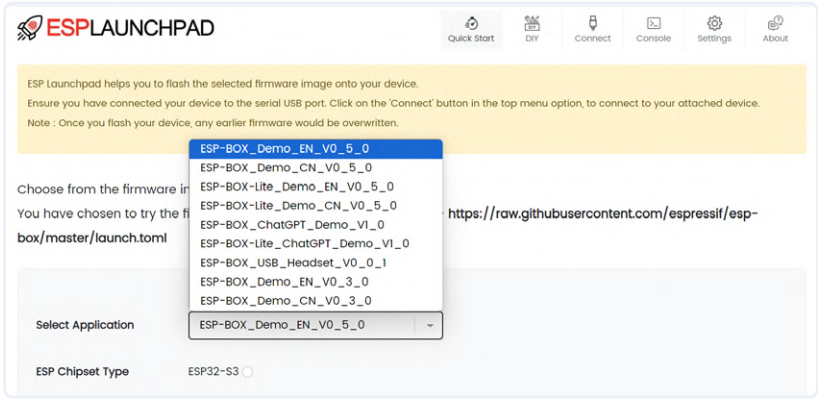
Avec l'appli, vous pouvez connecter le kit (et d'autres appareils) via un réseau Wifi au service cloud Espressif Rainmaker. Après avoir ajouté le dispositif à l'appli, celle-ci vous donne accès aux mêmes commandes que sur la page *Device Control* (lumière, interrupteur et ventilateur, mais pas *Air*, **figure 4**). Sur votre téléphone, vous pouvez maintenant basculer ces fonctions et le résultat s'affiche sur l'écran de l'ESP32-S3-BOX-3. Cependant, il semble que cette communication ne se fasse que dans un seul sens. Le fait d'appuyer sur un bouton sur l'écran ne met pas à jour l'application.

Si, dans l'application, vous appuyez sur l'icône du bouton au lieu de l'interrupteur à glissière, une page de paramètres s'ouvre. Vous pouvez y définir les broches GPIO contrôlées par le bouton et les commandes vocales. Vous pouvez également redéfinir les commandes vocales en tapant simplement une nouvelle phrase de commande. Le guide de l'utilisateur donne quelques conseils sur les bonnes commandes. Malheureusement, comme je n'ai pas réussi à réveiller l'appareil, je n'ai pas pu essayer de nouvelles commandes.

Je n'ai pas trouvé de document expliquant comment créer vos propres applications à utiliser avec l'application ESP-BOX.

▼

Figure 5. ESP Launchpad est un outil de programmation de firmware en ligne et un terminal série.



Port série

Sachez que vous pouvez contrôler ce qui se passe à l'intérieur de l'appareil si vous connectez l'unité d'affichage à un ordinateur et ouvrez un terminal série. Cette fonction s'est avérée très utile pour ce qui suit.

La plateforme ESP

Sur le fond de la boîte de l'ESP32-S3-BOX-3 se trouvent trois codes QR. L'un d'entre eux concerne l'application ESP Launchpad [3]. Elle ne fonctionne que sous Chrome (au moins sous Windows) et est destinée à reprogrammer le kit avec d'autres micrologiciels. Il semble que l'on puisse même y publier son propre firmware, afin que d'autres personnes puissent également l'essayer (**figure 5**).

Au début, il ne pouvait pas se connecter à mon appareil pour une raison quelconque, mais après l'avoir redémarré, j'ai obtenu une liste de programmes de démonstration.

Démo ChatGPT

Dans cette liste, j'ai sélectionné (bien sûr) *ESP-BOX_ChatGPT_Demo_V1_0*, je l'ai téléchargé avec empressement sur l'unité d'affichage et… rien. L'écran est resté noir. J'ai ensuite essayé la meilleure démo suivante, *ESP-BOX-Lite_ChatGPT_Demo_V1_0*, mais j'ai obtenu le même résultat. Après en avoir essayé d'autres, j'ai soudain compris (merci le port série) : ces fichiers ne sont pas pour l'ESP32-S3-BOX-3, ils sont destinés aux versions précédentes du kit, l'EPS32-S3-BOX (sans '-3') et l'ESP32-S3-BOX-Lite. Espérons que ce problème a été résolu lorsque vous lirez ces lignes.

Construisez vos propres applications

À ce stade, je me suis retrouvé sur GitHub, dans le dépôt qui prend en charge les trois versions de l'ESP32-S3-BOX [4]. Pourquoi n'ont-ils pas imprimé un grand code QR pour cette page sur la boîte ? Il n'est pas non plus mentionné dans le guide de l'utilisateur. Ce dépôt contient de nombreuses informations sur le kit ainsi que le code source des programmes d'exemple.

Nécessite ESP-IDF V5.1 ou plus récent

Il n'y a pas d'exécutables précompilés faciles à essayer, du moins je ne les ai pas trouvés, vous devez donc les compiler vous-même. Pour ce faire, vous devez d'abord

cloner le dépôt. Vous avez également besoin de ESP-IDF V5.1 ou plus récent.
La compilation et le téléchargement vers l'appareil ont pris un certain temps, mais une fois que c'était fait, j'étais finalement de retour au début de ce test lorsque l'unité d'affichage fonctionnait encore. À ma grande surprise, la commande vocale semblait mieux répondre et il était plus facile de réveiller l'appareil. La version du logiciel était toujours V1.1.1, mais ma version EPS-IDF était devenue « v5.11-dirty » (elle était « v5.2-dev-2164-g3befd5fff7-dirty »).

Démo ChatGPT, deuxième essai

Maintenant que j'avais un environnement de développement installé pour l'ESP32-S3-BOX-3, j'ai décidé d'essayer de compiler la démo ChatGPT en suivant les instructions fournies. Tout s'est bien passé.
Après avoir compilé et téléchargé le programme sur l'unité d'affichage, la démo a démarré normalement et a présenté des instructions sur la façon de la configurer [5]. Une fois cela fait, elle s'est connectée à mon réseau Wifi et j'ai pu commencer à lui parler. Cependant, le chatbot n'a jamais répondu à mes questions, mais a affiché le message *API Key is not valid* (clé API non valide). Dans le terminal série, j'ai trouvé cette ligne :

```
E (369916) OpenAI: You exceeded your current quota, please check your plan and billing details.
```

Il s'avère que vous avez besoin de crédit sur votre compte ChatGPT pour utiliser les clés API. Mon compte avait expiré. Vous obtenez un peu de crédit gratuitement lorsque vous créez un compte pour la première fois (un numéro de téléphone est nécessaire). Comme je ne voulais pas mettre en place un plan de paiement, je me suis arrêté ici.

Exemple d'affichage d'image

Enfin, j'ai décidé d'essayer un dernier exemple, l'affichage d'images. La description ne dit pas ce qu'il fait, ce serait donc une bonne surprise. Cette fonctionnalité est un peu plus légère que les autres, et donc la compilation et le téléchargement sont plus rapides. Le résultat ? Une liste de six smileys parmi lesquels vous pouvez en choisir un à afficher (**figure 6**).

Complexe et très intéressant

Ce test de produit m'a pris un peu plus de temps que je ne l'avais prévu. La raison en est la complexité du produit. L'ESP32-S3-BOX-3 a certainement beaucoup à offrir, mais pour en tirer quelque chose d'utile, vous devez investir un peu de temps.
L'endroit où commencer est le dépôt GitHub et non les codes QR à l'arrière de la boîte. Sur GitHub, vous trouverez non seulement le guide de l'utilisateur, mais aussi d'autres documents utiles et des programmes d'exemple. J'aurais gagné beaucoup de temps si j'avais su cela dès le départ. Bien qu'il y ait beaucoup d'informations, on a l'impression qu'elles sont incomplètes. Un tutoriel sur la façon de travailler avec l'application ESP-BOX ou sur la façon de créer une application à commande vocale à partir de zéro serait apprécié. Il n'y a pas grand-chose sur l'AIoT, pourtant c'est bien de cela qu'il s'agit avec ce produit, n'est-ce pas ? En ce qui concerne le matériel, tout ce que je peux dire, c'est qu'il est vraiment bien fait. La qualité de fabrication est bonne, les supports s'adaptent parfaitement, les boutons fonctionnent, l'écran est élégant, le tactile répond bien, et le tout est emballé dans une boîte de qualité. L'unité d'affichage, avec ou sans support, constitue un gadget sympa sur un bureau, ou dans une chambre ou un salon.

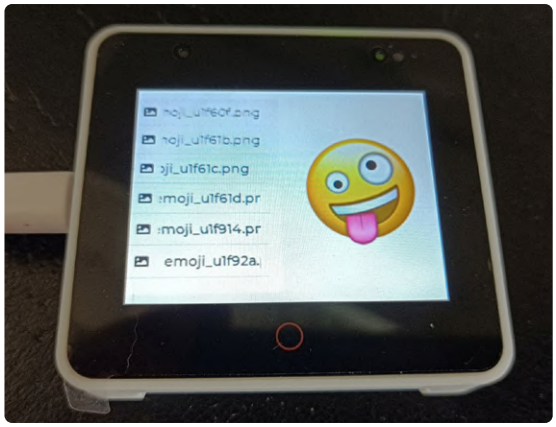


Figure 6. L'intelligence artificielle des objets (AIoT), encore un long chemin à parcourir ?

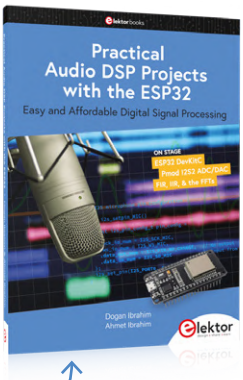
VF : Maxime Valens — 230555-04

Questions ou commentaires ?

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

Produits

- **ESP32-S3-BOX-3**
  www.elektor.fr/20627
- **Arduino Nano ESP32**
  www.elektor.fr/20562
- **Projets DSP audio pratiques avec l'ESP32**
  www.elektor.fr/20558



LIENS

[1] ESP32-S3-BOX-3 : https://www.espressif.com/en/news/ESP32-S3-BOX-3
[2] Guide de l'utilisateur : https://qr10.cn/CoahPA
[3] Plateforme ESP : https://qr10.cn/DCgKrD
[4] ESP32-S3-BOX-3 sur GitHub : https://github.com/espressif/esp-box
[5] Clé secrète ChatGPT : https://platform.openai.com/account/api-keys

Bien s'équiper pour mieux travailler

conseils et astuces des ingénieurs d'Espressif

Le labo d'électronique : c'est là que la magie opère. Êtes-vous curieux de savoir ce que d'autres électroniciens créatifs ont dans leur espace de travail ? Vous souhaitez obtenir des conseils sur l'aménagement et l'organisation de votre établi électronique ? Quelques ingénieurs d'Espressif vous proposent quelques idées.

Omar Chebib
Localisation : Shanghai, Chine

Organisez et prenez des notes

Mon espace de travail est mon coin bureau à la maison. Il n'est peut-être pas très grand, mais je m'efforce de le garder toujours propre et organisé. Je ne garde que les outils, appareils et composants dont j'ai besoin pour mes tâches actuelles. De plus, le fait de les placer systématiquement à l'endroit prévu me permet de rester efficace dans mes tâches. Naturellement, cela nécessite aussi une étagère bien organisée qui comprend des compartiments séparant les composants électroniques, avec des composants traversants et CMS, ainsi que les outils embarqués, avec un analyseur logique, quelques puces ESP32, ou même des dispositifs I2C. J'ai ainsi pu travailler sur plusieurs projets impliquant des portes logiques, un processeur Z80 8 bits, un FPGA, des soudures traversantes et en surface, y compris le "terrifiant" boîtier BGA.

Conseils : en ce qui concerne le matériel, ne gardez sur votre établi que les outils, les appareils et les câbles dont vous avez besoin pour la tâche en cours, n'achetez de nouveaux outils que lorsque vous en avez besoin et nettoyez après avoir terminé. En ce qui concerne les logiciels, vérifiez toujours la version utilisée, même lorsqu'il s'agit d'un projet local. Il n'est jamais trop tard pour commencer à gérer les versions d'un projet déjà existant. D'une manière plus générale, lorsque vous passez d'un projet à l'autre, gardez une note expliquant ce que vous avez déjà fait, où vous vous êtes arrêté et quelle est la prochaine étape. Cela vous aidera beaucoup à vous remettre au travail. Projets en cours : pendant mon temps libre, j'ai conçu un ordinateur Zeal 8-bit - un ordinateur basé sur le Z80 - à partir de zéro, de la conception du circuit imprimé au logiciel. Dans mon boulot, j'implémente l'émulation de l'ESP32-C3 sur QEMU.

Outils
- Analyseur logique
- Multimètre
- Oscilloscope
- Microscope numérique
- Station de soudage + flux
- Un bon extracteur de fumées !
- Ordinateur Linux

Liste d'envies
- Plaque chauffante pour le soudage
- Meilleur microscope
- Meilleure chaise

Jeroen Domburg
Localisation : Singapour

Un désordre organisé

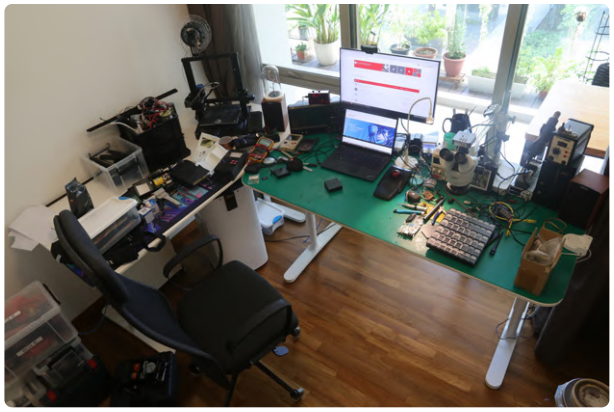
Mon établi a tendance à être « ordonné chronologiquement ». En d'autres termes, il s'agit d'une grande quantité d'objets, les plus récents étant placés en haut et les plus anciens en bas. Je le réorganise lorsque je manque d'espace pour travailler, lorsque j'oublie ce qui se trouve en bas ou lorsqu'il y a des choses à jeter. Si vous êtes aussi ce genre de personne, vous ne devez pas avoir honte d'avoir un « espace de travail désordonné » : à moins que vous ne le partagiez avec quelqu'un d'autre, votre espace de travail peut ressembler à ce que vous voulez qu'il soit ! Il semble que vous fonctionniez mieux en utilisant votre mémoire pour stocker et retrouver vos outils avec le moins d'effort possible, ce qui vous rend probablement plus efficace que les personnes « toujours en ordre ». Le stockage à long terme est une autre histoire. Je suis la devise « si je ne sais pas que je l'ai, je pourrais aussi bien ne pas l'avoir », mais je ne veux pas devoir me souvenir de chaque chose que j'ai. C'est pourquoi je stocke tous mes composants, mes anciens projets, etc., dans des boîtes numérotées, avec un fichier (très bien sauvegardé !) sur mon PC qui m'indique où se trouve chaque composant. Cela me permet de retrouver facilement un composant ou un outil lorsque j'en ai besoin. À propos de ces outils : si vous avez l'intention de souder des CMS, procurez-vous un bon microscope ; si vous pouvez en avoir un binoculaire, c'est encore mieux ! Je trouve que le fait d'avoir un microscope permet de mieux stabiliser les mains, et donc de souder les petits composants sans problème. Évidemment, un bon fer à souder est également utile, et les bons fers à souder sont de moins en moins chers. Même si le Weller WTCP de votre père était excellent à l'époque, un Pinecil ou un TS-100 moderne et bon marché le surpassera pour une fraction du prix.

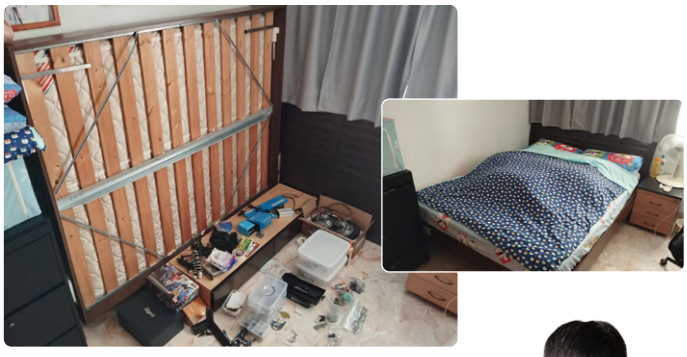
Outils

- Ordinateur. Impossible de programmer des puces sans ordinateur.
- Station de soudage moderne. Mon préféré est un Aixun T3A avec une poignée T245 .
- Purificateur d'air
- Microscope optique binoculaire
- Oscilloscope
- Analyseur logique
- Alimentation électrique de labo

Liste d'envies

- Meilleur bloc d'alimentation pour le labo
- Espace où je peux faire des tâches « salissantes » (par exemple, impression de résine, peinture, etc.)
- Plus de temps libre pour expérimenter l'électronique





Kai Jie Tan
Localisation : Singapour

Commencez n'importe où

Un espace de travail professionnel entièrement équipé, avec du matériel posé sur votre établi et un tableau de rangement rempli d'outils à portée de main, est vraiment l'objectif ultime, mais vous n'en avez pas besoin pour commencer. Il y a des années, j'ai commencé avec une boîte à outils et un kit d'électronique que mon école d'ingénieurs exigeait que tout le monde achète. Par manque de place, ils sont toujours rangés dans les armoires de ma chambre. En tant que jeune adulte à Singapour, vivant dans une chambre chez mes parents, il s'agit d'une préoccupation majeure. Les prix des logements étant élevés et les délais d'attente pour obtenir un logement social étant longs, la plupart des jeunes adultes ne quittent le domicile de leurs parents que lorsqu'ils se marient. Comme j'ai commencé à manquer de place avec chaque projet que je construis, surtout ceux de grandes dimensions (comme les accessoires de cosplay et les scooters électriques), je commence à avoir des outils dans les endroits les plus inattendus. Par exemple, mon imprimante 3D se trouve au-dessus de mon armoire, à 2 m du sol. Au plus fort des restrictions de COVID-19, j'ai perdu mon accès à un Fab Lab et j'ai dû tout faire à la maison. Cela m'a amené à construire un lit Murphy avec du matériel d'AliExpress. Cela a changé la donne, car cela m'a permis de garder mon travail en cours en toute sécurité sous mon lit - accessible, et peu de temps d'installation par rapport au fait de devoir tout garder dans des boîtes à outils. Il était supposé y avoir une phase 2 à ce projet, où j'ajoute des étagères à quatre barres, de sorte que les étagères deviennent les jambes du lit lorsqu'elles sont pliées. C'est quelque chose sur laquelle j'ai besoin de dormir (jeu de mots). Mon espace de travail est loin de ressembler à un Fab Lab professionnel de rêve. J'ai simplement beaucoup plus de boîtes à outils et de boîtes à compartiments d'électronique qu'auparavant ; cependant, j'ai rarement eu l'impression que cela m'empêchait de construire tout ce que je voulais. Quelques conseils : en passant d'un projet à l'autre, vous accumulerez plus d'outils et remplacerez les outils usés. Il n'est donc pas nécessaire d'attendre d'avoir un espace de travail entièrement équipé pour commencer. Commencez n'importe où.

Outils

- Multimètre Fluke 115
- Pince à dénuder Proskit 7 en 1
- Imprimante 3D (il est très valorisant d'en posséder une)

Liste d'envies

- Extracteur de fumées de soudure
- Purificateur d'air
- Enceinte pour imprimante 3D
- Un garage pour ranger correctement les outils

Pedro Minatel
Localisation: Braga, Portugal

Créez un sanctuaire

Je décris mon établi d'électronicien comme mon sanctuaire – un endroit où je peux créer et inventer des choses, même si la plupart d'entre elles resteront toujours des travaux en cours. Je m'efforce d'organiser la plupart de mes outils, mais parfois, le désordre est inévitable. Je suis fier de mes équipements ; certains sont même plus vieux que moi, comme mon pied à coulisse datant des années 50. Mon préféré, et l'un des outils les plus cruciaux de mon établi, est l'oscilloscope Rohde & Schwarz RTB2002. Conseils : Organisez vos composants ! Je préfère utiliser des boîtes de rangement peu coûteux (semblables à ceux des pilules). Toutefois, pour les composants sensibles aux décharges électrostatiques, il est essentiel de les stocker dans des sacs ESD appropriés. J'utilise une étiqueteuse pour m'assurer que toutes mes pièces sont correctement rangées. Projet en cours : je travaille sur une autre carte de développement, cette fois-ci basée sur l'ESP32-C6.

Outils

- Oscilloscope (avec analyseur logique)
- Multimètre fiable
- Station de soudage de qualité pour les CMS
- Outils de base agréables
- Imprimante 3D

Liste d'envies

- Analyseur de spectre avec VNA (tel que le SVA1032X)
- Machine pick-and-place de table
- Four à refusion
- Alimentation numérique (60 V 10 A)

Show Off Your Electronics Workspace!

Would you like Elektor's editorial team to consider featuring your electronics workspace on Elektormagazine.com or in the pages of *Elektor Mag*? Use our online Workspace Submission Form to share details about the space where you innovate, design, debug, and program! **elektormagazine.com/workspaces**

230579-04

Jakob Hasse
Localisation : Chine, Shanghaii

Espace de travail avec la table de mahjong

Mon espace de travail est principalement utilisé pour tous les types de soudage, le soudage par refusion de circuits imprimés, et les réparations de tous types de produits électroniques. Je dispose une station de soudage clone T12 qui fonctionne très bien grâce à l'élément chauffant direct de la panne. Les fumées de soudure ne sont vraiment pas saines, c'est pourquoi j'ai aussi un ventilateur d'extraction des fumées que j'ai acheté sur Taobao, une plateforme de vente chinoise appartenant à Alibaba, pour environ 150 $ - de l'argent bien investi pour ma santé. Pour le soudage par refusion des circuits imprimés, j'ai une plaque chauffante dont je ne me souviens plus du prix. Mais lorsqu'il s'agit d'équipements de mesure et d'outils métalliques « ordinaires », j'achète de la qualité. Croyez-moi, la vie est trop courte pour utiliser des outils de mauvaise qualité !
Comme je vis actuellement à Shanghai, et que l'espace y est plutôt luxueux, mon espace de travail ne fait que 80×80 cm de large. C'est une table normale que j'ai commandée sur Taobao. Le plateau est en bambou, un matériau vraiment dur et durable. Il s'avère cependant que le revêtement se dissout au contact de l'IPA. Par conséquent, la prochaine chose que j'achèterai sera certainement un vrai tapis en silicone pour protéger mon etabli non seulement de la soudure, mais aussi de l'alcool isopropylique. La taille réduite de mon espace de travail présente un avantage : si j'ai des invités, je débarrasse le bureau et le déplace dans le salon. La taille est parfaite pour jouer au Mahjong.

Outils

- Station de soudage T12-clone
- Ventilateur d'extraction des fumées Weinan
- Plaque chauffante Vectech
- Kit de réparation iFixit
- Alimentation de laboratoire Maisheng MS-605D 300 W
- Alcool isopropylique
- Multimètre Gossen-Metrawatt Metraline DM62
- Tournevis LUX pour les types de vis les plus courants
- Pince multifonction Knippex

Liste d'envies

- Tapis en silicone
- Oscilloscope
- Station à air chaud
- Brucelles de rechange

l'histoire de l'ESP RainMaker

comment nous avons construit « votre » nuage IdO

Amey Inamdar, Espressif

Les appareils connectés au nuage constituent les éléments principaux de l'IdO. Cependant, avec les fournisseurs de solutions nuage complètes, les fabricants d'appareils perdent une partie du contrôle des données, et la mise en place d'une solution nuage en partant de zéro représente un effort considérable. Chez Espressif, nous nous sommes rendu compte de ces difficultés et avons décidé de proposer une solution qui permet d'obtenir le meilleur des deux mondes.

Le « I » de IdO est l'abréviation d'internet. L'internet, et donc la connectivité au nuage, fait partie intégrante des appareils connectés afin de bénéficier des avantages de la connectivité. Les appareils qui se connectent au nuage pour y envoyer des données et recevoir des commandes représentent une valeur ajoutée, non seulement pour les utilisateurs finaux, mais aussi pour offrir des fonctions utiles aux fabricants d'appareils. Cependant, pour les consommateurs, l'utilisation d'un nuage IdO comporte des risques sur le plan de la sécurité et de la confidentialité comme aussi des problèmes d'extensibilité, et accroissent les défis d'ingénierie pour les fabricants d'appareils.

C'était un choix naturel pour de nombreux fabricants d'appareils de s'adresser à divers fournisseurs de solutions IdO complètes qui proposent leur propre plateforme sur le nuage, toute prête à l'emploi disposant des applications correspondantes pour les appareils mobiles. Toutefois, les fabricants d'appareils n'ont pas pu se différencier suffisamment et ont perdu le contrôle nécessaire sur les données générées dans le nuage. D'autre part, certains fabricants d'appareils auraient eu la possibilité de construire leur propre nuage à partir de zéro, mais cela représentait un effort trop important, y compris pour en assurer la maintenance.

Chez Espressif, nous avons compris ce dilemme et décidé de proposer une solution qui reprend le meilleur des deux mondes. D'une part, nous voulions fournir des fonctionnalités complètes pour le nuage, et d'autre part, offrir aux clients la propriété des données, un contrôle maximal et une personnalisation optimale, afin qu'ils n'aient pas à repartir de zéro. C'est ainsi qu'est né ESP RainMaker.

Choisir l'architecture du nuage

Le premier choix consistait à déterminer l'architecture du nuage à utiliser (**figure 1**).

Vous avez peut-être déjà entendu dire que le nuage n'est que l'ordinateur de quelqu'un d'autre. Cette boutade est effectivement vraie. Mais l'important, c'est la manière dont l'ordinateur d'autrui est utilisé pour héberger votre application. Avec l'essor des technologies de virtualisation, des serveurs puissants sont utilisés pour héberger plusieurs applications sur le même matériel, complètement indépendantes les unes des autres. Avec les machines virtuelles (pensez à VMWare) vous pouvez exécuter plusieurs instances du système d'exploitation sur le même matériel, tandis qu'avec les technologies de conteneurs (Kubernetes, Docker), vous pouvez avoir des environnements d'exploitation parallèles sur le même matériel. Bien que cela soit formidable, ces deux architectures vous obligent à vous préoccuper des logiciels à exécuter

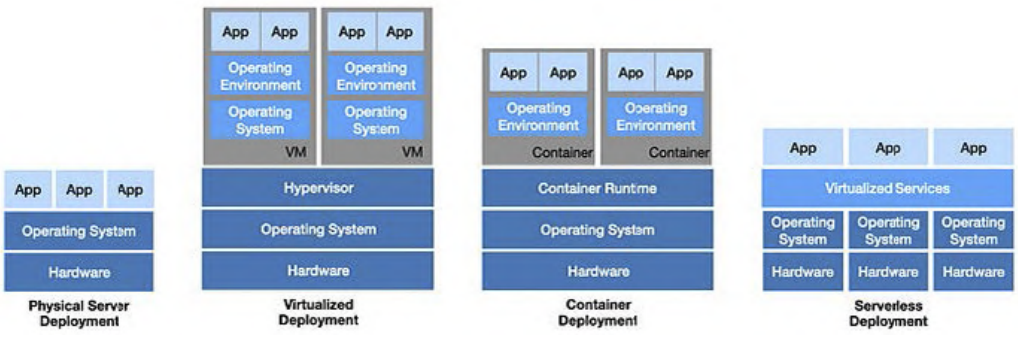


Figure 1. Évolution des architectures en nuage.

et à maintenir, de la manière d'évoluer en fonction de l'augmentation/diminution du nombre d'appareils (c'est le travail d'ingénieurs spécialisés DevOps). Alors que nous voulions que nos clients disposent de leur propre nuage IdO, nous ne pouvions pas les accabler de ces efforts. C'est là qu'un paradigme relativement nouveau d'architecture nuage « sans serveur » nous est venu à l'aide, offrant la promesse de construire un nuage IdO facile à maintenir.

L'architecture sans serveur ne se débarrasse pas des serveurs. Les abstractions sont simplement réalisées à un niveau supérieur. Par exemple, vous obtenez un courtier MQTT (*Message Queuing Telemetry Transport*) sans avoir à vous soucier du logiciel MQTT qu'il utilise, du nombre de connexions d'appareils qu'il peut prendre en charge ou de la plateforme sur laquelle il tourne. Amazon Web Services (AWS) fourni de tels services gérés, qui nous ont permis de mettre en œuvre notre nuage IdO. Parmi les nombreux services gérés, AWS IoT Core fournit un courtier MQTT géré, DynamoDB (Service de base de données NoSQL dans le cloud) fournit une base de données noSQL gérée, S3 (*Amazon Simple Storage Service*, abrégé S3) fournit le stockage et, surtout, Lambda fournit une « fonction en tant que service », où vous pouvez mettre en œuvre la logique sans vous soucier de l'endroit où l'exécuter.

Avec cette mise en œuvre basée sur les services gérés AWS, ESP RainMaker correspond à la combinaison des configurations de divers services et de leurs interactions, et de la mise en œuvre de fonctionnalités par le biais de fonctions Lambda. Il est important de noter qu'il peut être déployé avec n'importe quel compte AWS. Ceci permet facilement de créer une implémentation nuage IdO que les clients peuvent déployer dans leur propre compte AWS et contrôler ainsi entièrement les données et la personnalisation.

Gérer l'extensibilité

Le nuage IdO doit généralement se préoccuper d'un grand nombre d'appareils et d'utilisateurs (par le biais d'applications mobiles ou d'autres clients tels que des assistants vocaux) qui se connectent au nuage et lui envoient des messages. Ce nombre varie en fonction du secteur d'activité et, pour les fabricants d'appareils grand public, peut atteindre des millions d'appareils et d'utilisateurs. Il est donc important de s'assurer que la mise en œuvre de l'informatique en nuage puisse s'adapter à un grand nombre d'utilisateurs.

AWS gérant les différents services, il est logique de supposer que vous n'avez pas à vous soucier de l'extensibilité du backend (serveur) du nuage. Si cela est en grande partie vrai, il n'en reste pas moins que chaque service peut avoir certaines limites. Par exemple, le nombre maximum de fonctions lambda (= fonction anonyme, sans nom) s'exécutant simultanément dans le compte AWS, est limité. Cela nécessite une réflexion approfondie sur l'architecture pour s'assurer que le système est conçu de manière à gérer un grand nombre de messages en utilisant une file d'attente appropriée. Cela signifie également que le micrologiciel de l'appareil et les applications mobiles doivent gérer correctement les erreurs.

Aujourd'hui, ESP RainMaker est testé par plus de 3,5 millions d'appareils et d'utilisateurs qui se connectent et communiquent via le nuage (**figure 2**).

Gestion des coûts opérationnels

Dans une architecture nuage traditionnelle, vous payez généralement pour le calcul, le stockage et la mémoire de vos machines virtuelles. Dans l'architecture sans serveur, l'unité de facturation change. Vous devez payer pour les ressources réellement utilisées, telles que le nombre d'appareils connectés et le temps de connexion total, le nombre de messages MQTT, le nombre de lectures/écritures dans la base de données, la quantité de données stockées dans S3, ainsi que les ressources consommées par les fonctions lambda exécutées. Bien qu'il s'agisse d'un modèle de tarification à l'usage, l'utilisateur du service payant ce qu'il consomme, il se peut que les coûts soient bien plus élevés si votre architecture et votre implémentation ne sont pas optimisées.

Le coût d'exploitation pour les clients était l'un des critères clés lors de la conception de l'ESP RainMaker. L'objectif était facile à définir. Nous voulions développer un backend nuage qui fonctionnerait même pour une ampoule intelligente ; certainement le dispositif IdO le moins cher. Il a fallu beaucoup d'effort pour choisir avec soins les services AWS afin de s'assurer qu'ESP RainMaker puisse atteindre cet objectif. De plus, l'implémentation a été considérablement ajustée pour atteindre un nombre optimal de lectures/écritures dans la base de données, le choix du langage d'implémentation pour utiliser un minimum de ressources pour l'exécution, et ainsi de suite.

ESP RainMaker a actuellement plusieurs clients qui vendent des ampoules à bas prix répondant aux exigences en matière de coûts opérationnels.

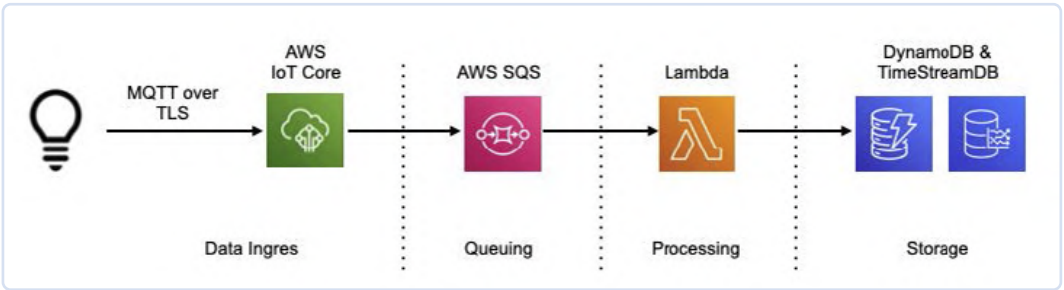


Figure 2. Exemple de traitement des messages de l'appareil dans ESP Rainmaker.

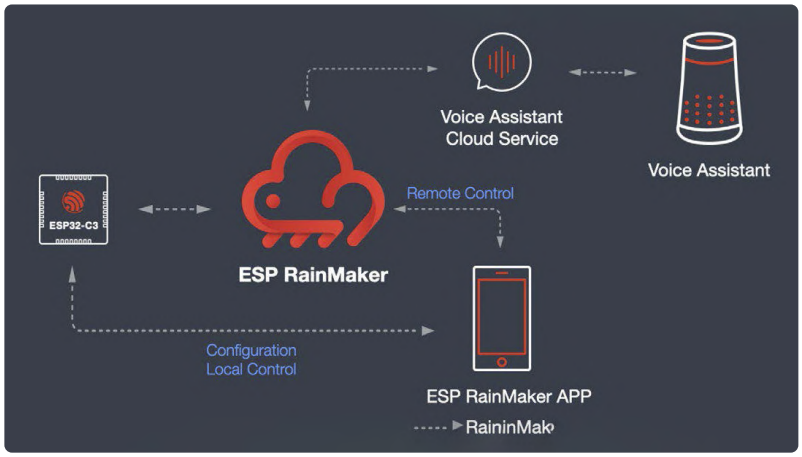


Figure 3. Composants de l'ESP RainMaker.

Sécurité des données et protection de la vie privée

L'utilisation des services gérés par AWS a également permis de garantir la sécurité des données. Ils offrent des méthodes standard d'authentification, d'autorisation, de stockage de données cryptées, de pare-feu applicatifs, etc. Par exemple, le courtier AWS IoT Core MQTT fournit une authentification des appareils basée sur un certificat X.509. Ce qui permet à la fois à un appareil d'authentifier le nuage et au nuage d'authentifier l'appareil. Les polices IAM permettent un contrôle précis des autorisations. Le pare-feu des applications web WAF offre une protection contre les attaques DoS et DDoS sur les points d'extrémité du service RESTful.

ESP RainMaker utilise toutes ces méthodes de sécurité de la manière prescrite, en garantissant une authentification et une autorisation appropriées par des méthodes standards, et il n'y a pas d'escalade de privilèges inutiles. Nous avons également travaillé avec une société tierce de test et de certification pour nous assurer qu'ESP RainMaker met en œuvre toutes les fonctionnalités requises pour répondre aux exigences en matière de confidentialité des données, afin de permettre aux clients de se conformer facilement à la GDPR lorsqu'ils déploient ESP RainMaker pour leur propre compte.

Le nuage IdO ne suffit pas à lui seul

Bien que le backend en nuage constitue une partie importante de la solution IdO complète, il n'est pas suffisant. Du point de vue de l'appareil IdO connecté, en plus du nuage auquel il se connecte, il est tout aussi important de disposer d'un micrologiciel pour l'appareil, d'applications pour smartphone mobile permettant de configurer et d'effectuer des contrôles, et d'intégrer des assistants vocaux, qui dans la maison intelligente sont devenus le moyen le plus naturel pour les utilisateurs de communiquer avec les appareils connectés (**figure 3**).

Chez Espressif, nous avons créé tous ces composants. Le SDK (Software Development Kit, en français KDL = Kit de Développement Logiciel) d'ESP RainMaker pour l'appareil est entièrement open source et facile à intégrer grâce aux nombreux exemples qu'il propose. Les applications de référence iOS et Android sont également entièrement open source. ESP RainMaker comprend les instructions d'Alexa et de Google Assistant. En outre, le nuage n'est guère utile s'il ne peut pas être utilisé pour améliorer le produit grâce à des informations techniques et commerciales. Pour cela, nous avons construit un module de surveillance à distance des appareils, ESP-Insights, qui récolte un journal des événements survenus, des analyses de crashs, un suivi des métriques, et surtout, facilite la création de requêtes étoffées sur les données. Ce module est disponible via le tableau de bord web d'ESP RainMaker, qui permet également la mise à jour à distance (OTA - *Over-The-Air update*), le regroupement d'appareils, le contrôle d'accès basé sur les rôles et les informations commerciales.

Préparer l'avenir

Le protocole Matter apporte la normalisation tant attendue pour les appareils domestiques intelligents. Matter est un protocole de réseau local. La connectivité au nuage peut donc être considérée comme orthogonale avec Matter. Néanmoins, le cloud joue un rôle dans la gestion des appareils et l'accès à distance à ces derniers. ESP RainMaker offre un support Matter Fabric qui peut être utilisé pour créer votre propre écosystème Matter. Une combinaison de backend ESP RainMaker et d'applications téléphoniques mettent en œuvre la structure Matter avec l'infrastructure PKI complète requise pour la structure Matter. Les bases de données des appareils, des utilisateurs et des ACL sont synchronisées en toute sécurité entre plusieurs applications. De plus, lorsque vous avez un contrôleur de maison intelligente qui se connecte à RainMaker, vous pouvez fournir un contrôle à distance à partir de vos applications mobiles, non seulement à vos propres appareils, mais aussi à d'autres appareils Matter.

Avec ESP RainMaker, nous avons également annoncé la prise en charge de l'intégration de la topologie réseau Mesh-Lite. Avec Mesh-Lite vous pouvez avoir un réseau Wifi maillé dédié à vos appareils IdO de plus grande portée et couvrant également des zones qui seraient à considérer comme mortes dans la maison. Les appareils IdO basés sur la puce ESP32 agissent comme des points d'accès pour permettre la connexion à d'autres appareils, formant ainsi une topologie maillée (proche d'une arborescence). Avec ESP RainMaker, les utilisateurs peuvent facilement créer le réseau maillé et approvisionner les nœuds à l'intérieur de celui-ci.

Pour commencer

Nous avons construit le nuage ESP RainMaker en tenant compte de considérations que tout fabricant d'appareils devrait également respecter lors de la mise en œuvre de sa propre plateforme IdO en nuage. Plus important encore, Espressif fournit une plateforme entièrement personnalisable. Pour les développeurs, Espressif fournit également une instance ESP RainMaker gérée par Espressif ainsi que des applications téléphoniques disponibles dans les magasins d'applications respectifs. Vous pouvez utiliser le tableau de bord web pour la gestion des appareils et effectuer des mises à jour OTA. Ainsi, vous pouvez non seulement évaluer les fonctionnalités, mais aussi créer vos projets pour votre usage personnel.

Lorsque vous commencer, vous pouvez utiliser n'importe quelle carte de développement Espressif et suivre les étapes mentionnées dans le guide « Get Started » [1].

VF : Jean-Philippe Nicolet — 230621-04

À propos de l'auteur

Amey Indamar est le directeur du marketing technique chez Espressif. Il a 20 ans d'expérience dans le domaine des systèmes embarqués et des appareils connectés, ayant occupé des postes dans l'ingénierie, la gestion de produits et le marketing technique. Il a travaillé avec de nombreux clients pour construire des appareils connectés basés sur la connectivité Wifi et Bluetooth.

LIEN

[1] Démarrer avec ESP RainMaker : https://rainmaker.espressif.com/docs/get-started

assemblage du kit CLOC 2.0 D'ELEKTOR

un produit Elektor déballé par Espressif

Jeroen Domburg (Espressif)

Le Cloc 2.0 d'Elektor est une horloge flexible avec alarme à base d'un ESP32-Pico-Kit d'Espressif. Il se présente sous la forme d'un kit de pièces à assembler soi-même. Dans cet article, nos amis d'Espressif en ont assemblé un et nous on fait part de leurs impressions.

Comment vous réveillez-vous le matin ? Avez-vous une alarme programmée sur votre téléphone, avez-vous encore un vieux radio-réveil à côté de votre lit, êtes-vous réveillé par le coq le matin ou laissez-vous simplement les rideaux ouverts pour que les premiers rayons du soleil atteignent votre visage ?
Vous pensez peut-être que toutes ces méthodes présentent des inconvénients. Votre téléphone ne vous permet pas vraiment d'ouvrir un œil pour voir combien de temps de sommeil il vous reste, et le radio-réveil n'a probablement qu'une seule alarme, qui vous réveillera également le matin pendant un week-end ou un jour férié si vous ne l'éteignez pas. Et bien que le coq et les rayons de soleil soient bon marché et généralement fiables, il est assez difficile de les configurer pour une alarme. D'une manière générale, aucune de ces solutions n'est « véritablement » flexible.
Si vous souhaitez tout de même bénéficier de cette souplesse, l'option la plus simple est évidemment de créer votre propre réveil. Si vous le construisez vous-même, vous pouvez y intégrer toutes les fonctionnalités que vous souhaitez. Les horaires d'alarme hebdomadaires, la connectivité Wifi, les sons d'alarme amusants, l'augmentation de la température ambiante, la mise en marche de la machine à café, tout ce que vous souhaitez..

Figure 1. *Le kit se présente sous la forme d'un ensemble de sachets soigneusement étiquetés, ainsi que du boîtier de la société Hammond.*

▼

Découvrez Cloc 2.0

Mais que faire si vous n'avez pas les compétences, le temps ou la motivation de passer par tout le chemin de l'architecture, de la conception matérielle et logicielle, de la fabrication, etc. Elektor peut vous aider avec le Cloc 2.0. Ce modèle est doté d'un double affichage LED à 7 segments pour visualiser l'heure actuelle et l'alarme, d'une connexion Wifi, afin de toujours afficher l'heure exacte et de permettre de nombreuses options configurables. De plus, elle dispose d'un émetteur et d'un récepteur de télécommande IR qui vous permet d'allumer, par exemple, un système de sonorisation lorsque l'heure de l'alarme à sonné. Mieux encore, elle est basée sur un ESP32-Pico-Kit et le logiciel est open-source, ce qui signifie qu'il est facile d'ajouter une fonction s'il en manque une. Elektor a eu la gentillesse de nous envoyer un kit pour que nous puissions l'évaluer. Le projet complet de la Cloc 2.0 est publié sur le site d'Elektor Labs ici [1].

Kit

Le Cloc 2.0 nous a été livré en kit, et donc en pièces détachées (**figure 1**). Le plus étonnant était le boîtier rouge de la société Hammond, sur lequel était imprimé le logo d'Elektor Labs. Outre l'aspect esthétique, la combinaison d'un boîtier rouge et d'un afficheur à 7 segments à LED rouge est intéressante car elle améliore le contraste des LED, ce qui rend l'horloge plus facile à lire. Le reste des composants se trouvait dans plusieurs sachets, étiquetés avec les valeurs des

composants se trouvant à l'intérieur. Chose intéressante, chaque sachet contient un ensemble de composants différents et reconnaissables. Par exemple, aucun sachet ne contenait plus de deux valeurs de résistance, et il n'est donc pas nécessaire de regarder les codes de couleurs pour savoir de quelle résistance il s'agit. C'est une idée intelligente qui permet d'économiser du plastique tout en permettant simplement de trouver le bon composant. Bravo à l'équipe d'Elektor pour avoir imaginé ce système ! À noter que le kit ne comporte pas de boutons accessibles depuis l'extérieur du boîtier, l'idée étant que vous en avez peut-être encore quelques-uns à votre disposition quelque part.

Trouver la documentation

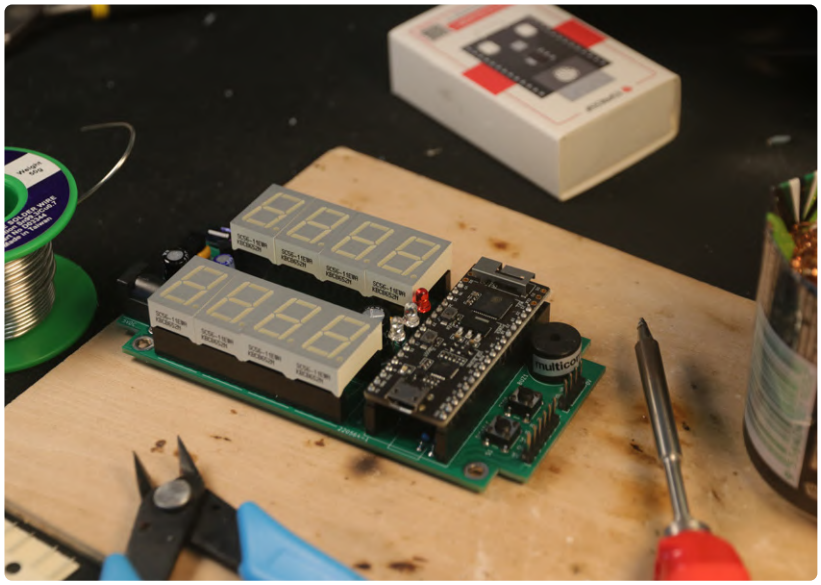
La recherche de la documentation nécessaire à la réalisation de ce projet aurait pu être un peu plus aisée. Le site d'Elektor comporte des pages web réparties sur les trois domaines (*elektorlabs.fr*, *elektor.fr* et *elektor-magazine.fr*) avec des téléchargements et des textes différents sur chaque page. C'est un peu déroutant, et il aurait été utile de trouver au moins un document imprimé détaillé, pour avoir tous les téléchargements et la documentation nécessaires concernant le kit. Cependant, une fois que nous avons tout trouvé, il s'est avéré très complet : l'article d'Elektor qui décrit le projet est inclus ainsi que tous les téléchargements, et même une vidéo [2] sur la meilleure façon d'assembler le tout.

Avec la documentation en main, le montage du circuit imprimé est un jeu d'enfant (**figure 2**). Tous les composants sont traversants, ce qui facilite le soudage. La vidéo propose un ordre de soudage des composants qui a très bien fonctionné pour nous.

Il faut remarquer que les connecteurs qui supportent les afficheurs LED à 7 segments sont du même type que ceux dans lesquels l'ESP32-Pico-Kit est inséré. C'est un peu un problème, car les broches des afficheurs LED sont plus fines et ne font pas toujours bon contact. Ceci est compréhensible, car il ne semble pas y avoir de connecteurs pour des broches plus petites disponibles à la même hauteur que celles-ci. Et de plus, tout ce qui est problématique ici est facilement résolu en pliant un peu les broches des afficheurs.

Programmation de l'ESP32-Pico-Kit

Une fois que la carte est assemblée, il fallait programmer l'ESP32-Pico-Kit. Il a fallu quelques étapes pour installer les bonnes bibliothèques et le support de la carte, mais cela n'a pas été trop compliqué non plus. Un tel processus aurait pu être simplifié en utilisant par exemple ESP-Launchpad [3], mais la configuration actuelle a l'avantage de construire le croquis à partir des sources. Cela signifie que si vous trouvez quelque chose que vous voulez changer dans le code, vous avez déjà tout ce qui est nécessaire pour reconstruire et flasher le micrologiciel.



▲

Figure 2. *Le circuit imprimé étant entièrement traversant, le soudage a été un jeu d'enfant.*

Construction mécanique

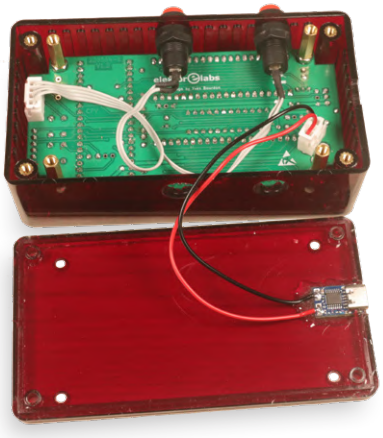
En ce qui concerne le montage du circuit imprimé dans le boîtier, il y a un peu plus de flexibilité dans la façon dont vous voulez faire les choses. Elektor fournit un gabarit de perçage et quelques éléments de quincaillerie dans les kits récents, de sorte que vous pouvez le monter de la manière « standard ». Peut-être que vous souhaitez ou même devez modifier certaines choses afin de les installer selon vos souhaits. Par exemple, en fonction de la forme et de la taille des boutons que vous souhaitez utiliser, vous pouvez créer votre propre gabarit de perçage ici.

Alimentation

Dans notre cas, bien que nous soyons sûrs d'en avoir des dizaines quelque part, nous n'avons pas réussi à trouver un bloc de 5 V avec le bon connecteur jack dans la grande caisse contenant les diverses alimentations. Ce que nous avons trouvé, c'est un petit circuit imprimé avec un connecteur USB-C, que nous avons pu modifier pour qu'il délivre simplement 5 V avec les 500 mA ou plus qui sont nécessaires. Cette solution offre l'avantage supplémentaire de ne pas avoir à utiliser une alimentation dédiée pour notre Cloc. N'importe quelle alimentation USB-C et n'importe quel câble feront l'affaire. Évidemment, nous avons modifié les plans de perçage du boîtier en conséquence. Plutôt que de percer un trou pour le connecteur jack, nous avons découpé un peu de plastique à l'arrière du boîtier pour le connecteur USB-C. Le circuit imprimé USB peut alors être fixé à l'arrière du boîtier avec de la résine époxy pour le maintenir en place.

Figure 3. *L'arrière du boîtier contient suffisamment d'espace. Les boutons et les câbles y trouvent leur place, mais vous pouvez également y glisser un circuit imprimé d'extension.*

▼



Percez avec soin

À ce propos, voici un bon conseil. Lorsque vous percez des trous et installez des éléments, veillez à garder à l'esprit l'ordre des choses et l'orientation du boîtier. Évidemment, en tant que professionnels

Figure 4. *Notre horloge Cloc terminée, alimentée par un adaptateur USB-C.*

chez Espressif, nous en sommes parfaitement conscients et nous pouvons donc vous assurer que… les trous de drainage… dans le fond de notre boîtier faisaient partie de nos plans initiaux depuis le début (**figure 3**).

Pour finir, nous avons tout assemblé proprement. Il nous restait quelques jolis boutons rouges assortis avec le boîtier que nous pouvions mettre en place. Plutôt que d'utiliser les connecteurs fournis, nous avons utilisé des connecteurs JST pour les connecter au circuit imprimé. Nous avons également utilisé un connecteur JST pour connecter la carte d'alimentation USB. De cette façon, nous pouvons sortir le circuit imprimé principal du boîtier au cas où il aurait besoin d'être modifié ou réparé.

Configuration de l'horloge Cloc

Une fois que tout est assemblé, la configuration de l'horloge Cloc s'est avéré très simple. Mettez-la sous tension, connectez-vous et allez à l'adresse IP indiquée, et vous pourrez la connecter au réseau Wifi local. A partir de là, chaque fois que le Cloc démarre, elle vous indiquera son adresse IP, ce qui est très bien car vous n'avez pas besoin de vous embêter avec des scanners réseau ou de regarder la configuration du routeur pour accéder à l'interface web. L'interface web elle-même est suffisamment simple pour ne pas avoir besoin d'un manuel, mais elle est très complète. Nous avons configuré les éléments de base comme le fuseau horaire et l'heure d'été en quelques instants seulement.

Dans son ensemble, nous aimons beaucoup l'horloge (**figure 4**). L'affichage LED est agréable et lisible, et le fait qu'il soit rouge signifie qu'il ne détruira pas votre vision nocturne si vous le regardez dans l'obscurité. L'horloge est très facilement configurable à l'aide de l'interface web et si vous souhaitez la modifier pour vous réveiller comme vous le souhaitez, il y a moyen de le faire. À la fois au sens figuré avec l'accès facile grâce à l'utilisation de l'environnement Arduino, et au sens physique puisque le boîtier de l'horloge a de la place pour un circuit supplémentaire, par exemple derrière celui de le Cloc.

Suggestions pour une Cloc 3.0

Il y a quelques améliorations que nous aimerions voir pour une horloge Cloc V3, comme l'utilisation d'un connecteur USB comme alimentation (petite astuce : si le design utilisait par exemple un ESP32-S3, les broches d'E/S USB fournies pourraient être connectées au connecteur USB également, permettant la programmation et le débogage de l'horloge Cloc sans ouvrir le boîtier). De plus, l'ESP32 pourrait peut-être piloter un haut-parleur pour un son de réveil un peu moins rude que le signal sonore actuel. Nous connaissons l'émetteur IR qui peut allumer, par exemple, une radio, mais nous pensons que de moins en moins de gens en possèdent dans leur chambre à coucher de nos jours. D'un autre côté, si ce sont des choses dont vous avez vraiment, mais alors vraiment besoin, rien ne vous empêche de les intégrer dans la version actuelle du Cloc, étant donné sa conception et son logiciel ouverts.

VF : Laurent Rauber — 230561-04

Questions ou commentaires ?

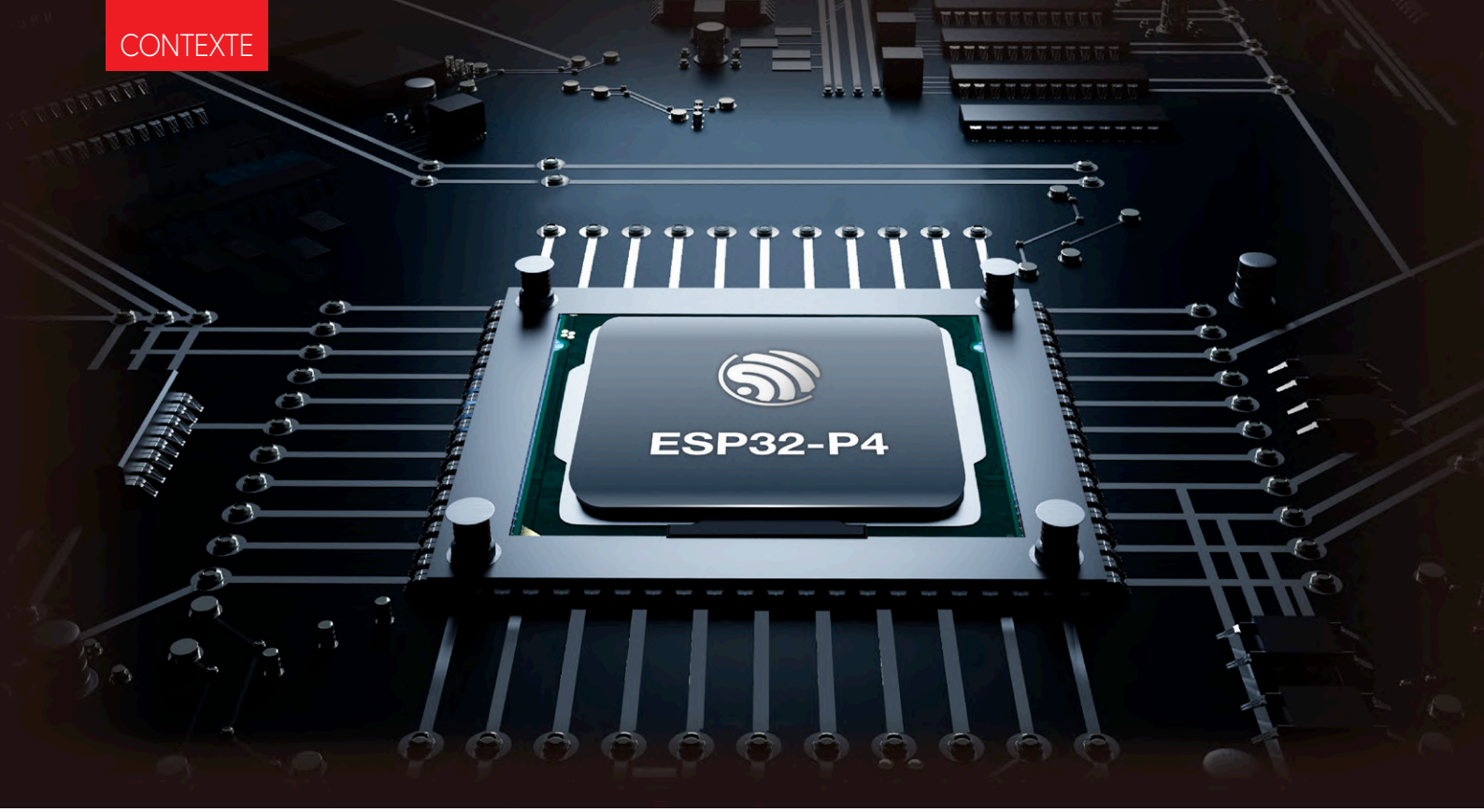
Contactez Elektor (redaction@elektor.fr).

Produits

- **Cloc 2.0 d'Elektor en kit**
  www.elektor.fr/20438
- **ESP32-PICO-Kit V4**
  www.elektor.fr/18423

LIENS

[1] Le projet Cloc 2.0 sur Elektor Labs : https://elektormagazine.fr/labs/cloc-le-reveil-20
[2] Horloge « maison » avec un ESP32 - guide d'assemblage du Cloc 2.0 d'Elektor : https://youtu.be/9VSLdFz6jyI
[3] ESP-Launchpad : https://espressif.github.io/esp-launchpad



le lancement de l'ESP32-P4

la nouvelle ère des microcontrôleurs

Anant Raj Gupta, Espressif

Bienvenue dans cette nouvelle ère des microcontrôleurs, où se combinent sécurité abordable, performances de pointe et connectivité inégalée. Le microcontrôleur hautes performances promet de réinventer le monde des systèmes embarqués, ouvrant la voie à de nouvelles perspectives pour les développeurs, les ingénieurs et l'ensemble de la communauté de l'Internet des Objets.

Au cœur de ce saut technologique se trouve l'ESP32-P4, un SoC (système sur une puce) soigneusement conçu pour des performances élevées et renforcé par des fonctions de sécurité de premier plan. Ce puissant microcontrôleur est doté d'une mémoire interne étendue, d'impressionnantes capacités de traitement de l'image et du son, et possède des périphériques à grande vitesse. Tous ces éléments sont inclus (**figure 1**) pour offrir la capacité de répondre aux exigences de la nouvelle ère d'applications embarquées.

Le processus de calcul de l'ESP32-P4

Unité centrale et sous-système de mémoire haute performance

Les performances de l'ESP32-P4 sont dopées par un processeur RISC-V double cœur cadencé à 400 MHz. Cette unité centrale est également équipée d'unités à virgule flottante (FPU) à simple précision et d'extensions d'intelligence artificielle, offrant un arsenal de ressources de calcul. L'intégration d'un cœur à basse consommation (LP-Core), capable de fonctionner jusqu'à 40 MHz, vient compléter ce dispositif. Cette architecture « grand-petit » est essentielle pour les applications à très faible consommation qui exigent des pics occasionnels de puissance de calcul.

Dans la quête des performances exceptionnelles, l'ESP32-P4 se distingue, comme l'illustre la **figure 2** en comparant ses performances à celles d'autres produits Espressif tels que l'ESP32 et l'ESP32-S3.

Architecture de la mémoire pour une efficacité inégalée

L'accès à la mémoire est un facteur essentiel de garantie de performances sans faille. Le système de mémoire de l'ESP32-P4 est un maître d'efficacité. Avec 768 Ko de SRAM sur la puce, reconfigurables en cache en cas de disponibilité de PSRAM externe, et 8 Ko de RAM TCM sans délai d'attente pour une mise en mémoire tampon rapide des données, ce SoC garantit que la latence et la capacité d'accès à la mémoire ne sont jamais des goulots d'étranglement pour vos applications.

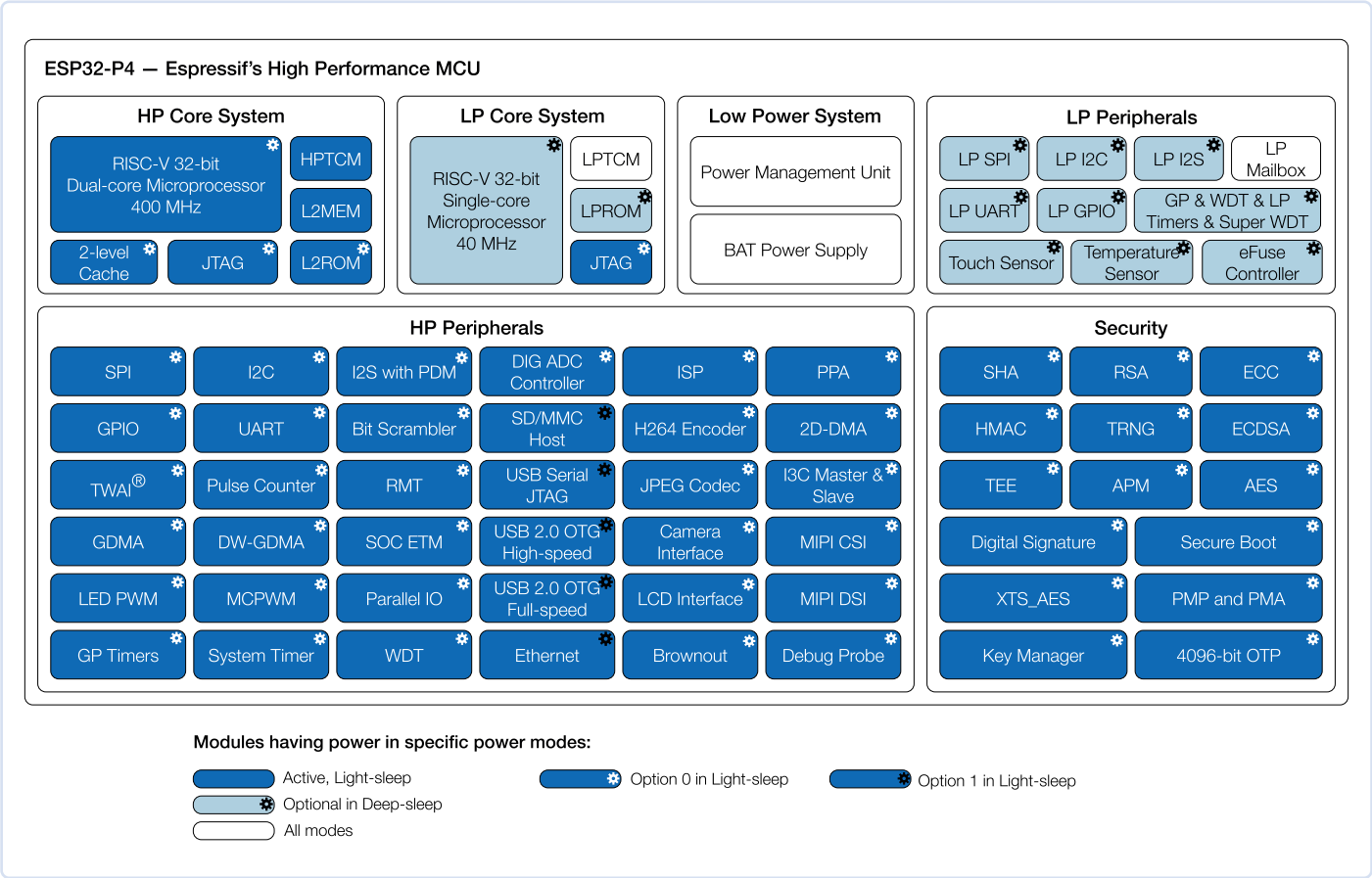


Figure 1. Schéma fonctionnel de l'ESP32-P4.

De plus, la mémoire externe de l'ESP32-P4 est directement accessible, offrant un espace contigu de 64 Mo pour l'accès aux mémoires flash et PSRAM via le cache. La mémoire interne de 768 Ko peut être configurée comme L2MEM, partitionnée pour les instructions et les données, et est complétée par un cache d'instructions L1 dédié de 16 Ko et un cache de données L1 de 64 Ko. Une mémoire supplémentaire de 8 Ko étroitement couplée au cœur HP assure un accès à cycle unique aux données stockées. La PSRAM SPI offre la possibilité de lire et d'écrire sur seize lignes DDR, tout en améliorant la prise en charge du fonctionnement à une vitesse d'horloge de 250 MHz, produisant un débit de données maximal de 1 Go/s. Cette architecture de mémoire à plusieurs niveaux est conçue pour rendre l'accès à la mémoire presque transparent aux applications, ce qui améliore substantiellement leurs performances.

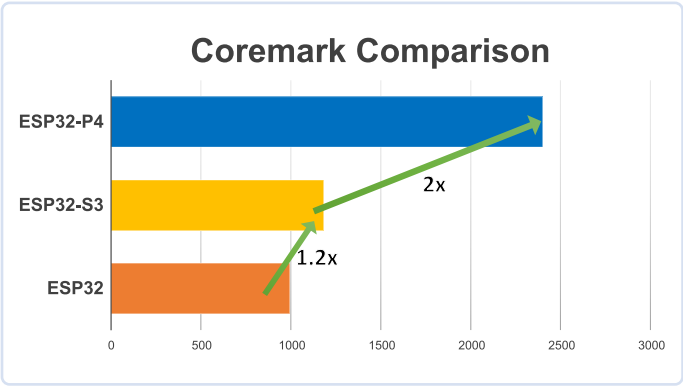


Figure 2. Comparaison des critères de référence.

Traitement personnalisé pour les charges de travail d'IA et de DSP

L'ESP32-P4 est doté d'un processeur RISC-V 32 bits à double cœur qui prend en charge les extensions RV32IMAFCZc standard. Cependant, ce qui le distingue vraiment, c'est son jeu d'instructions étendu et personnalisé. Ce jeu est conçu pour réduire le nombre d'instructions dans les corps de boucle, ce qui améliore considérablement les performances. De plus, le SoC incorpore des extensions IA et de traitement de signal (DSP) personnalisées, optimisant l'efficacité opérationnelle d'algorithmes IA et DSP spécifiques. Ces instructions vectorielles personnalisées permettent à l'ESP32-P4 de s'atteler à des tâches telles que les réseaux neuronaux et l'apprentissage profond, permettant ainsi un calcul accéléré et efficace.

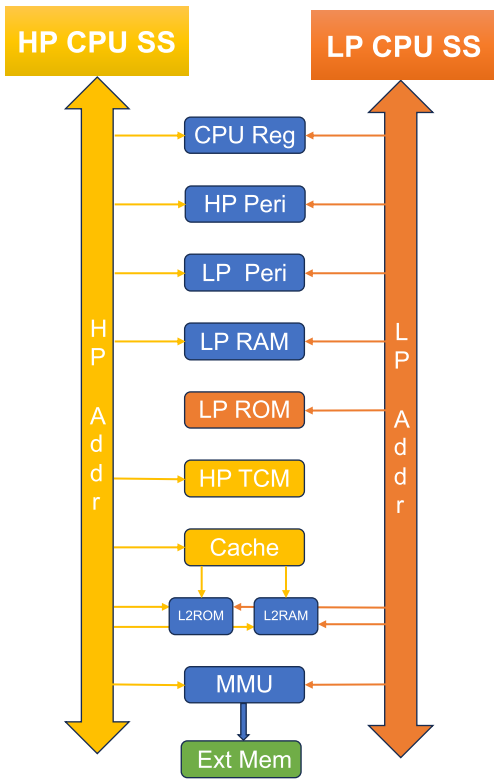


Figure 3. Accès à l'espace d'adressage de l'ESP32-P4.

Accès efficace à la mémoire pour chaque cœur

Comme le montre la **figure 3**, l'architecture d'accès à la mémoire de l'ESP32-P4 garantit que tous les périphériques et la mémoire sont accessibles à partir des cœurs HP et LP.

Cette configuration permet au cœur LP de gérer la plupart des fonctions et de ne réveiller le cœur HP que lorsqu'un calcul de haute performance est nécessaire. En outre, les périphériques du cœur LP permettent d'exécuter des fonctions critiques même dans les modes de consommation les plus faibles. Grâce à ces caractéristiques, l'ESP32-P4 apparaît comme le choix idéal pour les applications nécessitant des capacités de calcul de pointe tout en maintenant une faible consommation d'énergie sur de longues périodes.

Une sécurité de premier ordre

Démarrage sécurisé

La sécurité n'a pas été ajoutée après coup, elle a été à la base du développement de l'ESP32-P4. Elle a été soigneusement conçue pour mettre des solutions de sécurité à la portée de tous. Le démarrage sécurisé, une fonction de sécurité essentielle, protège l'appareil contre l'exécution de codes non autorisés et non signés. Il vérifie minutieusement chaque logiciel, y compris le chargeur de démarrage de deuxième niveau et chaque binaire d'application. Comme l'indique la **figure 4**, le chargeur de premier niveau, implanté en ROM, est inaltérable et n'a pas besoin d'être signé. L'ESP32-P4 offre une grande souplesse en prenant en charge les systèmes de vérification du démarrage sécurisé basés sur RSA-PSS et ECDSA, ECDSA offrant une sécurité comparable à celle de RSA, mais avec des longueurs de clé plus courtes.

Chiffrement de la mémoire Flash

L'ESP32-P4 innove avec le chiffrement de la mémoire flash externe. Lorsque cette fonction est activée, le micrologiciel est initialement flashé en clair, puis chiffré lors du premier démarrage. Cela signifie que les tentatives physiques de lecture de la mémoire flash ne permettront pas de récupérer des données significatives. Dans ce mode de fonctionnement, tous les accès en lecture à la mémoire flash sont déchiffrés de manière transparente et sécurisée à l'exécution. L'ESP32-P4 utilise le mode de chiffrement par bloc XTS-AES avec une clé robuste de 256 bits pour le chiffrement de la mémoire flash, ce qui garantit une protection de premier ordre des données.

Accélération matérielle de la cryptographie

L'ESP32-P4 est équipé d'une suite complète d'accélérateurs cryptographiques matériels, prêts à utiliser une série d'algorithmes standard dans les applications de réseau et de sécurité, notamment AES-128/256, SHA, RSA, ECC et HMAC, améliorant la capacité de l'ESP32-P4 à sécuriser la transmission, le stockage et l'authentification des données, ce qui en fait un choix exceptionnel pour les applications sensibles à la sécurité.

Protection des clés privées

La protection des clés privées est primordiale, et l'ESP32-P4 utilise

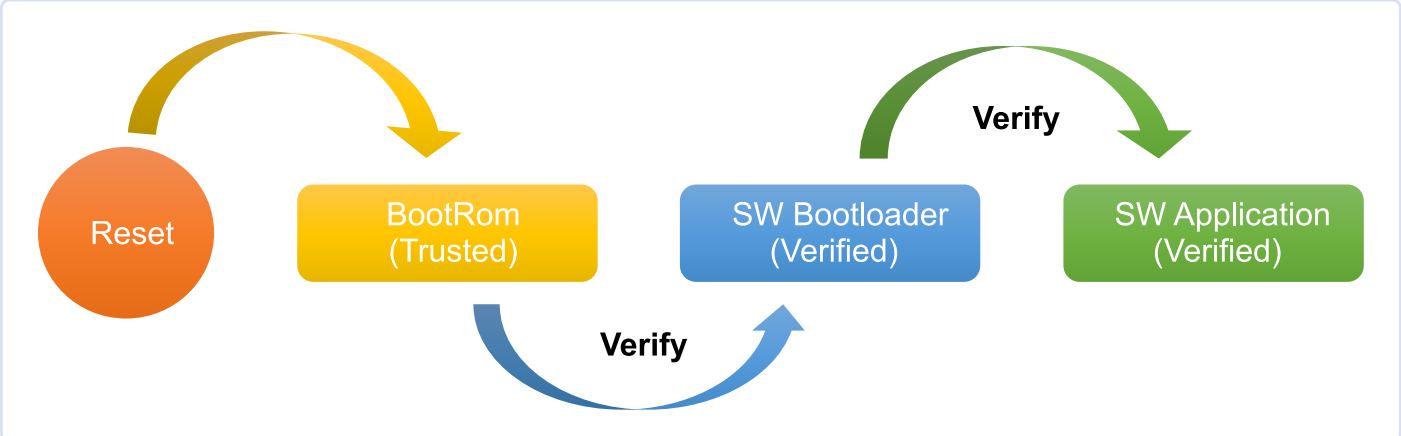


Figure 4. Processus de démarrage sécurisé.

des mécanismes robustes pour garantir leur sécurité. Le SoC génère des clés privées sur la puce, inaccessibles aux attaques logicielles ou physiques en texte clair.
Grâce à cette production de clés interne accélérée par le matériel, les applications peuvent effectuer des opérations de signature avec une clé privée chiffrée jamais exposée en clair. L'ESP32-P4 exploite un gestionnaire de clés qui utilise des fonctions non physiquement clonables (PUF) propres à chaque puce pour générer une clé matérielle unique (HUK), qui sert de racine de confiance (*Root of Trust*, RoT) pour la puce et est automatiquement générée à chaque mise sous tension, disparaissant lorsque la puce est éteinte. La protection matérielle de la clé privée de l'ESP32-P4 garantit une sécurité inégalée pour les opérations sensibles.

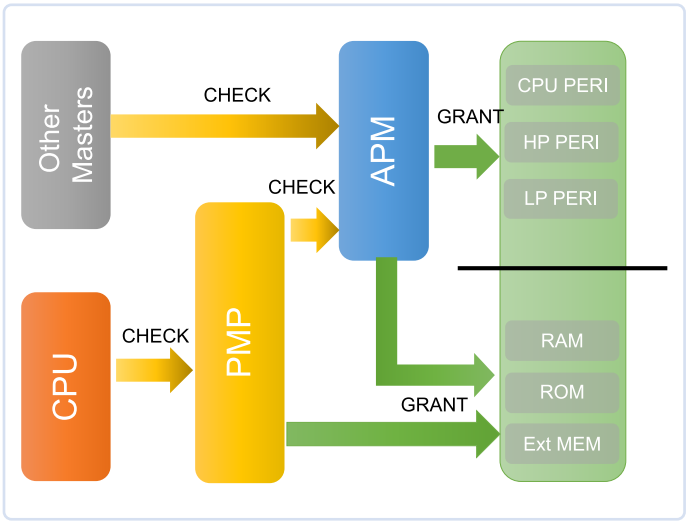


Figure 5. Contrôle d'accès sur l'ESP32-P4.

Contrôle d'accès

L'ESP32-P4 porte le contrôle d'accès à un niveau supérieur grâce à la protection matérielle de l'accès, qui facilite la gestion des autorisations d'accès et la séparation des privilèges (**figure 5**). Ce système se compose de deux éléments : la protection de la mémoire physique (PMP) et la gestion des autorisations d'accès (APM). La PMP gère l'accès du CPU à tous les espaces d'adressage, tandis que l'APM s'occupe de l'accès à la ROM et à la SRAM. La PMP doit autoriser l'accès du CPU à la ROM et à la SRAM HP avant que l'APM puisse intervenir pour les autres espaces d'adressage. En cas de refus, les contrôles de l'APM ne sont pas effectués. Cette approche par couches garantit un contrôle d'accès solide, faisant de l'ESP32-P4 le bon choix pour une large gamme d'applications.

Périphériques divers et interface homme-machine (IHM)

Une expérience visuelle et tactile améliorée

L'ESP32-P4 améliore l'expérience IHM grâce à sa prise en charge des interfaces MIPI-CSI (Interface Série pour Caméras) et MIPI-DSI (Interface Série pour Affichage). Intégrée au traitement du signal d'image (ISP) sur l'interface MIPI-CSI, cette fonction permet au SoC de gérer les principaux formats d'entrée, tels que RAW8, RAW10 et RAW12, avec des résolutions allant jusqu'au Full HD (1980×1080) à 30 images par seconde (ips). Cette capacité le rend idéal pour des applications telles que les caméras IP (**figure 6**) et les sonnettes de porte vidéo qui exigent des entrées de caméra à haute résolution. Côté affichage, l'interface MIPI-DSI de l'ESP32-P4 prend en charge deux voies à 1,5 Gbps, ce qui se traduit par une prise en charge de l'affichage en HD 720 p à 60 ips ou en Full HD 1080 p à 30 ips. L'intégration d'entrées tactiles capacitives et de fonctions de reconnaissance vocale fait de l'ESP32-P4 une plateforme idéale pour toute application basée sur une IHM, des panneaux de commande interactifs aux contrôleurs de charge.

Traitement des médias

L'ESP32-P4 ne s'arrête pas aux caméras et aux écrans ; elle se prête à des tâches d'encodage et de compression des médias. Grâce à la prise en charge intégrée du format H.264 et d'autres formats d'encodage, l'ESP32-P4 permet la diffusion et le traitement de vidéos. Ces fonctions peuvent être exploitées pour créer des solutions de caméras IP économiques, en tirant parti des divers périphériques IHM mentionnés plus haut. Le SoC dispose également d'un accélérateur matériel de traitement des pixels (PPA) intégré, adapté au développement d'interfaces utilisateur graphiques.

Prise en charge étendue des GPIO et des périphériques

Avec un nombre remarquable de 55 GPIO programmables, l'ESP32-P4 établit un nouveau standard pour les produits Espressif. Il gère divers protocoles d'usage courant, tels que SPI, I2S, I²C, LED PWM, MCPWM, RMT, ADC, DAC, UART et TWAITM. L'ESP32-P4 prend en charge USB OTG 2.0 HS, Ethernet et SDIO Host 3.0 pour une connectivité à grande vitesse.

Connectivité continue sans fil

Pour les applications nécessitant une connectivité sans fil, l'ESP32-P4 s'associe sans difficulté aux produits de la série ESP32-C/S/H en tant que puce compagne sans fil. C'est réalisable via SPI/SDIO/UART à l'aide des solutions ESP-Hosted ou ESP-AT. Avec ESP-Hosted, le développement d'applications reste très proche du travail avec une puce à connectivité sans fil intégrée. De plus, l'ESP32-P4 peut servir de microcontrôleur hôte pour d'autres solutions de connectivité, notamment ACK et AWS IoT ExpressLink. Les développeurs peuvent s'appuyer sur l'environnement de développement IdO mature d'Espressif (ESP-IDF) pour le support, en tirant parti de leur connaissance d'une plateforme qui équipe déjà des millions d'objets connectés.

Figure 6. Exemple de flux de données pour une caméra IP.

Bâtir l'avenir avec l'ESP32-P4

L'ESP32-P4 représente une nouvelle ère dans le domaine des microcontrôleurs, grâce à ses performances, sa sécurité et sa connectivité, qui permettent aux développeurs de donner vie à leurs idées les plus audacieuses. L'ESP32-P4 est une plateforme polyvalente, sûre et performante.

L'avenir de l'IdO est prometteur, et l'ESP32-P4 se place à la pointe de cette évolution, en offrant un traitement plus rapide et plus efficace. L'informatique de pointe rapprochera l'intelligence des sources de données. L'intelligence artificielle et l'apprentissage automatique permettront aux objets de l'IdO de prendre des décisions intelligentes et autonomes. L'ESP32-P4 est prêt à accueillir cette évolution, avec sa puissance de traitement, sa sécurité robuste et sa communauté dynamique de développeurs.

L'ESP32-P4 représente un immense saut dans le monde des microcontrôleurs. Ce n'est pas seulement une puce, c'est une invitation à l'innovation. Que vous soyez un développeur chevronné ou un nouveau venu passionné, l'ESP32-P4 vous permet de réaliser vos idées. Franchissez le pas, explorez le potentiel de l'ESP32-P4 et participez à la mutation que représente ce microcontrôleur. Ensemble, nous pouvons bâtir l'avenir de l'IdO et créer un monde plus connecté, plus efficace et plus sûr.

Vf : Helmut Müller — 230615-04

Questions ou commentaires ?

Contactez Espressif via le code QR ou Elektor à l'adresse redaction@elektor.fr.

À propos de l'auteur

Anant Gupta est un directeur du marketing technique avec plus de 15 ans d'expérience dans l'architecture de système, l'architecture IP et la conception de SoC. Il est passionné par l'innovation et la résolution des problèmes des clients d'Espressif Systems.

Produits

> **Produits ESP32 d'Espressif**
> http://www.elektor.fr/espressif

SDK

ESPRESSIF

les composants les plus intéressants d'ESP-IDF

Ce n'est pas seulement Node.js qui a *npm* ou Python qui a *pip*. ESP-IDF dispose également d'un gestionnaire de composants, où vous pouvez importer dans votre projet des composants fournis par Espressif et des composants tiers.

Les composants consistent en une bibliothèque logicielle accompagnée d'exemples et de documentation qui vous aident à les utiliser dans votre projet. Vous trouverez ici une variété de bibliothèques logicielles pour votre projet, allant de différents pilotes de périphériques, d'intergiciels (par exemple, LVGL porting), à différents paquets de support pour différentes cartes.

Cette liste s'enrichit continuellement au fur et à mesure de l'ajout de composants Espressif et de composants tiers. En tant que développeur open-source, vous pouvez également créer vos propres composants et les enregistrer ici afin que les autres développeurs puissent les trouver facilement.

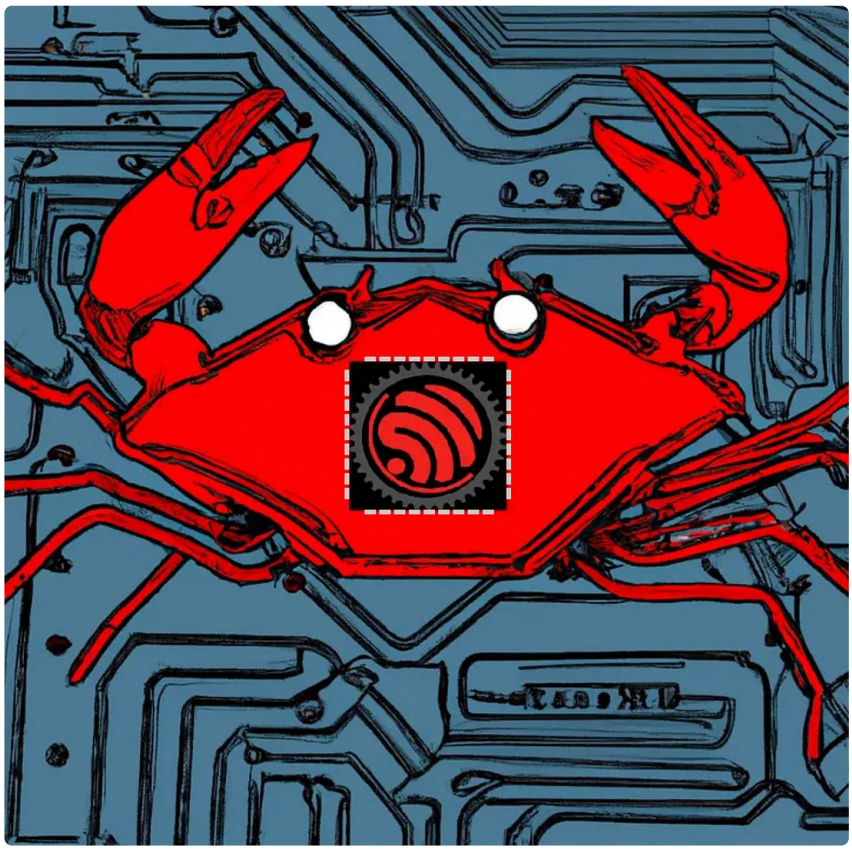
https://components.espressif.com

Rust + les systèmes embarqués

deux outils puissants pour le développement

Juraj Sadel (Espressif)

Principalement axé sur la sécurité de la mémoire et des processus, Rust est un langage populaire permettant la réalisation de logiciels fiables et sécurisés. Rust est-il pour autant une solution adéquate pour les applications embarquées ? Comme vous allez l'apprendre, Rust offre de nombreux avantages, en comparaison avec les langages de développement traditionnels pour les solutions embarquées, telles que C et C++, en particulier protection de la mémoire, support du parallélisme et performances.



Source : Image générée par DALL-E.

Au niveau mondial, Rust est devenu le nouveau langage le plus en vogue, de plus en plus de personnes s'y intéressent chaque année. Il est efficient sans le domaine de la sécurité de la mémoire et des processus concurrentiels, il se présente avec l'intention de produire des logiciels fiables et sécurisés. Le support de la concurrence des processus et du parallélisme est particulièrement remarquable dans le domaine du développement des logiciels embarqués où l'utilisation optimisée des ressources est critique.

Les débuts de Rust

L'idée initiale du langage de programmation Rust est née par accident. En 2006, à Vancouver, M. Graydon Hoare rentrait chez lui, mais en raison d'un bug logiciel, l'ascenseur était à nouveau en panne, M. Hoare vivait au 21ème étage, alors qu'il montait les escaliers, il réfléchissait. « Nous les informaticiens, nous ne sommes même pas capables d'assurer le contrôle d'un ascenseur sans qu'il ne dysfonctionne ! »

Cet incident amena M. Hoare à réfléchir à la conception d'un nouveau langage de programmation. Il souhaitait rendre possible l'écriture d'un programme compact, rapide, sans défauts mémoire. [1] Si vous êtes intéressé par davantage d'information technique détaillée sur l'histoire de Rust, merci de visiter [2] et [3]. Environ 18 ans après, Rust est devenu mondialement le langage le plus en vogue, et de plus en plus de personnes s'y intéressent chaque année. Au premier trimestre 2020, il y avait environ 600 000 développeurs Rust, et au premier trimestre 2022, ce nombre a franchi la barre des 2,2 millions. [4] Des sociétés importantes telles que Mozilla, Dropbox, Cloudflare, Discord, Facebook (Meta), Microsoft, et davantage utilisent Rust dans leurs logiciels. Dans les six dernières années, le langage Rust était le langage de programmation le plus « aimé » [5]

Développement des systèmes embarqués

Le développement des systèmes embarqués n'est pas aussi populaire que le développement web ou celui des logiciels personnels, voici quelques raisons pour lesquelles cela pourrait se révéler exact :

- Contraintes matérielles : les systèmes embarqués ont en général des ressources matérielles limitées, notamment au niveau de la mémoire et des performances. Cela rend plus difficile le développement du logiciel de ces systèmes.
- Marché niche plus réduit : le marché des systèmes embarqués est plus limité que celui des applications web ou personnelles, de ce fait, ils sont moins rémunérateurs pour les développeurs spécialisés en logiciels embarqués.
- Connaissances de bases spécifiques : la connaissance très spécialisée du matériel et des langages de programmation de bas-niveau est un besoin fondamental en développement de systèmes embarqués.
- Cycles de développement plus longs : le développement d'un logiciel pour système embarqué prend davantage de temps que celui d'une application personnelle ou web, en raison de la nécessité de tester et optimiser le code créé pour un matériel spécifique.
- Langages de programmation de bas-niveau : ces langages, comme par exemple l'assembleur ou le langage C, n'apportent pas une grande abstraction au programmeur et permettent l'accès direct aux ressources matérielles et à la mémoire, ce qui favorise les erreurs de programmation.

Seuls quelques exemples permettent de comprendre pourquoi et comment le développement de logiciels embarqués est unique est n'est pas aussi réputé et lucratif pour les jeunes programmeurs que ne l'est la programmation web. Si vous êtes habitués aux langages de programmation modernes les plus utilisés, comme Python, Javascript, ou C# avec lesquels vous n'avez pas à tenir compte de chaque cycle du processeur ou de chaque kilooctet de mémoire utilisé, débuter le développement de systèmes embarqués est un changement radical. Il peut être très décourageant d'aborder

Le support de la simultanéité et du parallélisme de Rust est particulièrement important pour le développement des systèmes embarqués où l'utilisation optimisée des ressources est importante.

le monde des systèmes embarqués, pas uniquement pour les débutants, mais aussi pour les développeurs expérimentés d'applications web, personnelles ou mobiles. C'est pourquoi il serait très intéressant, et nécessaire, de pouvoir disposer d'un langage de programmation moderne pour les systèmes embarqués.

Pourquoi Rust ?

Rust est un langage moderne relativement jeune qui se concentre sur la sécurité des accès mémoire et des processus, dont l'objectif est de produire des logiciels fiables et sécurisés. Le support de la simultanéité et du parallélisme est particulièrement important pour le développement de logiciels embarqués où l'utilisation optimisée des ressources est critique. La popularité grandissante de Rust et son écosystème en font une option intéressante pour les développeurs, particulièrement pour ceux qui recherchent un langage efficace et sécurisé. Ce sont les principales raisons pour lesquelles Rust devient un choix populaire, pas uniquement pour le développement des systèmes embarqués, mais particulièrement pour les projets qui ont pour priorité la sécurité, la protection et la fiabilité.

Avantages (en comparaison avec C/C++)

Passons en revue les avantages notables de Rust.

- Protection mémoire : Rust garantit une forte protection de la mémoire par son système de possession/prêt qui est très utile pour prévenir les bugs relatifs aux accès mémoire, tels que les pointeurs nuls, les déréférencements ou les dépassements de capacité des mémoires tampons. En d'autres termes, Rust garantit l'intégrité mémoire dès la phase de compilation grâce à son mécanisme possession/prêt. Ceci est particulièrement important dans les systèmes embarqués dans lesquels les ressources mémoire limitées rendent ces erreurs plus difficiles à localiser et corriger.
- Simultanéité : Rust apporte un excellent support pour les abstractions et la simultanéité sécurisée et le multi-tâche avec une syntaxe `async/await` standard et un type système puissant qui empêche les erreurs classiques telles que la concurrence des données. Cela facilite l'écriture de code concurrentiel sécurisé et performant, pas uniquement pour les systèmes embarqués.
- Performances : Rust est conçu pour obtenir des performances élevées et peut rivaliser avec C et C++ en ce domaine, tout en garantissant une sécurité mémoire élevée et le support de la simultanéité.
- Lisibilité : la syntaxe de Rust est conçue afin d'offrir une lisibilité accrue et de moins favoriser les erreurs que C ou C++, comme par exemple, le filtrage de vraisemblance, l'inférence de type, et la programmation fonctionnelle. Ceci facilite l'écriture et la maintenance du code, en particulier pour les projets complexes et les plus importants
- Essor de son écosystème : Rust possède un écosystème de bibliothèques grandissant (crates), d'outils, et de ressources pour (mais pas uniquement) le développement des systèmes embarqués. Rust le rend plus facile à aborder et pour trouver le support et les ressources nécessaires à un projet particulier.
- Gestionnaire de paquets et système de compilation : la distribution de Rust inclut un outil officiel appelé Cargo, qui est utilisé pour automatiser les processus de construction, test et publication, tout en permettant la création d'un nouveau projet et ses dépendances.

Désavantages (en comparaison avec C/C++)

Malgré tout, Rust n'est pas un langage parfait, il présente également quelques désavantages par rapport à d'autres langages (pas uniquement vis-à-vis de C ou C++).

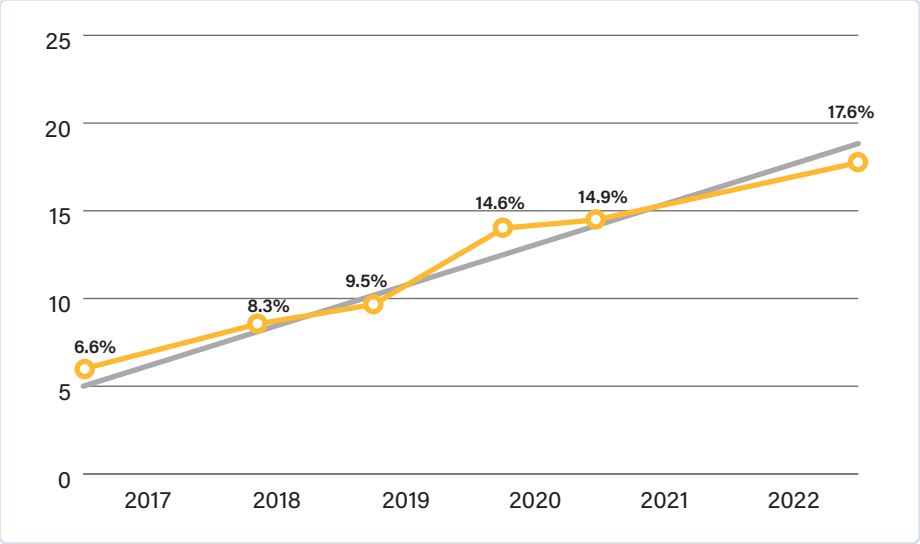


Figure 1. Accroissement du pourcentage de développeurs désirant développer en Rust. (Source : Yalantis [4])

- Temps d'apprentissage : Rust a une courbe d'apprentissage plus raide. Ses caractéristiques uniques, comme par exemple le mécanisme possession/prêt déjà mentionné, prennent davantage de temps pour leur compréhension et s'habituer à les utiliser, ce qui rend les débuts avec Rust plus difficiles.
- Temps de compilation : les particularités système et le vérificateur de prêt rendent le temps de compilation plus important par rapport à celui d'autres langages, en particulier dans le cas de projets d'envergure.
- Utilitaires : bien que l'écosystème de Rust croisse rapidement, il n'a pas encore atteint le même niveau de support que la plupart des autres langages de programmation. À titre d'exemple, C et C++ sont présents depuis plusieurs décennies et possèdent une base importante de code disponible. Cela peut rendre plus difficile la recherche et l'utilisation des outils appropriés à un projet spécifique.
- Manque de possibilités de contrôle de bas-niveau : les caractéristiques de protection de Rust imposent l'utilisation de C ou C++ pour le contrôle de bas-niveau. Cela rend plus délicat certaines opérations d'optimisation à bas-niveau ou l'interaction directe avec le matériel, mais cela reste possible.
- Taille de la communauté : Rust est un nouveau langage de programmation relativement jeune, comparativement avec les langages bien établis tels que C et C++. Cela implique que sa communauté de contributeurs et développeurs est plus réduite et qu'il dispose de moins de ressources, de bibliothèques de code et d'utilitaires.

Malgré tout, Rust présente de nombreux avantages par rapport aux langages de développement tels que C et C++, en particulier en matière de sécurité, simultanéité, performances, lisibilité du code et du fait de son écosystème en plein essor. De ce fait, Rust devient un choix de plus en plus populaire pour le développement des systèmes embarqués, notamment pour les projets qui imposent protection, sécurité et fiabilité. Les désavantages se Rust, par rapport à C et C++ sont surtout la conséquence de la jeunesse du langage Rust et de ses caractéristiques uniques. Malgré cela, de nombreux développeurs considèrent que les avantages de Rust le rendent compétitif pour certains projets.

Comment Rust fonctionne-t-il ?

Il existe plusieurs façons d'exécuter les logiciels basés sur Rust, selon l'environnement et les nécessités de l'application. Un code créé par Rust peut être utilisé selon deux modes : *environnement hôte* ou *bare-metal*. Voyons ce dont il s'agit.

Qu'est-ce que l'environnement hôte ?

L'*environnement hôte* de Rust s'approche de l'environnement habituel d'un PC [6], ce qui signifie qu'un système d'exploitation est présent. Avec un système d'exploitation, on peut créer la bibliothèque standard Rust (*std*). [7] Le terme *std* se réfère à la bibliothèque standard que l'on peut voir comme une collection de modules et types présents dans chaque installation Rust. La bibliothèque *std* offre un ensemble de fonctionnalités multiples permettant de créer des programmes Rust, incluant des structures de données, le support réseau, les mutex (accès exclusif à des ressources partagées) ainsi que les primitives de synchronisme, les entrées/sorties et bien davantage.

Selon l'approche de l'*environnement hôte*, on peut utiliser les fonctionnalités du cadre de développement C nommé ESP-IDF [8] car il offre l'environnement *newlib* [9], suffisamment puissant pour créer la bibliothèque standard *std* de Rust. En d'autres termes, grâce à l'*environnement hôte* de Rust (parfois nommé simplement *std*), on utilise ESP-IDF comme système d'exploitation et pouvons créer une application Rust superposée. De cette façon, on peut utiliser toutes les possibilités de la bibliothèque standard mentionnées précédemment, mais incorporer également des fonctionnalités C à partir des API ESP-IDF.

Dans le **listage 1**, vous pouvez voir comment se présente un exemple de clignotement d'une LED [10] développé à l'aide d'ESP-IDF (FreeRTOS). D'autres exemples se trouvent dans l'*esp-idf-hal* [11].

Quand est-il souhaitable d'utiliser l'environnement hôte ?

- Richesse des fonctionnalités : si votre système embarqué nécessite de nombreuses fonctionnalités, telles que le support de protocoles de communication, les entrées/sorties fichiers, ou des structures de données complexes, vous choisirez probablement l'approche de l'*environnement hôte* du fait de la présence de la bibliothèque *std* qui offre une large gamme de fonctions pouvant être utilisées pour la création relativement rapide et performante d'applications complexes.
- Portabilité : le contenu de la bibliothèque *std* apporte un ensemble d'API standardisées pouvant être utilisées avec un grand nombre de plateformes et architectures différentes, simplifiant l'écriture d'un code portable et réutilisable.
- Rapidité de développement : le contenu de *std* offre un ensemble de fonctionnalités riches pouvant être utilisées pour développer rapidement et efficacement des applications, sans devoir se préoccuper des détails de bas-niveau.

Qu'est-ce que le mode *Bare-Metal* ?

Bare-Metal (signifiant métal nu en anglais) signifie que l'on travaille sans système d'exploitation. Quand un programme

Listage 1. Exemple de clignotement d'une LED en mode environnement hôte et std

```
// Import peripherals we will use in the example
 use esp_idf_hal::delay::FreeRtos;
 use esp_idf_hal::gpio::*;
 use esp_idf_hal::peripherals::Peripherals;

 // Start of our main function i.e entry point of our example
 fn main() -> anyhow::Result<()> {
     // Apply some required ESP-IDF patches
     esp_idf_sys::link_patches();

     // Initialize all required peripherals
     let peripherals = Peripherals::take().unwrap();

     // Create led object as GPIO4 output pin
     let mut led = PinDriver::output(peripherals.pins.gpio4)?;

     // Infinite loop where we are constantly turning ON and OFF the LED every 500ms
     loop {
         led.set_high()?;
         // we are sleeping here to make sure the watchdog isn't triggered
         FreeRtos::delay_ms(1000);

         led.set_low()?;
         FreeRtos::delay_ms(1000);
     }
 }
```

Rust est compilé avec l'attribut `no_std`, cela signifie que le programme n'aura pas accès à certaines possibilités. Cela ne veut pas nécessairement dire qu'en mode `no_std`, vous ne pouvez pas utiliser la communication réseau ou des structures complexes. Sans `std`, vous pouvez réaliser les mêmes fonctions qu'en utilisant std, mais cela est plus complexe et difficile. La programmation `no_std` se base sur un ensemble de caractéristiques du langage qui sont présentes dans tous les environnements Rust, par exemple les types de données, structures de contrôle ou management de la mémoire à bas-niveau. Cette approche est utile pour la programmation des systèmes embarqués où l'utilisation de la mémoire est souvent contrainte et le contrôle direct du matériel nécessaire.

Dans le **listage 2**, vous trouverez comment se présente un exemple de clignotement d'une LED [12] fonctionnant en mode *bare-metal* (sans système d'exploitation). D'autres exemples se trouvent dans l'*esp-hal* [13].

Quand est-il souhaitable d'utiliser le mode Bare-Metal ?

- Taille mémoire réduite : si votre système embarqué a des ressources limitées et possède une taille mémoire réduite, il sera préférable d'utiliser le mode *bare-metal* car les fonctionnalités de *std* augmentent significativement la taille mémoire de son module exécutable et allonge le temps de compilation.
- Contrôle direct des ressources matérielles : si votre système embarqué nécessite davantage de contrôle direct du matériel, comme par exemple des pilotes de bas-niveau ou l'accès à des dispositifs matériels spécialisés, vous devrez préférer l'utilisation du mode *bare-metal* car *std* ajoute de l'abstraction rendant difficile l'interaction directe avec le matériel.
- Contraintes du temps réel ou applications au timing critique : si votre système embarqué nécessite des performances en temps-réel ou des temps de réponse à faible latence, le mode *bare-metal* sera préférable, car *std* introduit des délais imprévisibles et une surcharge qui peut altérer les performances.
- Besoins spécifiques : le mode *bare-metal* permet davantage de personnalisation et le contrôle précis du comportement d'une application ce qui peut être utile dans le cas des environnements spécialisés ou non usuels.

Devez-vous passer de C à Rust ?

Si vous commencez un nouveau projet ou une tâche dans laquelle la sécurité mémoire ou la simultanéité est exigée, il pourrait être profitable de considérer le passage de C à Rust. Néanmoins, si votre projet est déjà bien entamé et fonctionnel en C, le bénéfice du passage à Rust ne justifierait probablement pas le coût induit par la réécriture et le test de l'ensemble de votre code. Dans ce cas, il vous sera possible d'envisager de conserver l'ensemble de votre code C et d'utiliser Rust pour ajouter de nouvelles possibilités, modules et fonctionnalités. Il est relativement facile d'appeler des fonctions C à partir du code Rust. Il est également possible de créer des composants ESP-IDF en utilisant Rust [14]. En définitive, le passage de C à Rust doit suivre une évaluation spécifique des besoins et des bénéfices attendus.

VF : Jean Boyer — 230569-04

Questions ou commentaires ?

Envoyez un courriel à l'auteur (juraj.sadel@espressif.com) ou contactez Elektor (redaction@elektor.fr).

Listage 2. Exemple de clignotement d'une LED en mode bare-metal

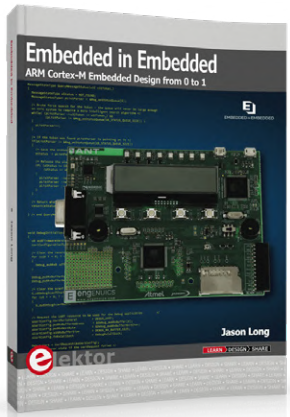
```
#![no_std]
 #![no_main]

 // Import peripherals we will use in the example
 use esp32c3_hal::{
     clock::ClockControl,
     gpio::IO,
     peripherals::Peripherals,
     prelude::*,
     timer::TimerGroup,
     Delay,
     Rtc,
 };
 use esp_backtrace as _;

 // Set a starting point for program execution
 // Because this is `no_std` program, we do not have a main function
 #[entry]
 fn main() -> ! {
     // Initialize all required peripherals
     let peripherals = Peripherals::take();
     let mut system = peripherals.SYSTEM.split();
     let clocks = ClockControl::boot_defaults(system.clock_control).freeze();

     // Disable the watchdog timers. For the ESP32-C3, this includes the Super WDT,
     // the RTC WDT, and the TIMG WDTs.
     let mut rtc = Rtc::new(peripherals.RTC_CNTL);
     let timer_group0 = TimerGroup::new(
         peripherals.TIMG0,
         &clocks,
         &mut system.peripheral_clock_control,
     );
     let mut wdt0 = timer_group0.wdt;
     let timer_group1 = TimerGroup::new(
         peripherals.TIMG1,
         &clocks,
         &mut system.peripheral_clock_control,
     );
     let mut wdt1 = timer_group1.wdt;

     rtc.swd.disable();
     rtc.rwdt.disable();
     wdt0.disable();
     wdt1.disable();

     // Set GPIO4 as an output, and set its state high initially.
     let io = IO::new(peripherals.GPIO, peripherals.IO_MUX);
     // Create led object as GPIO4 output pin
     let mut led = io.pins.gpio5.into_push_pull_output();

     // Turn on LED
     led.set_high().unwrap();

     // Initialize the Delay peripheral, and use it to toggle the LED state in a
     // loop.
     let mut delay = Delay::new(&clocks);

     // Infinite loop where we are constantly turning ON and OFF the LED every 500ms
     loop {
         led.toggle().unwrap();
         delay.delay_ms(500u32);
     }
 }
```

À propos de l'auteur

Juraj Sadel est un développeur de systèmes embarqués passionné par Rust, il se consacre à l'amélioration des systèmes embarqués. Il est également un membre actif de l'équipe Rust, faisant bénéficier la communauté de son expertise et de son enthousiasme pour la technologie de pointe d'Espressif.

Produits

- **J. Long, *Embedded in Embedded* (Elektor 2018)**
  version papier : www.elektor.fr/18876
  version numérique : www.elektor.fr/18877
- **A. He and L. He, *Embedded Operating System* (Elektor 2020)**
  version papier : www.elektor.fr/19228
  version numérique : www.elektor.fr/19214



LIENS

[1] MIT Technology Review, "How Rust went from a side project to the world's most-loved programming language," 2023: https://technologyreview.com/2023/02/14/1067869/rust-worlds-fastest-growing-programming-language
[2] Rust Blog, "Announcing Rust 1.0," 2015: https://blog.rust-lang.org/2015/05/15/Rust-1.0.html
[3] Rust Blog, "4 years of Rust," 2019: https://blog.rust-lang.org/2019/05/15/4-Years-Of-Rust.html
[4] Yalantis, "The state of the Rust market in 2023" : https://yalantis.com/blog/rust-market-overview
[5] Stack Overflow, "Stack Overflow Developer Survey," 2021: https://insights.stackoverflow.com/survey/2021#most-loved-dreaded-and-wanted
[6] The Embedded Rust Book: https://docs.rust-embedded.org/book/intro/no-std.html#hosted-environments
[7] The Rust Standard Library (std): https://doc.rust-lang.org/std
[8] ESP-IDF: https://github.com/espressif/esp-idf
[9] Newlib library: https://sourceware.org/newlib
[10] Blinky example running on top of ESP-IDF: https://github.com/esp-rs/esp-idf-hal/blob/master/examples/blinky.rs
[11] ESP-IDF-HAL: https://github.com/esp-rs/esp-idf-hal/tree/master/examples
[12] Blinky example running on bare-metal: https://github.com/esp-rs/esp-hal/blob/main/esp32c3-hal/examples/blinky.rs
[13] ESP-HAL: https://github.com/esp-rs/esp-hal/tree/main
[14] ESP-IDF components in Rust: https://github.com/espressif/rust-esp32-example

SDK

ESPRESSIF

ESP-Matter
SDK d'Espressif pour Matter

Espressif fournit une API facile à utiliser sur le SDK open-source « connectedhomeip » pour vous aider à réaliser facilement des projets compatibles avec Matter. Ce SDK prend en charge tous les SoC d'Espressif, tels que ESP32, ESP32-C3, ESP32-C2, ESP32-S3, ESP32-H2 et ESP32-C6. Il prend en charge le protocole Matter sur Wifi ainsi que Thread. Outre les exemples d'appareils Matter standard prenant en charge différents types d'appareils, il fournit également des implémentations des ponts Matter ZigBee, Matter BLE Mesh, et Matter ESP-Now.

https://github.com/espressif/esp-matter

Q&A avec Scott Mabin

qui sont les développeurs de solutions embarquées Rust ?

comment Espressif développe le langage Rust embarqué pour l'ESP32

Intro et questions par Stuart Cording (Elektor)

Il n'a qu'un peu plus de dix ans mais le langage de programmation Rust a déjà atteint la 17ème place dans l'index de la communauté de programmation TIOBE [1]. Il a gagné 194 places dans ce laps de temps pour se classer aux côtés de R, Ruby, Delphi, Scratch et MATLAB. De plus, Linus Torvalds, le père de Linux, a accepté des propositions visant à permettre l'utilisation de code écrit en Rust dans le noyau, ce que le C++ a tenté en vain de faire pendant des années. Le langage Rust a également de plus en plus d'adeptes parmi les développeurs de systèmes embarqués, ce sur quoi Espressif a décidé de s'appuyer.

Le langage C est le langage par défaut des systèmes embarqués depuis des décennies, laissant l'assembleur à ceux qui optimisent à la main ou développent des noyaux en temps réel. Développé dans les années 1970 par Dennis Ritchie alors qu'il travaillait aux Bell Labs, il est étroitement lié à la création du système d'exploitation Unix. Unix a été écrit en assembleur pour le PDP-7 [2] mais a dû être réécrit pour être porté sur le PDP-11 [3] (les PDP étaient des ordinateurs plus petits et polyvalents vendus comme alternatives aux ordinateurs centraux dans les années 1960). Le langage C a permis de développer un code, comme Unix, qui était indépendant du processeur et portable sur de nombreuses architectures différentes.

Les systèmes embarqués étaient initialement programmés en assembleur mais à mesure que le code devenait plus complexe et que les développeurs cherchaient à améliorer la réutilisation, ils ont, eux aussi, été attirés par le langage C. Parmi les autres caractéristiques qui leur ont facilité la vie, citons la prise en charge de la manipulation des bits de bas niveau, la définition fluide des variables, la facilité avec laquelle les algorithmes pouvaient être codés et la simplicité de la manipulation de la mémoire. Cependant, ce dernier point est à l'origine du blocage de la plupart des applications. Si les pointeurs permettent aux programmeurs d'accéder à (presque) n'importe quel emplacement de mémoire à n'importe quel moment, ce qui est très puissant et efficace dans un système embarqué, cela peut aussi être très dangereux. Les pointeurs peuvent pointer vers une mémoire longtemps après qu'elle a été désallouée ou la manipuler pour appeler une fonction qui entraîne une faille de sécurité. En fait, Microsoft attribue environ 70 % des problèmes logiciels en C/C++ à des bogues de corruption de la mémoire [4].

Rust s'attaque à ce problème en renforçant la sécurité de la mémoire. En outre, contrairement au code C/C++, de nombreux problèmes de programmation sont détectés à la compilation plutôt qu'à l'exécution. Enfin, grâce aux similitudes de syntaxe et de performance, les développeurs C et C++ se sentiront rapidement à l'aise, tant sur leur PC que sur les systèmes embarqués. Alors, dans quelle mesure les fabricants de microcontrôleurs prennent-ils au sérieux le langage Rust pour les systèmes embarqués ? Pour en savoir plus, Elektor s'est entretenu avec Scott Mabin, un jeune développeur qui, libéré du carcan des précédents langages, a décidé de se plonger encore plus profondément dans le monde de Rust. Utilisateur assidu des ESP32, il a beaucoup contribué au langage Rust embarqué, ce qui lui a valu d'être embauché par Espressif.

Stuart Cording : les systèmes embarqués sont traditionnellement programmés en C. Mais vous avez décidé d'ignorer cela et de vous lancer directement dans Rust. Pourquoi ce choix ?

Scott Mabin : j'ai utilisé Rust dans mon projet de fin d'études à l'université. J'ai construit une montre intelligente – pas intelligente selon les normes actuelles, mais elle faisait plus que donner l'heure. Une partie de ce projet visait à déterminer si Rust pouvait remplacer

le langage C pour le développement de micrologiciels et quels étaient les compromis possibles. C'est à ce moment-là que j'en suis tombé amoureux. Je me suis demandé pourquoi tout le monde n'utilisait pas Rust. Il offrait tellement de possibilités qu'une fois compilé, on avait la tranquillité d'esprit de savoir que toute une catégorie de situations de compétition et de mémoire avaient été éliminées. Bien sûr, je comprends que certains projets ont un héritage et que l'on ne veuille pas apprendre et déployer un nouveau langage sans raison valable. Néanmoins, il me semblait insensé qu'il n'y ait pas plus de gens qui se tournent vers Rust.

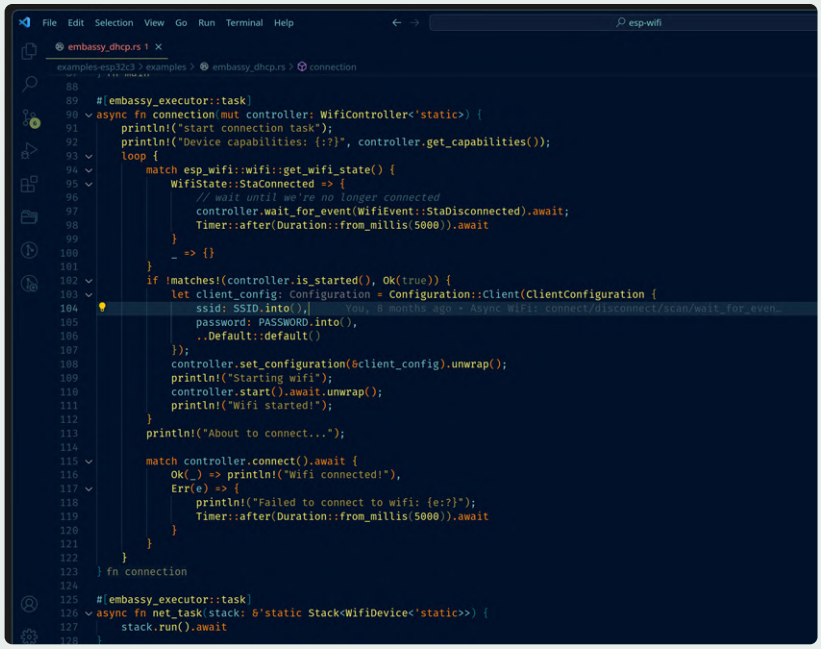
Stuart Cording : Rust est déjà supporté par la communauté sur STM32 et certains appareils Nordic. Pourquoi avoir choisi Espressif à l'époque ?
Scott Mabin : à la maison, j'avais tous ces ESP32 qui traînaient et je me suis dit : « je peux sûrement aussi programmer ces choses en Rust ? ». Et bien, cela m'a rapidement conduit dans une voie inconnue assez opaque. Le plus gros problème était que le noyau, Xtensa, qui alimente les appareils tels que l'ESP8266, n'était supporté que par le compilateur GCC et non pas par LLVM. Je me suis tourné vers *mrustc* [5], un outil qui convertit Rust en C et peut ensuite être compilé mais ce processus de compilation croisée ne m'a pas convaincu. Heureusement, Espressif a publié un embranchement (fork) pour la chaîne de compilation LLVM afin de supporter Xtensa et, bien qu'initialement difficile à utiliser, cela signifiait que vous pouviez compiler du Rust natif pour les appareils ESP32. Grâce à cela, j'ai écrit mon premier code de LED clignotante en Rust ; enfin, je dis « en Rust » mais il se contentait d'écrire dans des registres et n'était pas très Rust dans sa structure. Cependant, c'était un premier succès que j'ai documenté dans mon premier article de blog [6] relatant mon expérience avec Rust sur ESP32.

Stuart Cording : vous avez donc commencé à bricoler avec Rust sur ESP32. Comment s'est déroulée la relation avec Espressif ?
Scott Mabin : j'ai créé le groupe *esp-rs* sur GitHub pendant mon temps libre pour rassembler des projets Rust pour ESP32. Avec d'autres membres de la communauté, nous avons mis en place un support pour les périphériques, tels que SPI et autres. Malheureusement, le Wifi nous échappait encore. Puis, après un an de travail communautaire et d'échange sur nos progrès, Espressif a proposé de m'embaucher pour assurer le support de Rust pour leur écosystème. L'équipe d'Espressif s'est toujours montrée très serviable à notre égard, répondant aux demandes de support sur leurs forums ou Reddit. Mais lorsqu'ils m'ont contacté, il est apparu clairement qu'ils s'intéressaient à Rust depuis un certain temps.

Stuart Cording : y a-t-il des domaines d'application spécifiques qui suscitent cet intérêt pour Rust ou s'agit-il d'une sorte de boule de cristal vaporeuse de la part d'Espressif ? Et quel est leur niveau d'implication ?
Scott Mabin : l'intérêt des clients est indéniable. Les clients utilisent principalement Rust dans des projets IdO ou des applications connectées à l'internet. La plupart des projets sont assez simples, comme les enregistreurs de données. Ensuite, il y a des applications plus complexes telles que la capture de mouvement VR pour le corps entier et le capteur de ciel nocturne en matériel libre. Espressif s'efforce également de préparer l'avenir, en prévision du moment où Rust jouera un rôle plus important dans la vie des développeurs de systèmes embarqués. Environ 10 ou 11 d'entre nous contribuent à l'équipe « Rust Enablement » d'Espressif, dont environ la moitié travaillent à temps plein. De plus, « Rust embedded » dispose d'une communauté impressionnante qui contribue également.

Stuart Cording : vous avez fait du développement en C embarqué et vous utilisez maintenant Rust quotidiennement sur des microcontrôleurs. Qu'est-ce que les programmeurs C traditionnels doivent prendre en compte lorsqu'ils passent à Rust ?
Scott Mabin : si vous venez du développement en C pur et que vous n'avez aucune expérience du C++, c'est assez difficile. Le changement le plus radical est que Rust nécessite beaucoup d'écriture. Ensuite, il y a l'aspect ressources. Les petits microcontrôleurs dotés de quelques kilo-octets de mémoires SRAM et flash auront du mal à s'adapter en raison des contraintes de mémoire. Dans de tels cas, vous finirez par écrire du Rust de type C pour réduire la taille de la mémoire, perdant ainsi certains des avantages de ce nouveau langage. Vous bénéficierez toujours des avantages de Rust en matière de sécurité de la mémoire, donc ce langage reste donc meilleur que le C. Si on les compare dos à dos, les applications en C et en Rust ont des performances à peu près équivalentes. Dans certains cas, Rust est meilleur, mais la taille du programme est généralement plus importante.

Stuart Cording : alors que les fournisseurs de microcontrôleurs offrent un excellent support pour les pilotes de périphériques, le contrôle de la version de tout ce code pour le développeur en tant qu'intégrateur est souvent une belle pagaille. Les bibliothèques (crates) Rust seront-ils une raison de changer de langage de programmation ?
Scott Mabin : oui, je pense que les bibliothèques sont un énorme avantage – un gros bonus qui va au-delà du langage lui-même. Les bibliothèques offrent un écosystème d'empaquetage ou, au minimum, une



```rust
#[embassy_executor::task]
async fn connection(mut controller: WifiController<'static>) {
    println!("start connection task");
    println!("Device capabilities: {:?}", controller.get_capabilities());
    loop {
        match esp_wifi::wifi::get_wifi_state() {
            WifiState::StaConnected => {
                // wait until we're no longer connected
                controller.wait_for_event(WifiEvent::StaDisconnected).await;
                Timer::after(Duration::from_millis(5000)).await
            }
            _ => {}
        }
        if !matches!(controller.is_started(), Ok(true)) {
            let client_config: Configuration = Configuration::Client(ClientConfiguration {
                ssid: SSID.into(),
                password: PASSWORD.into(),
                ..Default::default()
            });
            controller.set_configuration(&client_config).unwrap();
            println!("Starting wifi");
            controller.start().await.unwrap();
            println!("Wifi started!");
        }
        println!("About to connect...");

        match controller.connect().await {
            Ok(_) => println!("Wifi connected!"),
            Err(e) => {
                println!("Failed to connect to wifi: {e:?}");
                Timer::after(Duration::from_millis(5000)).await
            }
        }
    }
}

#[embassy_executor::task]
async fn net_task(stack: &'static Stack<WifiDevice<'static>>) {
    stack.run().await
}
```

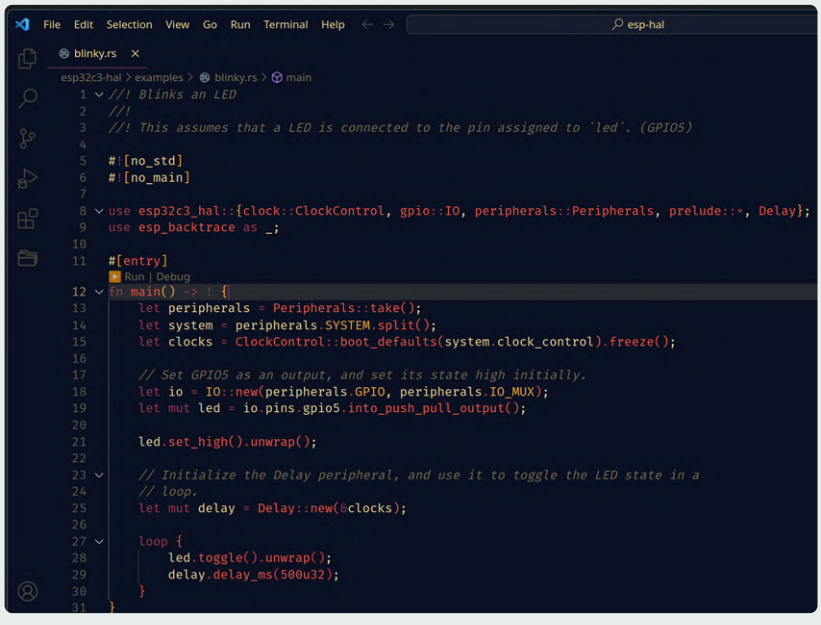
▲

Figure 1. Exemple d'application Wifi écrite en Rust pour l'ESP32.

méthode pour intégrer des dépendances sans les définir manuellement dans votre système de construction. La bibliothèque *embedded_hal* [7] offre une interface fonctionnelles pour les périphériques communs, tels que I²C. Si vous prenez maintenant un pilote de capteur de température I²C, il fonctionne simplement par-dessus. Si vous changez de microcontrôleur ou de capteur de température, il fonctionnera toujours, ce qui est très sympa. Mais cela ne fait qu'effleurer la surface. Les bibliothèques telles que *heapless* [8] permettent la création de structures de données allouées statiquement telles que les files d'attente, les vecteurs, les tables de hachage, les ensembles de hachage et les chaînes de caractères. Dans tous les projets C sur lesquels j'ai travaillé, quelqu'un s'assied et écrit une file d'attente, passant de nombreuses heures à perfectionner une structure de données de programmation standard. Ce temps peut désormais être consacré au développement d'applications.

Figure 2. Scott Mabin recommande VS Code pour le développement de Rust. Ici, on peut voir le code classique de la LED clignotante.

▼



```rust
//! Blinks an LED
//!
//! This assumes that a LED is connected to the pin assigned to `led`. (GPIO5)

#![no_std]
#![no_main]

use esp32c3_hal::{clock::ClockControl, gpio::IO, peripherals::Peripherals, prelude::*, Delay};
use esp_backtrace as _;

#[entry]
Run | Debug
fn main() -> ! {
    let peripherals = Peripherals::take();
    let system = peripherals.SYSTEM.split();
    let clocks = ClockControl::boot_defaults(system.clock_control).freeze();

    // Set GPIO5 as an output, and set its state high initially.
    let io = IO::new(peripherals.GPIO, peripherals.IO_MUX);
    let mut led = io.pins.gpio5.into_push_pull_output();

    led.set_high().unwrap();

    // Initialize the Delay peripheral, and use it to toggle the LED state in a
    // loop.
    let mut delay = Delay::new(&clocks);

    loop {
        led.toggle().unwrap();
        delay.delay_ms(500u32);
    }
}
```

Stuart Cording : tout comme le C, Rust propose une bibliothèque standard. Les développeurs d'applications embarquées détestent la bibliothèque standard, alors peuvent-ils aussi se passer de celle de Rust ?

Scott Mabin : Historiquement, je pense que nous avons eu du mal à décrire les cas d'utilisation pour lesquels un programmeur embarqué professionnel devrait utiliser une bibliothèque standard [9]. Ils jettent un coup d'œil et se disent « c'est beaucoup trop de travail » et je suis tout à fait d'accord avec eux. Cependant, la bibliothèque standard Rust est vraiment utile à plusieurs égards. Tout d'abord, si vous connaissez Rust mais pas de solution embarquée, ça vous semble familier. Vous avez toutes les tâches, le réseau et d'autres choses auxquelles vous êtes habitués. Deuxièmement, si vous programmez des ESP32 en utilisant l'ESP-IDF [10] (Espressif IoT Development Framework), vous pouvez créer des projets de bibliothèque standard Rust et commencer à travailler. Si vous la comparez à la bibliothèque standard Python, elle est en fait assez légère et, bien qu'il y ait du travail, elle n'est pas aussi mauvaise qu'elle en a l'air. Elle pose quelques principes supplémentaires par rapport à la bibliothèque de base (dont vous aurez besoin au minimum), telles que la disponibilité des tâches et du réseau.

De nombreuses personnes développent des systèmes Rust embarqués sans la bibliothèque standard (*no_std* [11]) et, si elles savent ce qu'elles font, optent pour l'approche *no_std* si cela a du sens pour le projet. Cependant, vous devrez choisir les éléments que vous voulez et les fusionner ensemble. Ainsi, par exemple, la bibliothèque Wifi contient un exemple de conversation avec un serveur HTTP ou de mise en place d'un point d'accès et d'exécution d'un serveur (**figure 1**). Vous pouvez faire dans *no_std* tout ce que vous pouvez faire avec *std* – c'est juste plus de travail. Avec *std*, c'est un environnement « tout inclus ».

Stuart Cording : une partie du problème réside-t-elle dans le fait qu'il est temps d'accepter que nous avons besoin de microcontrôleurs dotés de grandes mémoires ?

Scott Mabin : Je pense que, quelle que soit la taille des microcontrôleurs, il y aura toujours quelqu'un qui voudra fabriquer un million d'appareils et qui aura besoin de la solution la plus petite et la moins chère possible. Il y aura toujours un cas d'utilisation pour l'assembleur et le C – et peut-être même le Rust – dans un contexte spécifique pour cette petite application compacte. Mais je pense qu'il y a une tendance générale vers des microcontrôleurs plus gros et que l'expérience du développeur devient prioritaire sur les coûts du matériel. Ce n'est qu'une observation que j'ai faite au cours de ma courte période dans l'industrie, mais c'est ce qui semble se dessiner.

Figure 3. L'ESP32-C3 intègre un module de débogage USB avec une interface série, ce qui évite d'avoir recours à des sondes externes.

Stuart Cording : Rust n'en est qu'à ses débuts. Comment Espressif va-t-il assurer la formation des développeurs ?
Scott Mabin : nous nous sommes associés à Ferrous Systems [12], une société de conseil en Rust, pour proposer un cours de formation que nous avons rendu open source. Il existe des supports de formation pour l'approche de la bibliothèque standard et nous avons presque terminé d'ajouter l'approche de la bibliothèque non standard.

Stuart Cording : qu'en est-il des environnements de développement et des outils de débogage ?
Scott Mabin : j'admire l'esthétique d'environnements tels que vim, Neovim ou Emacs, mais comme je n'ai pas l'expérience de ces outils, ils me gênent. J'utilise donc VS Code (**figure 2**) qui fait l'affaire pour moi. Avec les ESP plus récents (C3 et plus), nous avons un module appelé USB Serial JTAG. Il s'agit d'un périphérique intégré extrêmement petit qui offre un port série et un dispositif JTAG (**figure 3**). Pour l'utiliser, il suffit de le connecter au port USB de votre PC et il est détecté automatiquement par OpenOCD ou *probe-rs*, une boîte à outils de débogage Rust dont l'objectif est similaire à celui d'OpenOCD.

Stuart Cording : la prise en charge d'un nouveau langage sur un processeur est une tâche énorme. Où en êtes-vous et que reste-t-il à faire ?
Scott Mabin : nous avons pratiquement tout mis en place pour la bibliothèque standard mais nous n'avons pas tout couvert pour ESP-IDF. ESP-IDF est vaste, existe depuis des années et est écrit en C. Nous utilisons donc *bindgen* pour créer des liens avec Rust. Fondamentalement, nous devons examiner les interfaces qui ne sont pas encore implémentées et créer une API Rust pour elles. Nous devons décider des priorités au cas par cas. Pour *no_std*, nous parlons de programmation en Rust pur, donc nous devons écrire du code pour tout le matériel existant. Nous avons le Wifi, le Bluetooth et le support Thread sur toutes les puces. Sur une échelle de 0 à 100, je dirais que nous avons fait entre 65 et 70 % du chemin. Malheureusement, il s'agit d'un objectif un peu mouvant car nous devons continuellement prendre en charge de nouveaux appareils au fur et à mesure de leur sortie. Nous avons discuté de la possibilité de créer certains éléments de l'ESP-IDF en Rust lorsque cela s'avérait judicieux. Par exemple, Rust est très bien adapté au passage de flux d'octets. Ainsi, si un nouveau composant est nécessaire, nous le créerons peut-être en Rust et fournirons une API C. Nous devrons voir si les avantages l'emportent sur les efforts.

Stuart Cording : enfin, où allez-vous, en tant que développeur de matériel embarqué, pour obtenir plus d'informations sur Rust ?
Scott Mabin : Je traîne surtout sur le canal ESP Rust matrix [13] et sur divers autres canaux Rust embedded. À part cela, Google, parfois Reddit ; lire du bon code Rust peut m'aider à comprendre et à apprendre des choses que je ne connaissais pas

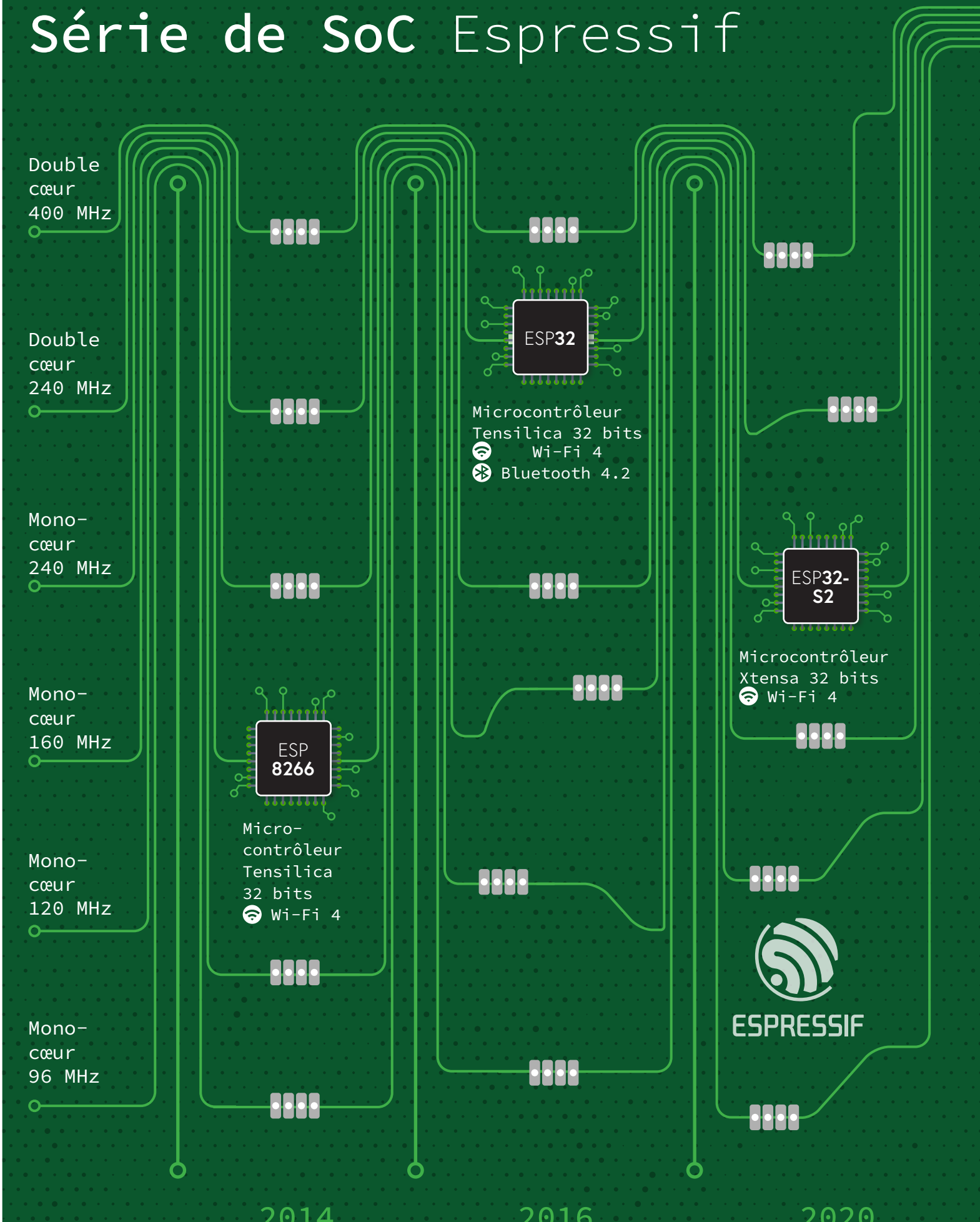
VF : Chris Elsass — 230616-04

Questions ou commentaires ?
Contactez Elektor (redaction@elektor.fr).

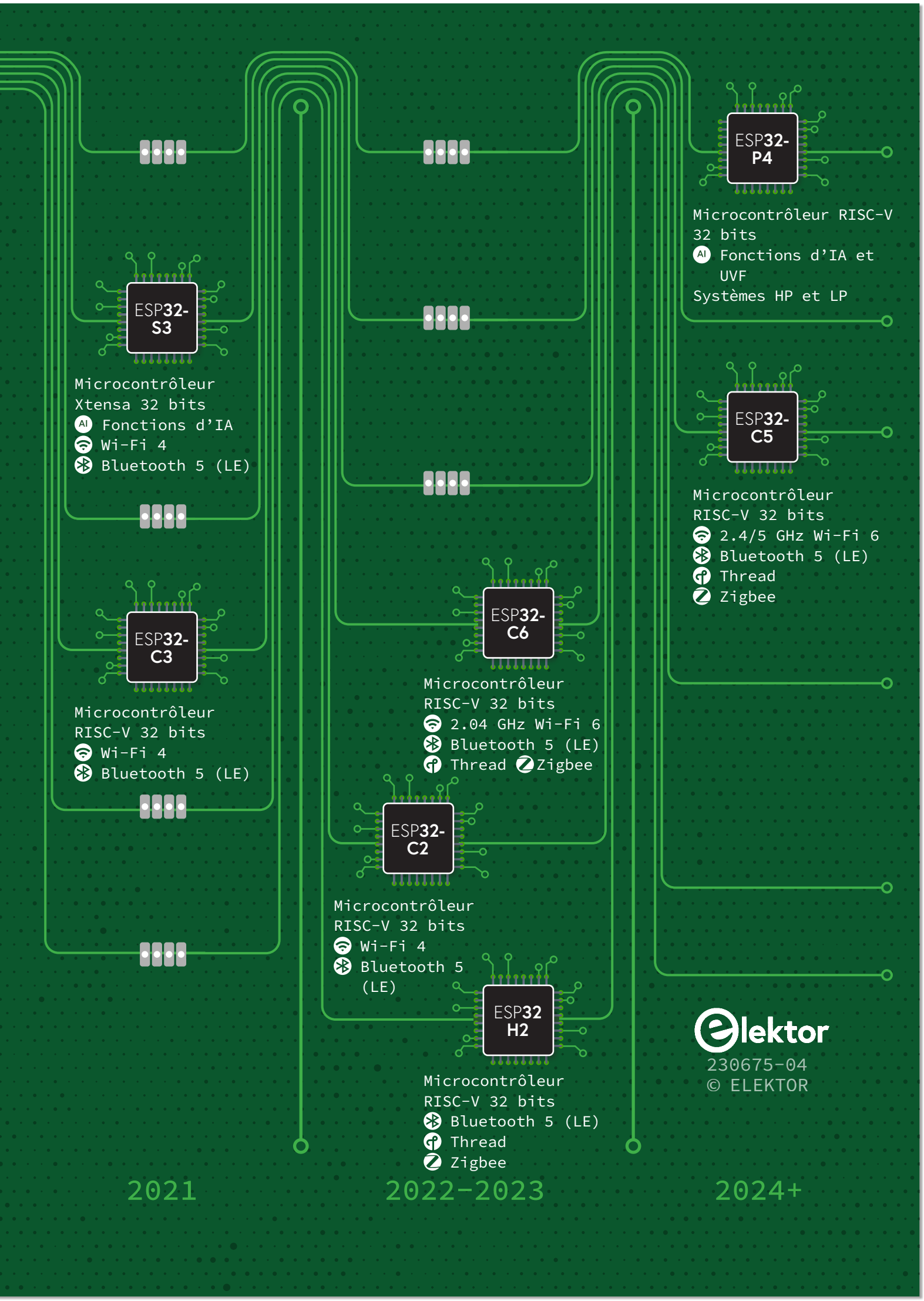
À propos de Scott
Scott Mabin est ingénieur en logiciels embarqués. Après avoir exploré les capacités de Rust embarqué à l'université, il a décidé de l'expérimenter sur l'ESP32 pendant son temps libre. Ces efforts ont conduit Espressif à lui demander de rejoindre leur équipe pour se concentrer sur l'amélioration du support de Rust sur leurs microcontrôleurs sans fil et leurs solutions AIoT.

LIENS
[1] Index de la communauté de programmation TIOBE : https://tinyurl.com/tioberust
[2] PDP-7 [Wikipédia] : https://tinyurl.com/pdp7wikipedia
[3] PDP-11 [Wikipédia] : https://tinyurl.com/pdp11wikipedia
[4] Équipe du CSEM, « une approche proactive d'un codage plus sécurisé », Microsoft, juillet 2019 : https://tinyurl.com/msrcsecurecode
[5] Projet mrustc : https://tinyurl.com/mrustcproject
[6] S. Mabin, « Rust sur l'ESP32 », septembre 2019 : https://tinyurl.com/rustesp32
[7] Documentation de la bibliothèque embedded_hal : https://tinyurl.com/embeddedhalcrate
[8] heapless Crate Documentation : https://tinyurl.com/heaplesscrate
[9] The Rust on ESP Book, "Using the Standard Library (std)": https://tinyurl.com/rustbookstd
[10] Guide de Programmation ESP-IDF « Get started » : https://tinyurl.com/espidfgetstarted
[11] The Rust on ESP Book, "Using the Core Library (no_std)" : https://tinyurl.com/rustbooknostd
[12] Formation « Embedded Rust on Espressif » : https://tinyurl.com/rustonespressif
[13] Rust user group on Matrix : https://tinyurl.com/rustmatrixug

Série de SoC Espressif



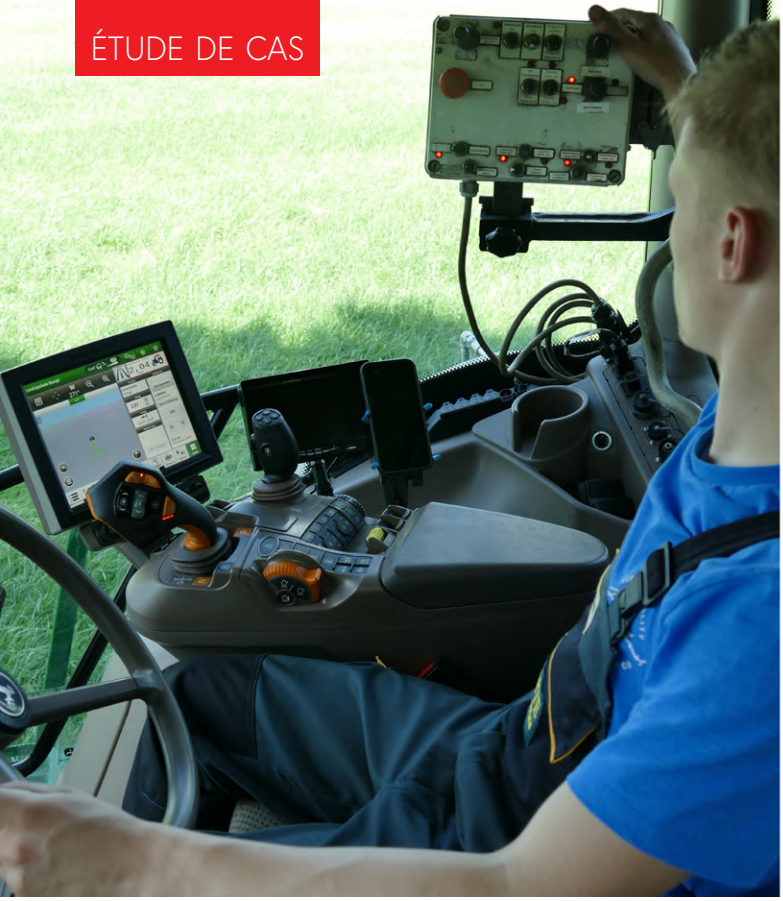
elektormagazine.fr/posters



elektor

230675-04

© ELEKTOR



une API avec les solutions d'Espressif

avec les capacités et les fonctionnalités du protocole ISOBUS

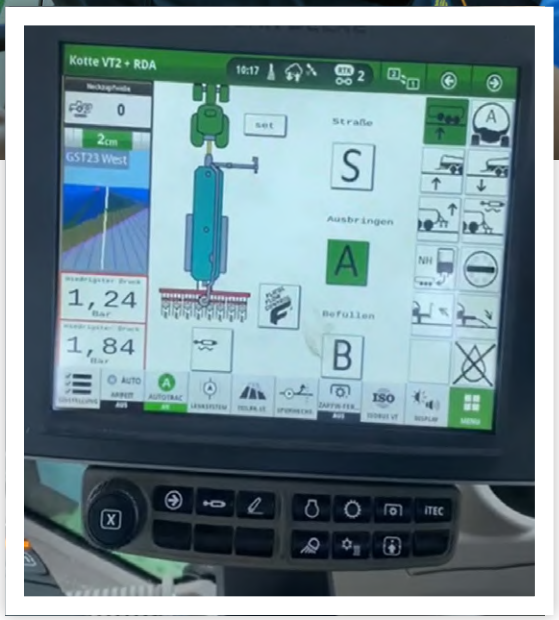


Figure 1. Interface utilisateur ISOBUS (ISO 11783) sur un moniteur John Deere. Il contrôle un slurry tanker.

Franz Höpfinger, HR Agrartechnik

HR Agrartechnik GmbH avait besoin d'une API peu coûteuse mais puissante, dotée de capacités ISOBUS (IOS 11783). La combinaison d'un ESP32 d'Espressif avec l'ESP-IDF, ainsi que le cadre Eclipse 4diac™, a donné naissance au projet logiBUS®.

Le défi

Les machines agricoles modernes nécessitent des systèmes de commande flexibles. Les nombres sont généralement faibles et la durée de vie des machines est longue, c'est pourquoi l'adaptation des systèmes de commande aux machines existantes sur le terrain est assez courante. Dans la **figure 1**, vous pouvez voir l'interface utilisateur du moniteur de contrôle du système après la mise à niveau.

L'apprentissage du langage C ou C++ et la gestion des bibliothèques et des systèmes sont complexes. Le débogage avec JTAG n'est pas flexible et nécessite des outils supplémentaires et un accès au microcontrôleur. En revanche, de nombreux programmeurs sont disponibles pour la programmation d'automates, et la programmation graphique est plus facile à apprendre.

ISOBUS, également connu sous le nom d'ISO 11783, est un protocole de communication développé pour les machines agricoles afin de faciliter l'échange de données entre différents types de machines et d'outils agricoles. ISOBUS se compose d'ISO (International Standardisation Organisation) et de BUS, qui est un système de communication entre plusieurs appareils.

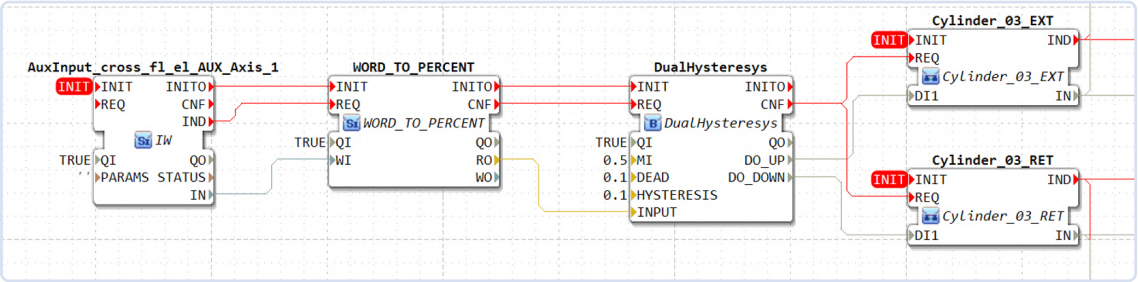


Figure 2. Ce listage du code source montre comment fonctionne la programmation dans la CEI 61499/Eclipse 4diac™. Vous n'écrivez pas de lignes de code, mais vous utilisez des blocs fonctionnels à la place.

Dans ce contexte, nous voulions concevoir un système API composé d'un système d'exécution sur le contrôleur lui-même et d'un EDI sur le PC qui fournirait un système d'automatisation flexible, convivial et rapide avec une programmation graphique à code bas basé sur des modèles et des bibliothèques puissants. Une séparation stricte entre un environnement d'exécution réutilisable et flexible et l'application figurait en tête de notre liste de souhaits. Le deuxième point était la possibilité d'observer réellement les variables et les blocs fonctionnels en ligne. D'autres points importants étaient la qualité du support des fournisseurs pour la pile logicielle et l'utilisation des ressources. Nous voulions éviter un système LINUX coûteux et entièrement intégré, principalement en raison du temps de démarrage. Les équipements agricoles sont susceptibles d'être allumés et éteints des dizaines de fois par jour.

Nous avons analysé plusieurs solutions, à la fois à code fermé et à code ouvert, et avons finalement choisi Eclipse 4diac™ [1]. « Pourquoi pas le projet OpenPLC ? », se demanderont peut-être certains lecteurs d'Elektor. La réponse est évidente : les deux principales exigences ne sont pas remplies : OpenPLC ne présente pas de séparation stricte entre l'application et le temps d'exécution, ni de fonctionnalité de veille en ligne.

La solution

Pour la solution logiBUS® [2], nous avons combiné l'aspect logiciel (cité ci-dessous) avec notre application de base :

- Cadre de développement ESP-IDF (actuellement Version 5.1.1)
- Runtime IEC 61499 open-source: Eclipse 4diac™ FORTE PLC
- Pilote CCI ISOBUS, une pile de protocoles ISO 11783

Il en résulte un système d'exécution d'automate indépendant de l'application souhaitée. La programmation se fait avec l'EDI 4diac™ d'Eclipse. Le flashage de l'application peut se faire via une connexion TCP/IP, donc via le Wifi ou l'Ethernet. Les solutions apportent de véritables fonctions d'automate telles que la surveillance et la modification en ligne.

Pour le matériel, un ESP32 avec PSRAM est un bon choix parce que le modèle utilise beaucoup de RAM, donc pour les modèles plus grands, un manque de PSRAM est un problème. Pour la connexion ISOBUS, nous utilisons le périphérique TWAI de l'ESP32 comme contrôleur de bus CAN, ainsi qu'un émetteur-récepteur CAN TLE9251VSJ externe d'Infineon. Certaines versions de matériel, en particulier ceux destinés à l'enseignement, sont libres [6].

Le projet logiBUS® ne cessant de se développer et de s'étendre, nous avons déjà réalisé une douzaine de projets clients réels ; la **figure 2** présente un exemple de listage du code source pour l'un d'entre eux.

Le système semble également adapté à d'autres secteurs, tels que l'automatisation de bâtiment [5]. Nous avons donc créé une version open-source de logiBUS® sans la pile ISOBUS, mais avec les mêmes fonctions et la même puissance.

Nous avons également mis en libre accès quelques applications de démonstration, telles que le Bale Counter [4] et le Slurry Tanker [5].

Nous espérons créer une communauté autour du thème des API open source [6].

230610-04

Questions ou commentaires ?
Contactez Elektor (redaction@elektor.fr).

LIENS

[1] Eclipse 4diac™ : https://eclipse.dev/4diac
[2] Page d'accueil de logiBUS® : https://www.logibus.tech
[3] Dérivés open-source de logiBUS® pour, par exemple, l'automatisation de bâtiment (pas ISOBUS) : https://gitlab.com/meisterschulen-am-ostbahnhof-munchen
[4] Bale Counter : https://github.com/Meisterschulen-am-Ostbahnhof-Munchen/4diac_EasyExampleCounter
[5] Slurry Tanker : https://github.com/Meisterschulen-am-Ostbahnhof-Munchen/4diac-SlurryTanker-sample
[6] Ressources et Wiki : https://github.com/Meisterschulen-am-Ostbahnhof-Munchen

la carte VGA ESP32-S3

le voyage passionnant de Bitluni dans la conception de produits

compilé par Elektor et Espressif

Le youtubeur Bitluni a conçu une carte VGA ESP32-S3 pour atteindre une résolution et une fidélité de couleurs remarquables. Il a utilisé le périphérique LCD de l'ESP32-S3 d'Espressif, qui a remplacé le I2S de la version précédente. Revenons sur ses étapes. Vous en apprendrez beaucoup sur le processus de création d'un nouveau produit.

Il y a quelques années, le célèbre youtubeur allemand Bitluni a réalisé une bibliothèque VGA et quelques cartes VGA pour l'ESP32 (**figure 1**). Beaucoup de ses fidèles abonnés ont adoré le design, mais il admet qu'il n'est pas un fabricant et qu'il n'en a vendu qu'un nombre limité. Les choses ont changé depuis ! L'API de l'ESP32 a été mise à jour plusieurs fois, et l'ESP32-S3 est arrivé sur le marché (**figure 2**). Comme beaucoup de ses abonnés lui demandaient de créer une nouvelle version de la bibliothèque et de la carte puisqu'elle était incompatible avec le S3, Bitluni a décidé de tenter le coup. C'est ainsi qu'est né le VGA ESP32-S3 (**figure 3**).

Lorsque Bitluni a décidé de concevoir la nouvelle carte (**figure 4**), il voulait qu'elle soit plus facile à utiliser. Les cartes précédentes nécessitaient un assemblage et une carte de développement supplémentaire. Heureusement, il avait acquis suffisamment de pratique pour en concevoir une qui comprenait tout (**figure 5**). *Plug-and-play* sans beaucoup de soudage ! Il voulait notamment que la carte soit compatible avec une plaque d'essai, de sorte que si vous soudiez les connecteurs, vous puissiez la placer sur une plaque d'essai et avoir deux rangées supplémentaires pour accéder aux broches (**figure 6**). Le connecteur VGA était assez grand, et l'antenne avait besoin d'un peu d'espace, donc ce format semblait raisonnable.

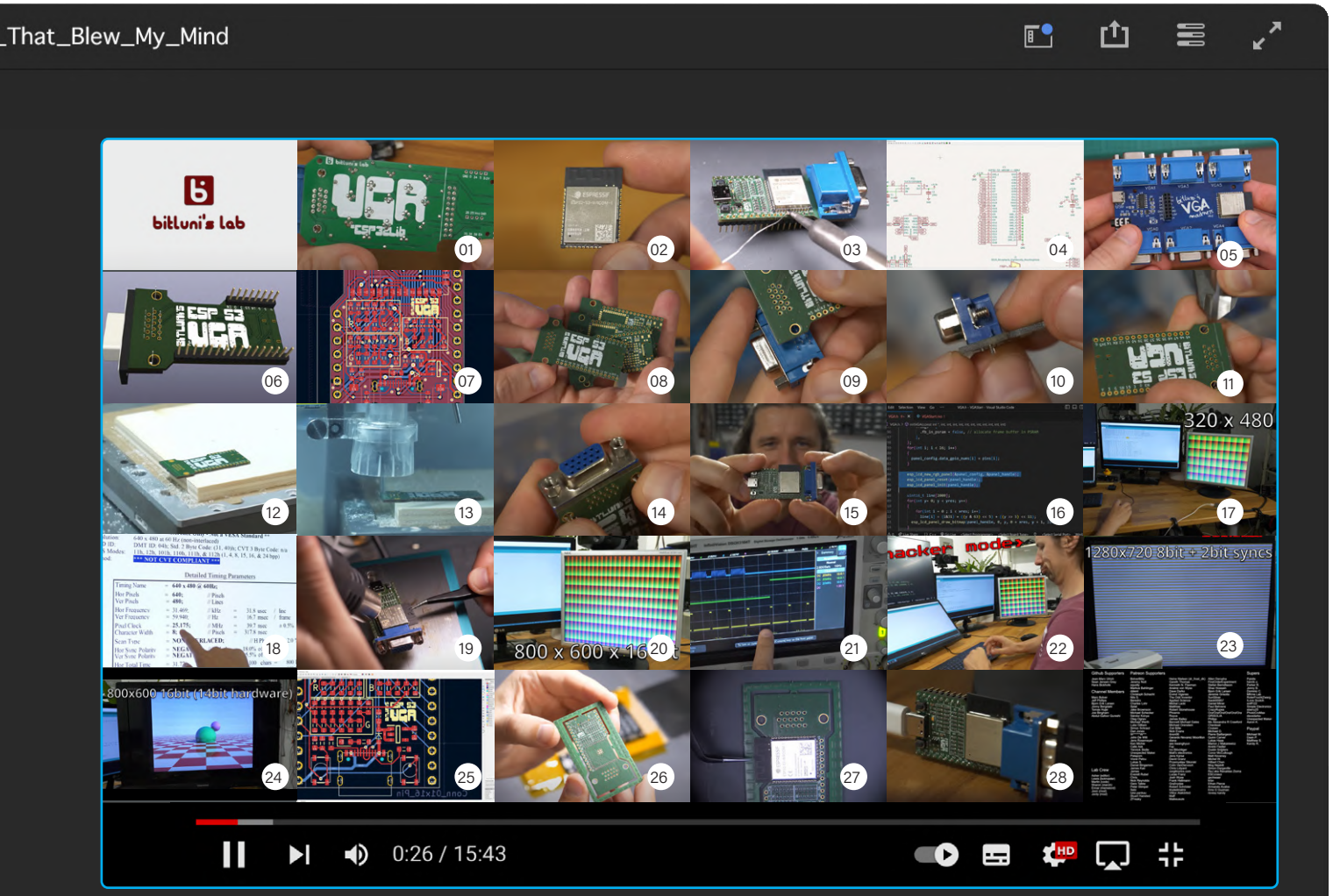
La conception a commencé (**figure 7**), les cartes ont été commandées, et les résultats étaient magnifiques (**figure 8**) ! Bitluni pensait que les connecteurs VGA dont il disposait seraient compatibles avec les empreintes des circuits imprimés, mais il a rapidement découvert que les broches ne correspondaient pas (**figure 9**) ! Heureusement, il disposait de quelques connecteurs plus étroits. Ceux-ci s'adaptaient à peine aux broches et à la carte, il y avait un débordement (**figure 10**). Comme il n'y avait pas de pistes dans cette zone, il a décidé de simplement fraiser quelques millimètres et le problème a été résolu (**figures 11...14**) ! L'assemblage étant terminé, il a pu commencer à coder (**figure 15**).

Jusqu'à ce stade, Bitluni n'avait pas consulté le manuel de référence technique, mais une fois qu'il l'a fait, il a compris pourquoi son ancienne bibliothèque ne fonctionnait pas avec le S3. Espressif avait supprimé le mode parallèle I2S et créé un nouveau périphérique pour les caméras et les afficheurs LCD. Il a donc étudié (**figure 16**), réfléchi et testé (**figure 17**) ! Puis plus d'études, plus de réflexion (**figure 18**), plus de tests, puis du dessoudage et de la reconfiguration, qui ont été suivis par plus de reconfigurations, plus de soudage, et plus de tests (**figure 19**). Il n'a pas chômé ! Heureusement, après tous ces efforts, ça a marché ! Non seulement avec une résolution de 640×480, mais aussi avec 800×600 (**figure 20**).

Après quelques tests sur différents appareils, Bitluni a découvert des problèmes de synchronisation. Trois de ses autres écrans semblaient modifier la synchronisation au début de chaque image. C'était un obstacle, explique-t-il : une fois qu'il a zoomé sur son scope, il a constaté un léger retard à chaque début d'image (**figure 21**). Il a donc continué à étudier et à réfléchir, et a activé le mode *hacker* (**figure 22**) !

Des développements intéressants ont suivi. "J'ai enfin pu obtenir un signal propre et continu », explique Bitluni. « J'ai enfin pu partager quelques succès en direct". Il a même pu obtenir un signal de 1280×720 p et 8 bits (**figure 23**). Ce n'était pas du bruit, mais de très petits signaux carrés ! Un jour plus tard, il a retrouvé le sourire (**figure 24**). Ensuite, il a ajouté des bits supplémentaires, réorganisé les choses (**figure 25**), et commandé de nouvelles cartes. Avance rapide d'une semaine : **figure 26**, **figure 27**, et **figure 28**. Succès !

230529-04



bitluni
231K subscribers

Produits

- **Espressif ESP32-S3-EYE**
  www.elektor.fr/20626
- **D. Ibrahim, *The Complete ESP32 Projects Guide* (Elektor 2019)**
  www.elektor.fr/18860

Regarder la vidéo : I Made a VGA Card That Blew My Mind

Vous pouvez regarder une vidéo détaillée de ce projet sur la chaîne YouTube de Bitluni :

Scannez le code QR...

...et regardez la vidéo en RA.

ou

Regardez la vidéo sur YouTube

https://youtu.be/muuhgrige5Q

empreinte acoustique sur ESP32

identification de chansons avec le projet open source Olaf

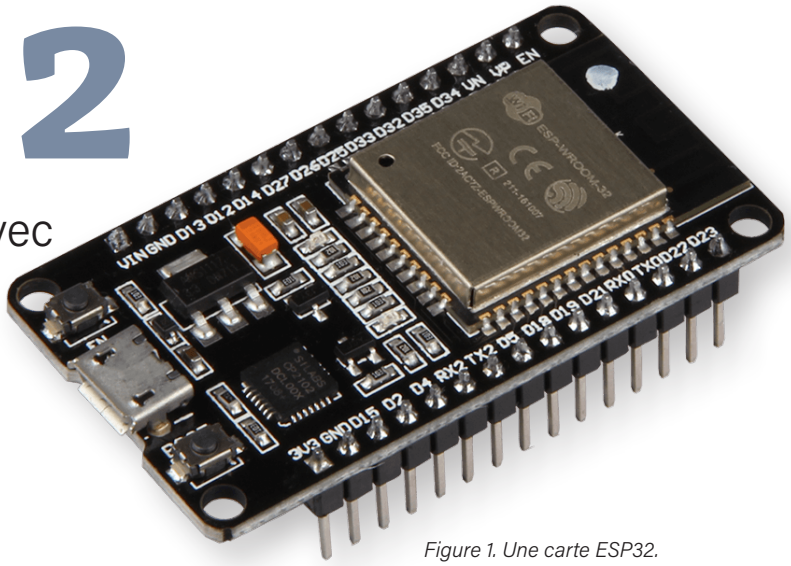


Figure 1. Une carte ESP32.

Joren Six, université de Gand (Belgique)

Il y a quelques années, Joren Six a été chargé de concevoir un dispositif d'informatique vestimentaire capable de reconnaître une chanson. Le code qu'il écrivit pour un module ESP32 donna naissance à Olaf, un projet multiplateforme et open source. Olaf, pour *Overly Lightweight Acoustic* Fingerprinting, est une bibliothèque d'identification musicale facile à utiliser et adaptée aux systèmes embarqués à mémoire et puissance de calcul limitées. Alors tous en rythme avec Olaf !

J'étais informaticien à l'université belge de Gand lorsque j'ai été chargé de mettre au point une technologie de reconnaissance audio pour un costume. Le cahier des charges impliquait de synchroniser les LED du costume sur le rythme d'une certaine chanson : seule cette chanson devait activer les LED, toute autre musique devant être ignorée. La reconnaissance et la synchronisation musicales sont généralement réalisées à l'aide de techniques d'empreintes audio. Le défi consistait à les implanter sur un microcontrôleur peu coûteux, disposant d'une puissance de traitement et d'une mémoire limitées. J'étais parvenu à créer un prototype fonctionnel, mais l'envie d'y revenir un jour ne me quitta plus vraiment. Nous étions en 2019.

L'occasion s'est présentée récemment, à l'approche du quatrième anniversaire de ma fille. Je me suis dit que je pourrais transformer cet ancien prototype en cadeau d'anniversaire extravagant : une robe « Elsa » qui réagit à la chanson *Libérée, délivrée* du film d'animation *La Reine des neiges* [1]. J'ai alors commandé une bande de LED RVB, une batterie Li-Ion, un microphone numérique I²S et, bien sûr, une robe Elsa. Je disposais déjà d'un microcontrôleur ESP32 (**figure 1**) et m'en suis donc servi. Il prend en charge I²S, comprend une unité à virgule flottante (FPU), dispose d'une mémoire vive suffisante, et est facile à relier à une bande de LED. L'unité FPU permet d'éviter les calculs en virgule fixe et facilite ainsi la réutilisation du code à la fois sur PC et sur dispositifs embarqués.

Assemblage de la version à ESP32

La version initiale du projet reposait sur une carte *ESP32 Thing* de Sparkfun – choisie pour son connecteur de batterie pratique – à laquelle était relié un microphone de type MEMS (un I*NMP441*) et une bande de LED adressables (**figure 2**) – un modèle générique, facilement trouvable sur eBay ou AliExpress.

Le câblage de ces éléments relève de la routine. Pour le microphone, il s'agit d'en relier les broches d'E/S *SDA*, *SCK* et *WS* à des broches appropriées du ESP32, par exemple les broches GPIO 32, 33 et 35. Sans oublier, bien sûr, l'alimentation du microphone.

Côté bande de LED, le câblage dépend du modèle utilisé. Pour un ruban à LED *WS2812B*, trois fils de connexion sont nécessaires (alimentation, masse, données). Je me suis appuyé sur la bibliothèque *FastLED* pour l'écriture du code, donc assurez-vous que votre ruban à LED est pris en charge par cette bibliothèque si vous souhaitez reproduire ce projet. L'ensemble une fois soudé et en place, restait à coudre la bande de LED sur le costume, tâche que je laissai aux mains expérimentées de ma meilleure moitié. La vidéo du lien [1] montre le résultat final.

Figure 2. Ruban à LED RGB.

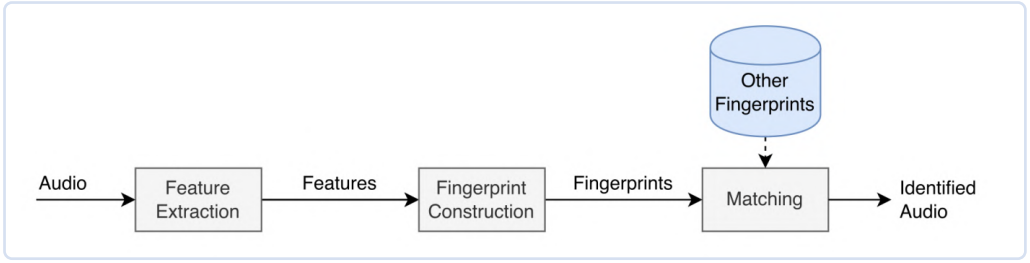


Figure 3. Les étapes de l'identification par empreintes acoustiques.

La première chanson n'est pas reconnue et n'active donc pas les LED, mais ces dernières entament bien leur sarabande lumineuse lorsque *Let it go* (*Libérée, délivrée*) se fait entendre. Les LED restent allumées un bref instant lorsque la chanson est mise sur pause, une caractéristique choisie à dessein pour tenir compte des « trous » durant la reconnaissance. La chanson est correctement identifiée lorsqu'elle reprend. Vous trouverez sur [2] les actualités de ce projet.
Bien entendu, comme souvent avec les projets à microcontrôleur, c'est du code que provient toute la magie.

Comment ça marche ?

Comment apprendre à un ordinateur à identifier une chanson ou de la musique ? Plusieurs techniques sont possibles. L'une d'elles, courante et sur laquelle reposent Olaf et le service d'identification musicale *Shazam*, est la reconnaissance par crête spectrale.
L'idée est simple : l'appli transforme le fichier audio enregistré sur le téléphone en un format qu'un processeur saura aisément comparer à d'autres chansons (**figure 3**). Ce processus repose sur l'empreinte acoustique de la chanson, c'est-à-dire sur un condensé numérique de ladite chanson en une signature unique, identifiable même parmi un bruit de fond sonore intense.
Une telle signature repose sur le spectrogramme de la chanson, autrement dit sur son diagramme temps-fréquence. Un spectrogramme indique aussi le niveau de puissance du signal audio au moyen de couleurs. Cette puissance reflète le niveau sonore perçu par l'oreille. Toutefois, si un service comme *Shazam* comparait directement plusieurs spectrogrammes entre eux pour identifier une chanson, le résultat n'arriverait que bien après la fin de celle-ci. Une meilleure approche consiste à comparer les pics du spectrogramme puisque ces pics contiennent des informations essentielles, y compris pour le cerveau humain.
Olaf enregistre donc un échantillon sonore, en calcule le spectrogramme, puis simplifie celui-ci en un diagramme de dispersion des pics de fréquence – surlignés par des points verts sur le spectrogramme de la **figure 4**.
Ces diagrammes de dispersion, qui pour l'essentiel représentent les signaux les plus caractéristiques du spectrogramme, doivent ensuite être comparés aux entrées d'une base de données contenant de nombreuses chansons ou, dans le cas du costume d'Elsa, être comparés à une seule chanson.
Plutôt que de rechercher dans une base de données de chansons la suite de points correspondante, il s'avère plus astucieux de relier par paires les pics les plus proches. Si en parcourant la base de données l'algorithme trouve suffisamment de paires correspondantes ayant un alignement temporel identique, Olaf (ou *Shazam*) peut identifier la chanson.

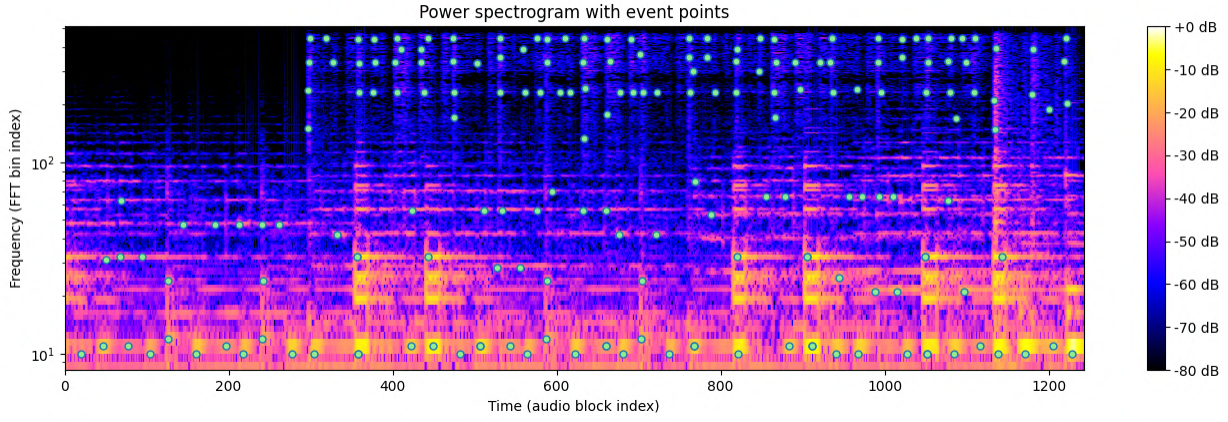


Figure 4. Un spectrogramme représente les fréquences sur un intervalle de temps donné. Les points verts sont extraits par Olaf et indiquent les pics de fréquence.

Évolution du projet

Le code du prototype initial de 2019 n'était pas très bien organisé, mais celui de la seconde version était bien plus clair, au point qu'il ne me parut pas déraisonnable de le partager sur GitHub. C'est à ce moment-là que j'ai baptisé le projet Olaf, pour *Overly Lightweight Acoustic Fingerprinting* [3].

J'ai progressivement étoffé le code pour l'amener à faire bien plus que son objectif premier. Le projet est ainsi devenu un système d'empreinte acoustique polyvalent, paré pour de multiples applications. Olaf fait montre de performances impressionnantes, grâce notamment à une exploitation frugale des ressources matérielles et à son recours à la bibliothèque *PFFFT*.

Le nom de cette bibliothèque ne vous dit sans doute pas grand-chose, surtout si, comme Bob Pease, votre langage de programmation préféré est le fer à souder ! PFFFT [4] est l'acronyme de *Pretty Fast FFT*, une bibliothèque conçue pour être plus légère que la renommée *FFTW*. Elle est de fait compacte et rapide, ce qui la rend idéale pour le ESP32. Sur un dispositif embarqué, les empreintes de référence sont stockées en mémoire, ce qui élimine le besoin d'une base de données. Sur un ordinateur classique, ces empreintes sont stockées dans une base de données très performante de type *LMDB* – la décrire en détail nous entraînerait trop loin, je vous encourage à en découvrir vous-même les avantages.

À l'origine simple gadget, Olaf est devenu une application/bibliothèque complète pour l'extraction, le stockage et la recherche d'empreintes acoustiques. Olaf sait extraire ces empreintes d'un flux audio de façon efficace, les stocker dans une base de données et, comme expliqué plus haut, rechercher leur correspondance à partir de paires de pics (technique appelée *landmark-based acoustic fingerprinting*).

J'ai aussi écrit Olaf car il ne semblait pas y avoir beaucoup de bibliothèques semblables, à la fois légères, faciles à déployer sur un système embarqué et ne requérant guère de mémoire et de ressources de calcul. Olaf est écrit en C et se veut aussi portable que possible. Il vise principalement les dispositifs à ARM 32 bits (comme divers modèles *Teensy*), certaines cartes Arduino et le module ESP32. D'autres plateformes de spécifications similaires pourraient être compatibles. Nul doute qu'Olaf, qui est open source, stimulera la création de projets innovants combinant reconnaissance musicale/sonore et dispositifs pour l'IdO !

Olaf fonctionne également sur PC, et rapidement. Le code peut être compilé et exécuté localement sur une machine, mais aussi être lancé depuis un navigateur web ! L'encadré **Exécuter Olaf depuis un navigateur web** explique comment procéder.

Le code source est sur GitHub, accompagné d'un exemple pour ESP32 et de petits outils de débogage écrits par mes soins. L'archive du lien [6] contient aussi un fichier de projet *PlatformIO*.

Déployer Olaf sur un ESP32

Lors du développement du projet, Olaf avait besoin d'un minimum de RAM pour le traitement audio, ce qui posait souvent problème avec de nombreux microcontrôleurs. La version PC à stockage clé-valeur nécessitait 512 Ko de RAM, alors que la version embarquée n'exigeait que 200 Ko avec le module ESP32. Comme je ne travaille pas souvent avec des microcontrôleurs, il me fallait un environnement de développement intégré facile à configurer et à utiliser. *PlatformIO* et l'EDI *Arduino* ont été à cet égard parfaits pour un utilisateur comme moi. Ils n'accaparaient pas mon processeur et reconnaissaient bon nombre de microcontrôleurs (dont Teensy 3+, RP2040 et les Cortex-M). Une unité FPU peut par ailleurs réduire la consommation d'énergie, un point essentiel lorsqu'on travaille sur batterie.

Je me suis aussi servi de quelques modules *M5StickC* de M5Stack, car ils sont faciles à programmer et ont un microphone intégré – donc parfaits pour une personne comme moi plus intéressée par le logiciel que le matériel, même si je me suis déjà attaqué à quelques projets de domotique (voir mon site web), dont une commande de ventilation et la surveillance du niveau d'un réservoir d'eau de pluie.

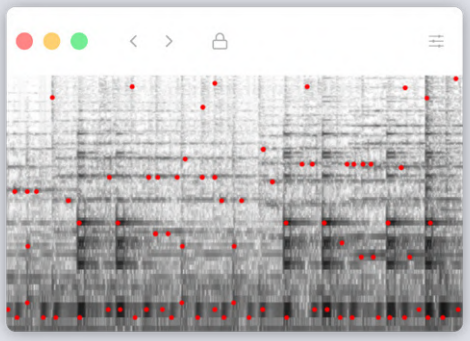
J'ai ajouté pour cet article un code de démo ESP32 à mon dépôt GitHub [7] et complété la documentation. La dernière version d'Olaf donne des résultats fiables sur ESP32 et avec un microphone MEMS – facile à se procurer. J'explique sur mon blog [6] comment utiliser des microphones I²S comme le *INMP441*.

Exécuter Olaf depuis un navigateur

Connaissez-vous WebAssembly, ou WASM [5] ? WASM est un format de code binaire portable, exécutable par un navigateur web. Sa spécification décrit aussi sa représentation au format texte. Son principal objectif est de faciliter le développement d'applications performantes pour le web.

Du code écrit en C peut être compilé en WebAssembly avec Emscripten. Ce compilateur permet, dixit sa documentation, d'exécuter du code C/C++ sur une page web à une vitesse presque native, sans recourir à des extensions. Combiner l'API Web Audio et la version WASM d'Olaf ouvre la voie à des applications d'empreintes acoustiques pour l'internet.

Sur mon blog, vous pouvez tester Olaf depuis votre navigateur – et lire mes derniers articles à ce sujet. Le code qui s'exécute sur l'ESP32 s'exécutera dans votre navigateur, autrement dit Olaf tentera d'identifier la chanson Let It Go de la bande originale de Frozen (La Reine des neiges). Lancez la vidéo YouTube embarquée, puis lancez Olaf (à droite de la vidéo). Olaf analysera le flux audio du microphone de votre PC, calculera sa transformée de Fourier rapide (FFT) et la visualisera avec Pixi.js. Les empreintes rouges (cf. capture d'écran) devraient devenir vertes au bout de quelques secondes si l'identification a réussi. Elles repasseront au rouge lorsque vous stopperez la chanson. Comme dans la démo vidéo mentionnée dans l'article, la transition visuelle entre correspondance et non-correspondance est temporisée pour tenir compte des « trous » durant la reconnaissance.



Le dépôt GitHub contient un programme de démonstration qui transmet le son du microphone à un PC par Wi-Fi. Il permet de s'assurer que le microphone fonctionne et que les échantillons audio sont bien interprétés. Taille des tampons, fréquences d'échantillonnage, formats audio et autres paramètres stéréo et mono sont conformes au protocole I²S. Un autre code d'exemple montre comment se fait la correspondance entre l'audio d'un microphone *INMP441* et les empreintes acoustiques. Contrairement à la version PC qui repose sur une base *LMDB* à stockage clé-valeur, la version embarquée d'Olaf utilise une liste de hachages stockée dans le fichier d'en-tête *src/olaf_fp_ref_mem.h*. Ce fichier est l'index des empreintes acoustiques.

Le fichier d'en-tête *src/olaf_fp_ref_mem.h* est par défaut inclus dans le code de l'ESP32. Pour le tester et le déboguer, utilisez la version *mem* d'Olaf sur votre ordinateur : `bin/olaf query olaf_audio_your_audio_file.raw "arandomidentifier"`. La version ESP32 est identique à la version *mem*, si ce n'est que l'audio provient de l'entrée d'un microphone MEMS et non d'un fichier.

Une fois testés avec succès la version *mem* d'Olaf et le microphone *INMP441*, vous pouvez déployer la version ESP32 sur le microcontrôleur à l'aide de l'EDI *Arduino*.

Figure 5. Le costume alimenté par ESP32.

Olaf et vous

J'espère que cet article suscitera en vous des idées de projets reposant sur l'identification acoustique : savoir que l'amélioration progressive d'un petit projet au départ purement ludique peut vite conduire à des résultats impressionnants est toujours motivant. Olaf est ainsi passé d'un premier prototype qui ne répondait guère à mes attentes à un second prototype à ESP32 largement amélioré, beaucoup plus satisfaisant pour moi et… ma fille (**figure 5**).

Olaf est devenu au fil du temps et d'itérations du code une application (et une bibliothèque) complète, open source, portable, multi-plateforme et très performante. J'ai écrit deux articles universitaires sur le sujet afin qu'Olaf soit reconnu et apprécié par la communauté des chercheurs. Et vous, lecteurs et lectrices : quels projets passionnants avez-vous en tête pour votre ESP32 ?

VF : Hervé Moreau — 230578-04

Questions ou commentaires ?

Contactez l'auteur (joren.six@ugent.be) ou Elektor (redaction@elektor.fr).

À propos de l'auteur

Joren Six est informaticien et chercheur en informatique musicale, recherche d'informations musicales et ethnomusicologie informatique. Il est titulaire d'un doctorat de l'université de Gand (Belgique) et travaille actuellement sur plusieurs projets mêlant informatique et acoustique.

Produits

- **Carte de développment ESP32 JOY-iT NodeMCU**
  www.elektor.fr/19973
- **Carte ESP32-DevKitC-32E**
  www.elektor.fr/20518
- **Carte ESP32-C3-DevKitM-1**
  www.elektor.fr/20324

LIENS

[1] Premier prototype du projet : https://0110.be/posts/Olaf_-_Acoustic_fingerprinting_on_the_ESP32_and_in_the_Browser
[2] Actualités du projet Olaf : https://0110.be/search?q=olaf
[3] Dépôt du projet Olaf : https://github.com/JorenSix/Olaf
[4] Bibliothèque Pretty Fast FFT : https://bitbucket.org/jpommier/pffft/src/master/
[5] WebAssembly sur Wikipédia : https://en.wikipedia.org/wiki/WebAssembly
[6] Fichiers à télécharger : https://0110.be/files/attachments/475/ESP32-Olaf.zip
[7] Version ESP32 d'Olaf sur GitHub : https://github.com/JorenSix/Olaf/tree/master/ESP32

arbre de Noël circulaire 2023

célébration high-tech des fêtes de fin d'année

Ton Giesberts (Elektor)

La WS2812D-F8 est une LED RVB de 8 mm de haute technologie, programmable numériquement et adressable individuellement avec jusqu'à 255 dispositifs en guirlande. Dans ce projet de sapin de Noël circulaire en 3D, 36 de ces dispositifs peuvent être commandés de l'extérieur ou par un Arduino Nano ESP32 intégré. Les effets de lumière sont étonnants – ne manquez pas d'avoir ce kit lumineux d'Elektor pour les fêtes de fin d'année !

Cet arbre de Noël se distingue par la forme de sa construction et l'utilisation de LED RVB numériques de 8 mm. Il ressemble plus à un arbre réel que la plupart des circuits imprimés (PCB) plats avec un contour en forme de sapin. Au lieu d'une simple conception en 2D, il s'agit d'une véritable création en 3D, basée sur les deux versions précédentes [1] et [2]. Le nouvel arbre est réalisé en séparant les cinq sections annulaires concentriques du PCB fourni avec le kit, et en les empilant avec des fils rigides pour espacer les cartes, donnant à ce projet l'aspect typique d'un arbre en forme de conifère.

Le projet [1] utilisait un générateur de nombres aléatoires simple avec deux circuits intégrés logiques standard et de petites LED CMS blanches. Le projet [2] reposait sur un microcontrôleur avec des LED connectées en matrice 6×6. Cette nouvelle version utilise 36 LED RVB numériques de 8 mm de type WS2812D-F8. Cela simplifie le circuit. Au lieu d'une matrice, les entrées et sorties des LED numériques sont connectées en série, chaque LED étant reliée à une alimentation en 5 V. La chaîne de LED peut être adressée comme une bande de LED NeoPixel. De nombreux programmes sur le web, dans de multiples langages, expliquent en détail comment gérer ces dispositifs.
Le connecteur K1 est relié au +5 V (4,3 V) interne, à la masse et à l'entrée de la première LED (par le biais du cavalier JP1). De cette manière, différents microprocesseurs, microcontrôleurs ou modules externes peuvent être utilisés pour commander les LED de notre arbre de Noël circulaire. Pour rendre l'arbre indépendant d'un circuit externe, on peut installer un module Arduino Nano ESP32 [3] sur le PCB de base.

Schéma

Le circuit se compose essentiellement de 36 LED RVB numériques de type WS2812D-F8, dont les broches d'entrée (D_{in}) et de sortie (D_{out}) sont connectées en série, chacune d'entre elles étant alimentée par une alimentation de 4,3 V (**figure 1**). La tension réelle sur les LED est plus faible en raison de la chute causée par les diodes de protection D1 et D2. Les LED peuvent être commandées par un microprocesseur externe, un microcontrôleur, un module ou un module Arduino Nano ESP32 optionnel (MOD1).

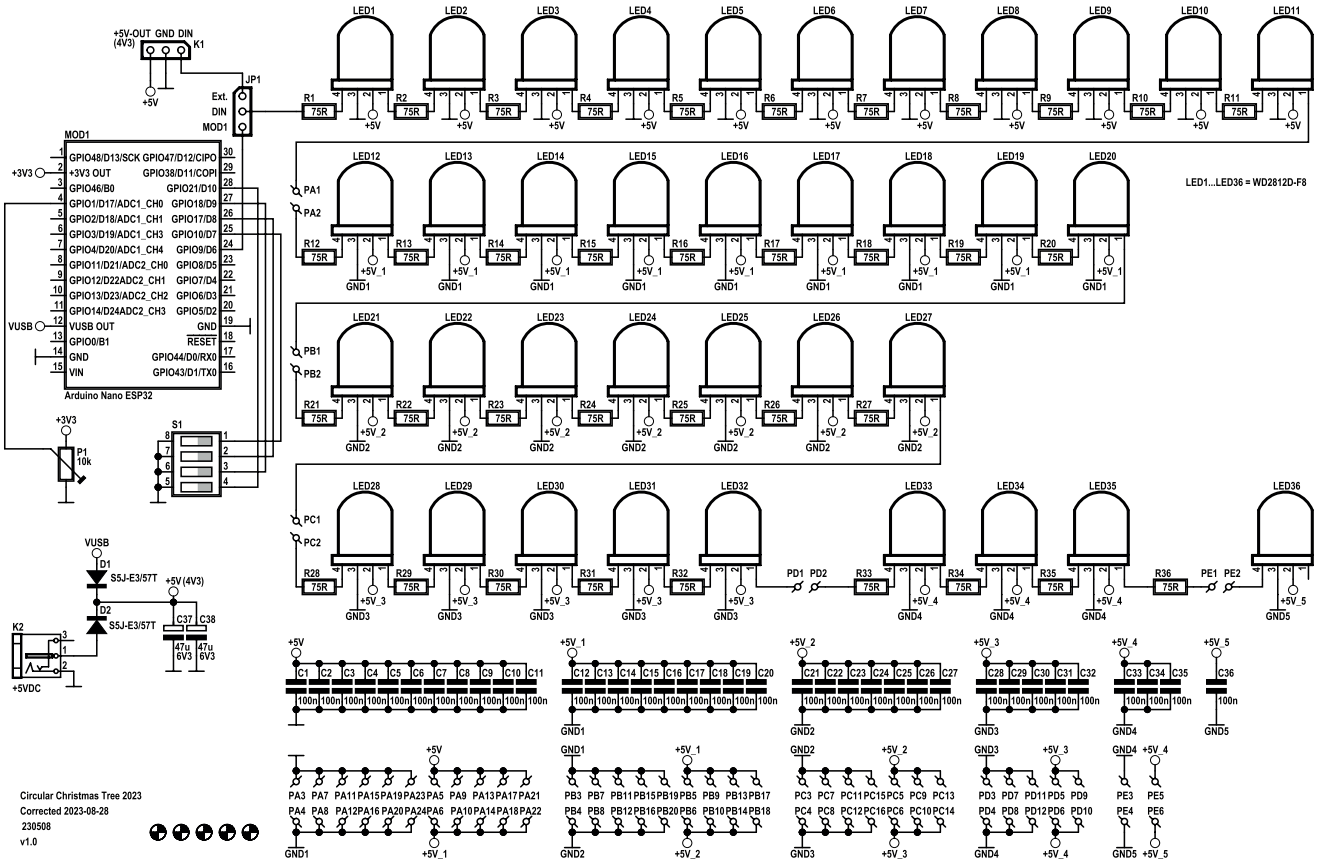


Figure 1. Schéma de l'arbre de Noël circulaire.

Le cavalier JP1, visible sur la **figure 2** près de la carte Arduino Nano, sélectionne le signal de commande de la LED1 (qui est la première des 36 LED en guirlande). L'utilisation de l'arbre est plus souple avec une carte Arduino Nano ESP32. On peut, par logiciel, modifier à distance les motifs d'éclairage de l'arbre en utilisant le Wi-Fi et/ou le Bluetooth LE intégrés. Toutes les entrées des LED ont une résistance de 75 Ω en série (R1...R36, selon les exigences indiquées dans la fiche technique des WS2812D-F8 [4]). Chaque LED est découplée par un condensateur de 100 nF (C1...C36). De plus, les condensateurs au tantale de 47 µF C37 et C38 découplent l'alimentation des LED sur le PCB de base.

Tous ces composants sont des CMS et sont montés sur la face inférieure des PCB, donc invisibles. On peut alimenter les LED avec un adaptateur 5 VDC, de préférence d'au moins 1,5 A, via le connecteur d'alimentation CC K2, ou la broche VBUS de la carte Arduino Nano (si elle est présente) dans la zone du PCB de MOD1. Les diodes D1 et D2 (5 A, 50 VDC, S5J-E3/57T, boîtier CMS, taille SMC), également placées au-dessous du PCB, évitent les dommages si les deux alimentations sont connectées simultanément. On peut néanmoins connecter une alimentation à la fois au MOD1 et à K2, mais ce n'est pas nécessaire. Le module lui-même est alimenté par son connecteur USB-C, et le courant maximal de la broche VBUS est suffisant (en supposant que l'adaptateur secteur connecté soit assez puissant, bien sûr).

Les diodes ne sont volontairement pas de type Schottky. La chute de tension à travers les diodes est plus élevée et il faut s'assurer que les signaux logiques de 3,3 V sont bien dans la spécification des WS2812D-F8. Pour cette raison, si la chaîne de LED est commandée par un circuit externe, la meilleure option est toujours d'alimenter ce

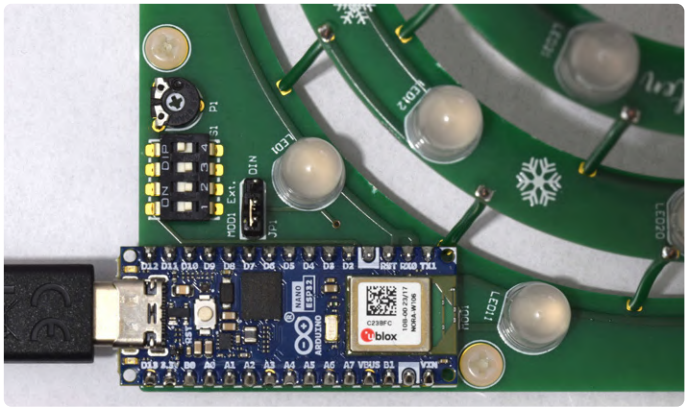


Figure 2. Détail de MOD1, S1 et P1 montés sur le PCB de base.

circuit par la source d'alimentation interne de K1. La tension réelle sur K1 sera inférieure aux 5 V de K2 - autour de 4,3 V (ou même plus bas, avec un réglage élevé du courant/luminosité) - à cause de la chute de tension à travers D2.

Pour connecter les PCB, 39 morceaux de fils relient le +5 V et la masse d'un PCB à celui qui se trouve au-dessus (**figure 3)**. Dans le schéma, il s'agit des connexions PA1-PA2 à PE5-PE6. Un fil supplémentaire sur chaque PCB relie la broche D_{out} de la dernière LED à la première LED du PCB au-dessus.

Figure 3. Un fil marqué et un autre dénudé. L'isolant restant doit avoir une longueur de 3 cm.

Si on utilise un Arduino Nano ESP32 pour commander les LED, on peut aussi installer sur le PCB le commutateur DIP S1 et le petit potentiomètre P1, tous deux visibles sur la **figure 2**. Le logiciel peut lire ces composants pour sélectionner différents motifs ou ajuster la luminosité. Ces composants sont inutiles sans le module.

Alimentation électrique

L'alimentation des LED est assurée par un adaptateur CA vers +5 V CC avec un connecteur en barillet, connecté à K2. Le kit disponible dans notre boutique ne contient que les composants pour le PCB, y compris le petit potentiomètre et les interrupteurs DIP. L'adaptateur secteur et le module optionnel Arduino Nano ESP32 ne sont pas inclus et doivent être achetés séparément.

Ce module dispose d'un connecteur USB-C qui alimente également les LED via sa broche VBUS. Comme mentionné ci-dessus, deux diodes (D1, D2) empêchent que les deux alimentations soient connectées directement l'une à l'autre. VBUS est connecté en interne au 5 V du connecteur USB-C par l'intermédiaire d'une diode Schottky de type PMEG6020AELR (Nexperia). Cette diode est prévue pour 2 A. Actuellement, le courant maximal de la broche VBUS n'est pas spécifié dans sa fiche technique [5]. Le courant des WS2812D-F8 est de 12 mA, selon une vue d'ensemble de cette famille sur le site web du fabricant.

Cependant, la fiche technique indique 15 mA dans ses spécifications, et de nombreux paramètres sont évalués pour I_F = 20 mA. Dans notre prototype, le courant maximum au réglage maximum (3×255 en décimal par LED) était d'un peu moins de 11 mA par couleur – loin de 20 mA. Ces différences par rapport aux spécifications sont un peu déroutantes. Apparemment, une valeur plus réaliste pour le courant est d'environ 12 mA. En supposant que cette valeur soit correcte, le courant maximal à la luminosité maximale des LED serait de :

36 (DEL) × 3 (couleurs) × 12 mA = 1,3 A.

Dans notre prototype de sapin de Noël, le courant maximal total mesuré est de 1,156 A. La broche VBUS du MOD1 ne devrait pas avoir de problème avec ce courant. Mais, en général, nous vous conseillons vivement de ne jamais utiliser le courant nominal maximum d'une LED, car cela réduit considérablement sa durée de vie ! La luminosité entre, par exemple, un tiers et le courant nominal maximum n'est pas beaucoup plus faible. Pensez à en tenir compte lors de l'écriture du logiciel. Régler toutes les LED pour qu'elles produisent une lumière blanche, pour un éclairage très intense dans une pièce bien éclairée, nécessitera un courant global d'environ 200 mA. Cela sera plus que suffisant dans la plupart des cas et vous permettra d'utiliser n'importe quel port USB standard de type A pour alimenter le module.

PCB

Comme prévu, le PCB de 136×136 mm est un panneau, le PCB de base contenant les cinq PCB de forme circulaire avec 24 ponts à casser. La carte entière figure dans l'encadré de la **liste des composants**, tandis que la **figure 4** montre toutes les parties séparées de cette carte, après avoir cassé les ponts. Sachez qu'il faut un peu forcer pour les casser. Un manuel de construction, disponible sur le site d'Elektor Labs [6], donne tous les détails sur la façon de construire l'arbre. Le cuivre des 39 morceaux de fil fait 0,8 mm d'épaisseur. Ces fils contribuent aussi à l'apparence de l'arbre, et c'est pourquoi nous avons choisi du fil isolé *vert.* Mais si vous trouvez qu'un fil nu étamé est plus joli, vous pouvez bien sûr enlever tout l'isolant.

Veillez à ce que tous les PCB soient fixés parallèles, avec un espace de 3 cm entre chaque carte. Le grand nombre de fils rend la construction assez rigide, comme on peut le voir sur la **figure 5**. Cette figure montre le prototype terminé sans le module supplémentaire. Pour que la chute de tension entre le bas et le haut soit minimale, tous les fils, à l'exception du fil de signal, sont connectés alternativement au +5 V et à la masse. Si vous ôtez tout l'isolant, prenez garde aux courts-circuits entre les fils dénudés !

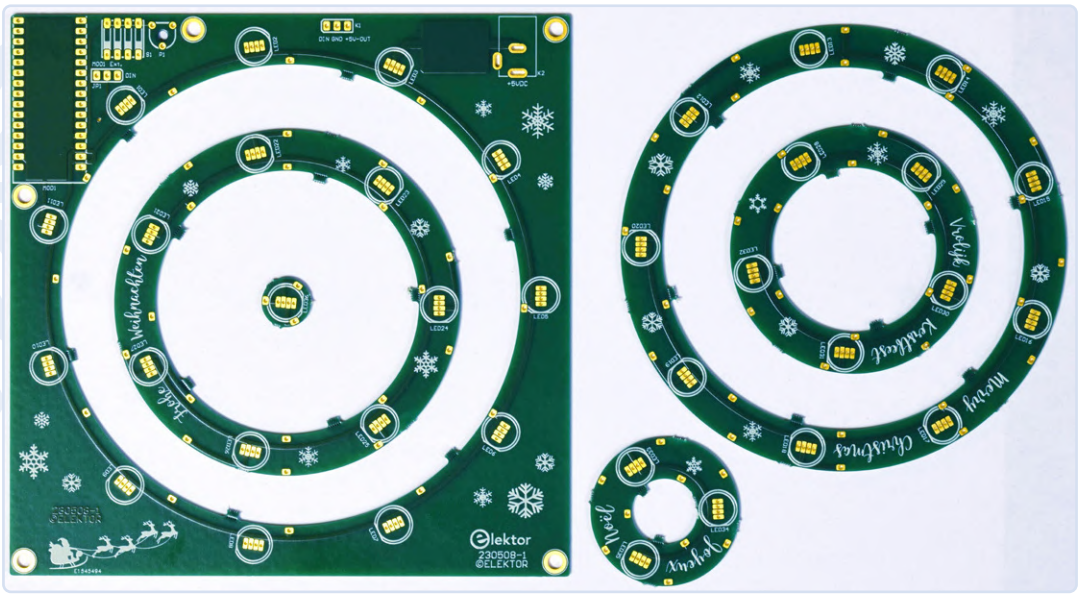
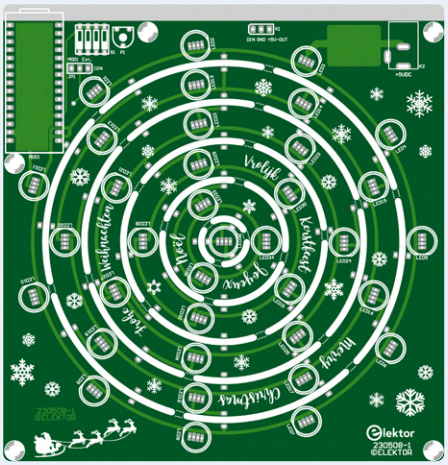
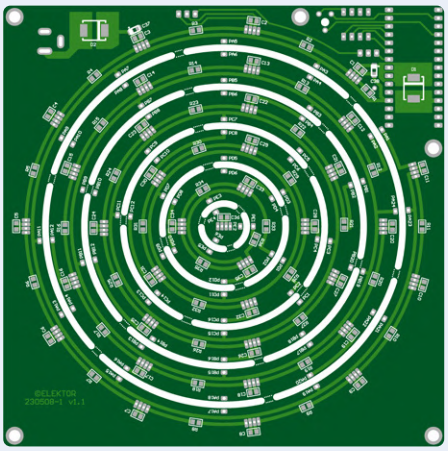


Figure 4. Tous les PCB séparés. Les ponts à casser qui reliaient les 6 PCB peuvent maintenant être retirés à l'aide d'une pince coupante.

LISTE DES COMPOSANTS





Résistances

R1…R36 = 75 Ω, 0W125, 5%, SMD 0805
P1 = potentiomètre circulaire 6 mm réglable par le dessus, 10 kΩ, 0W1, 20%, (Piher PT6KV-103A2020)

Condensateurs

C1…C36 = 100 nF, 50 V, 5%, X7R, SMD 0805
C37, C38 = 47uF, 6,3 V, 10%, tantale, taille de boîtier A (1206)

Semi-conducteurs

D1, D2 = S5J-E3/57T, taille de boîtier SMD SMC
LED1-LED36 = WS2812D-F8, 8 mm, THT
MOD1 (optionnel) = Arduino Nano ESP32 avec barrettes

Autres

K1, JP1 = barrettes, 3×1, vertical, pas de 2,54 mm
Cavalier de shunt pour JP1, pas de 2,54 mm
K2 = MJ-179PH (Multicomp Pro), connecteur d'alimentation CC, 4 A, diamètre des broches 1,95 mm
S1 = Interrupteur DIP, 4 voies
PA1…PE6 = 2 m de fil, 0,81 mm rigide, 0,52 mm2 / 20AWG, isolé vert (Alpha Wire 3053/1 GR005)
H1…H5 = Entretoise en nylon, femelle-femelle, M3, 5 mm
H1…H5 = Vis en nylon, M3, 5 mm

Divers

PCB 230508-1 (136 × 136 mm)

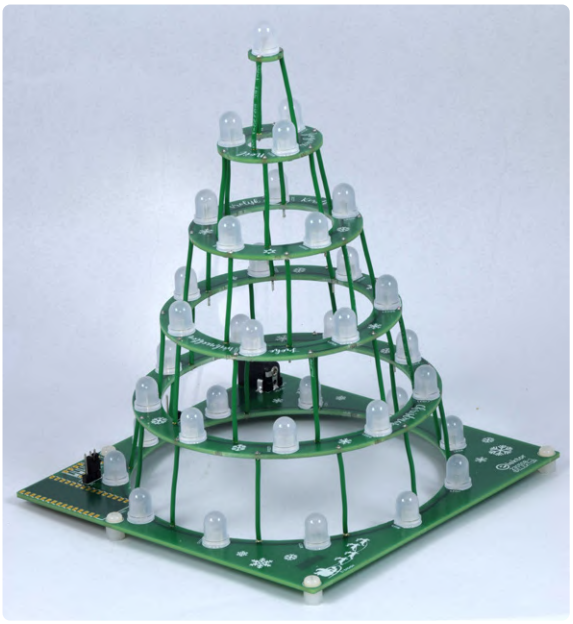


Figure 5. Le prototype terminé, sans le module Arduino optionnel.

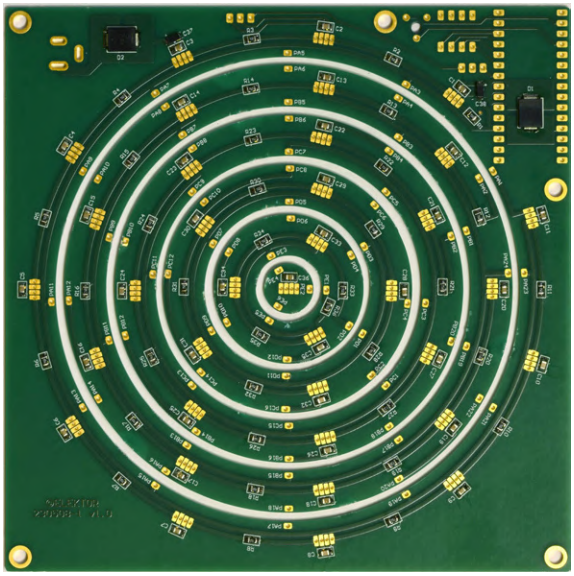


Figure 6. Tous les CMS sont soudés sur les faces arrière des PCB : R1…R36, C1…C38, D1 et D2.

Les pistes pour l'alimentation électrique font 1 mm de large. Tous les composants CMS sont montés sur la face inférieure des PCB, comme le montre la **figure 6**. Il faut séparer les PCB avant de commencer à souder. Il n'y a aucun autre composant sur le dessus des PCB circulaires, à l'exception des LED, dont les pattes doivent être coupées comme le montre la **figure 7**. Les composants à trous traversants JP1, K1, K2, P1, S1 et MOD1 sont placés sur le PCB carré de base. Comme ce projet utilise des composants CMS, il vaut mieux avoir une certaine expérience du soudage de ces composants.
Les résistances et condensateurs 0805 ne sont pas difficiles à souder à l'aide d'un fer à souder à panne fine et de la soudure fine. Une soudure très fine, d'un diamètre de 0,35 mm est recommandée, afin de ne pas utiliser trop d'étain pour les joints de soudure. L'arbre est placé sur cinq supports en nylon blanc de 5 mm fixés par cinq vis en nylon blanc. Ces vis sont moins visibles. Le module optionnel MOD1

Figure 7. Les fils de la LED RVB WS2812D-F8 sont coupés juste au-dessus de l'élargissement.

est placé dans un coin du PCB pour que la base soit aussi petite que possible. P1 et S1 sont placés à côté du module.

Module optionnel Arduino Nano ESP32

Pour que cet arbre de Noël fonctionne indépendamment de tout circuit externe, on peut placer le module Arduino Nano ESP32 (MOD1) sur le PCB de base ainsi que le petit potentiomètre P1 et les commutateurs DIP S1. Placez le cavalier JP1 en position MOD1. Les deux composants supplémentaires peuvent être utilisés pour ajuster et/ou sélectionner la luminosité/vitesse et différents motifs/modes.

La programmation du module peut se faire avec la dernière version de l'EDI Arduino et le langage de programmation C. Si vous cherchez un projet MicroPython, cet arbre de Noël est également un bon choix. Un EDI dédié appelé Arduino Lab pour MicroPython [7] peut être téléchargé depuis GitHub. Un installateur MicroPython [8] est également nécessaire, et également disponible sur GitHub. Tout cela est bien documenté sur le site web d'Arduino et donne aux débutants de bonnes ressources pour le faire fonctionner [9].

Elektor a développé un logiciel simple téléchargeable à partir de la page web Elektor Labs de ce projet [6], où vous trouverez également des détails sur ses fonctionnalités.

Pendant le développement du logiciel, vous pouvez alimenter le projet à partir du connecteur USB d'un PC, mais un adaptateur CA-CC avec un connecteur UCB-C sera nécessaire pour alimenter l'arbre une fois terminé. Le courant maximum disponible dépend, bien sûr, du port spécifique. Grâce à un adaptateur USB-C vers USB-A ou à un câble adaptateur, l'Arduino Nano peut également être connecté à un port USB « ancien » si la luminosité est maintenue à un faible niveau. Le courant est alors limité à 500 mA, à retenir lors des tests du logiciel ! Vous pouvez également connecter une alimentation à K2 et régler la tension à un peu plus de 5 V afin d'alimenter les LED. La **figure 8** ne montre pas vraiment les couleurs vives des LED, mais donne une bonne impression de ce qui peut être obtenu avec différents réglages.

VF : Denis Lafourcade — 230665-04

À propos de l'auteur

Ton Giesberts a commencé à travailler chez Elektuur (aujourd'hui Elektor) après ses études, alors qu'on recherchait une personne ayant des affinités avec l'audio. Au fil des ans, il a surtout travaillé sur des projets audios. La conception analogique a toujours eu sa préférence. Bien entendu, les projets dans d'autres domaines de l'électronique font également partie de son travail. L'une des devises de Ton est la suivante : « Si vous voulez que ce soit mieux fait, faites-le vous-même ». Par exemple, pour la conception d'un PCB destiné à un projet audio avec des taux de distorsion de l'ordre de 0,001%, une bonne disposition est cruciale !

Questions ou commentaires ?

Contactez Elektor (redaction@elektor.fr).

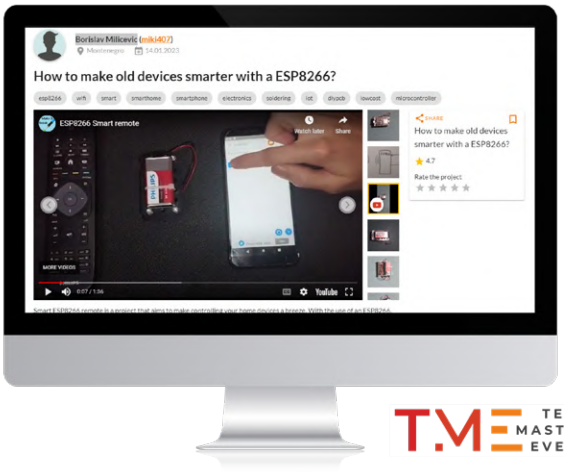
Produits

- **Kit sapin de Noël circulaire**
  www.elektor.fr/20672
- **Arduino Nano ESP32 avec connecteurs**
  www.elektor.fr/20529

Figure 8. Configuration de test de l'arbre de Noël.

LIENS

[1] Arbre de Noël V1 : https://elektormagazine.fr/labs/130478-xmas-tree-2014
[2] Arbre de Noël V2 : https://elektormagazine.fr/labs/circular-christmas-tree-150453
[3] Documentation sur l'Arduino Nano ESP32 : https://docs.arduino.cc/hardware/nano-esp32
[4] Fiche technique WS2812D-F8 [PDF] : https://soldered.com/productdata/2021/03/Soldered_WS2812D-F8_datasheet.pdf
[5] Fiche technique Arduino Nano ESP32 [PDF] : https://docs.arduino.cc/resources/datasheets/ABX00083-datasheet.pdf
[6] Ce projet sur Elektor Labs : https://elektormagazine.fr/labs/circular-christmas-tree-2023-230508
[7] Arduino Lab pour Windows [Zip] : https://tinyurl.com/arduinolab4win
[8] Arduino MicroPython Installer : https://github.com/arduino/lab-micropython-installer/releases/tag/v1.2.1
[9] Cours MicroPython - MicroPython 101 par Arduino : https://docs.arduino.cc/micropython-course/



une vie plus confortable et plus facile

un projet amateur basé sur le module ESP8266 Espressif

Contribué par Transfer Multisort Elektronik Sp. z.o.o.

L'idée de SmartHome devient de plus en plus populaire chaque année et les solutions qui nous aident à gérer plus efficacement notre espace de vie deviennent progressivement plus disponibles. De plus, il existe également sur le marché des produits qui permettent d'adapter des appareils encore plus anciens aux évolutions technologiques. Le contrôle à distance des appareils domestiques et l'automatisation de nombreux processus permettent d'augmenter l'efficacité énergétique, de prendre soin de l'environnement, d'augmenter le confort et de réduire les coûts. Ceci est confirmé par le projet Smart ESP8266 remote développé dans le cadre du concours de plateforme TechMasterEvent.

Espressif est un fabricant de systèmes SoC et de modules de transmission sans fil populaires qui sont disponibles chez TME. Grâce à leur taille compacte et leur faible consommation d'énergie, les produits Espressif peuvent être utilisés avec succès aussi bien dans l'électronique grand public que dans les systèmes industriels.

Ci-dessous, nous présentons une description de l'appareil basé sur le module ESP8266. Il s'agit d'un projet amateur créé par un participant au concours de la plateforme TechMasterEvent. La tâche du concours était de créer des projets électroniques qui facilitent la vie. Le module ESP8266 et de nombreux autres éléments utiles dans les projets IdO (ordinateurs monocartes, modules de communication et modules de mémoire, écrans et bien d'autres) sont disponibles sur [1].

ESP8266, LED IR et récepteur IR

L'objectif du projet Smart ESP8266 remote était de faciliter le fonctionnement des appareils que l'on retrouve dans chaque maison. Grâce à l'utilisation de la puce ESP8266, d'une diode émettrice et d'un récepteur infrarouge, ce projet permet d'éviter l'utilisation de plusieurs télécommandes pour faire fonctionner divers appareils tels que les climatiseurs et les téléviseurs. Smart ESP8266 remote se connecte à une application téléphonique, ce qui facilite l'utilisation des appareils par les utilisateurs et leur permet d'enregistrer les signaux envoyés par leurs télécommandes actuelles pour une utilisation ultérieure.

En plus de sa commodité et de sa facilité d'utilisation, le projet Smart ESP8266 remote constitue donc une excellente solution pour les appareils plus anciens qui ne sont pas compatibles avec la technologie traditionnelle de maison intelligente. Grâce à la fonction de lecture et d'écriture des signaux envoyés par les télécommandes traditionnelles, il permet de faire fonctionner des appareils plus anciens qui ne peuvent pas se connecter à Internet ou à d'autres systèmes de maison intelligente. En conséquence, nous n'avons pas besoin de mettre à niveau des appareils plus anciens ou d'acheter des appareils intelligents coûteux, ce qui permet d'économiser de l'argent.

La diode émettrice infrarouge et le récepteur infrarouge sont utilisés pour envoyer et recevoir des signaux infrarouges utilisés pour faire fonctionner les appareils de nos maisons. Ce projet permet de prendre en charge des appareils plus anciens qui ne peuvent pas se connecter à Internet ou à d'autres systèmes smart home.

En plus des éléments structurels, le projet *Smart ESP8266 remote* a également besoin d'un logiciel pour fonctionner correctement. Vous les trouverez sur [2].

Smart ESP8266 remote offre un certain nombre d'avantages tels que la commodité et la simplicité d'utilisation, des dépenses réduites et la polyvalence. Grâce à cela, nous n'avons pas besoin d'acheter des appareils intelligents coûteux ou d'améliorer des appareils plus anciens, réduisant ainsi les coûts. Cette conception peut être facilement modifiée ou adaptée pour fonctionner avec divers appareils et protocoles IR, ce qui en fait une solution polyvalente pour prendre en charge une variété d'appareils.

230656-04

LIENS

[1] TME shop : https://tme.eu
[2] Code source pour ce projet : https://techmasterevent.com/project/how-to-make-old-devices-smarter-with-a-esp8266

Blynk

comment construire des applications IoT sans expertise logicielle

avec la plateforme IoT Blynk et le matériel Espressif

Contribué par Blynk Inc.

Imaginez développer une application mobile sans coder, la personnaliser et la publier en un mois. Lancer un logiciel IoT de qualité professionnelle sans recruter d'ingénieurs logiciels ? Avec Blynk IoT, c'est possible en un mois et non en années !

Qu'est-ce qui est inclus dans la plateforme IoT Blynk ?

Blynk est une solution IoT low-code offrant le cloud, des bibliothèques de firmware, un constructeur d'application mobile sans code, et une console web pour tout gérer. Dès le départ, vous bénéficiez de la mise en service WiFi, de la visualisation de données, des automatisations, des notifications, des mises à jour OTA, ainsi que d'un système solide de gestion d'utilisateurs et d'appareils [1].

La bibliothèque de firmware Blynk prend en charge :
- ESP32
- ESP32-S2
- ESP32-S3
- ESP32-C3
- ESP8266
- et d'autres

Constructeur d'Application Blynk pour iOS et Android

Il permet de créer rapidement des prototypes et des applications complètes sans compétences en codage. En mode constructeur, sélectionnez parmi plus de 50 éléments UI personnalisables et glissez-les pour créer une interface utilisateur (UI) unique pour votre produit connecté. Personnalisez images, polices, couleurs et icônes.

Constructeur de tableau de bord web

Il offre une architecture similaire pour créer des visualisations de données en temps réel, pour contrôler et surveiller les appareils. Créez des interfaces indépendantes pour mobile et web selon vos besoins.

Figure 1. Interfaces No-Code créées avec Blynk.

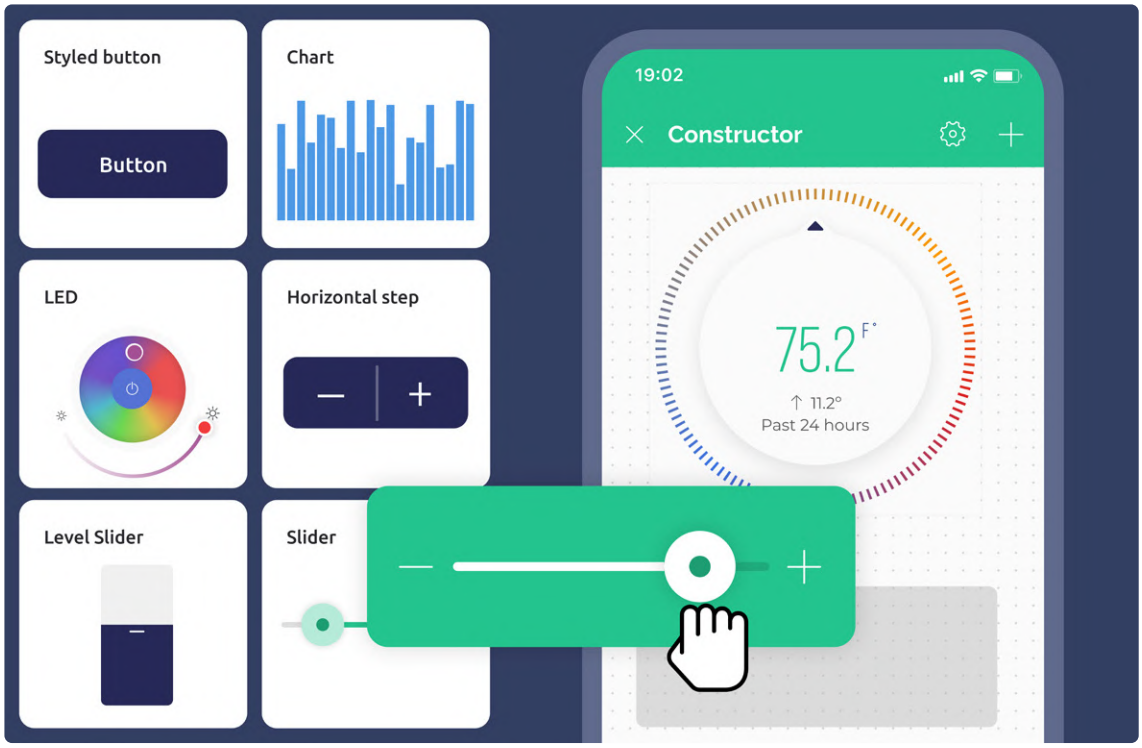


Figure 2. Constructeur d'application Blynk par glisser-déposer.

Système avancé de gestion des utilisateurs

Ce système permet de tout structurer, même à l'échelle de l'entreprise. Vous pouvez créer une structure d'organisation à plusieurs niveaux et gérer les appareils et les rôles des utilisateurs, les autorisations, les mots de passe et bien plus encore.

Gestion intégrée du cycle de vie des appareils

Cette fonctionnalité couvre tout ce qui est lié à la gestion des tokens, la mise en service WiFi, l'ajout d'appareils et leur affectation aux utilisateurs. Elle offre des mises à jour de firmware OTA fiables et sécurisées.

Automatisations sans Code

Peuvent être définis selon la date, l'heure, les actions des utilisateurs ou l'état de l'appareil. Informez les utilisateurs d'événements importants via des notifications push, des in-app, des emails ou des SMS.

Comment connecter votre ESP à Blynk ? Quel est l'effort d'intégration ?

Selon votre configuration, choisissez *Blynk.Edgent* [3] pour les appareils à microcontrôleur (MCU) unique. Si vous déchargez la connectivité sur un MCU secondaire, optez pour *Blynk.NCP* [4][5]. Les deux méthodes nécessitent un effort d'implémentation minimal avec des exemples fournis par Blynk. Pour la configuration à double MCU, utilisez un binaire prêt pour le NCP et une bibliothèque pour le MCU principal.
Votre parcours, de la configuration de l'appareil à une infrastructure IoT complète, peut prendre seulement quelques semaines [6].

230659-04

Obtenez 30 % de réduction sur le plan Blynk PRO pour la première année !

Code promo : ELEKTOR

Valable jusqu'au 31 janvier 2024 [2]

LIENS

[1] Site officiel : https://bit.ly/blynk-io
[2] Blynk.Console - créez votre compte gratuit : https://bit.ly/web-cloud
[3] Documentation Blynk.Edgent : https://bit.ly/docs-edgent
[4] Documentation Blynk.NCP : https://bit.ly/docs-ncp
[5] Qu'est-ce que Blynk.NCP : https://bit.ly/info-ncp
[6] Projet de station météo prêt à l'emploi pour expérimenter : https://bit.ly/blueprint-weather

concevoir une Interface Graphique sur ESP32

contribué par Slint

Les smartphones ont redéfini l'expérience utilisateur des interfaces utilisateur tactiles. La construction d'une interface utilisateur moderne nécessite l'utilisation de bibliothèques graphiques et d'outils modernes. Dans cet article, nous partagerons des conseils et présenterons Slint, une boîte à outils permettant de créer des UI interactives qui répondent et dépassent les attentes des utilisateurs.

Figure 1. Logos C++ et Rust.

À propos de Slint – une boîte à outils de nouvelle génération pour construire des UI graphiques natives en C++, Rust et JavaScript, avec un support multiplateforme, et plus de 10 000 étoiles sur GitHub.

Choisissez un langage de programmation - C/C++ ou Rust

En programmation embarquée, C et C++ ont été les langages de programmation préférés pendant longtemps. Mais Rust, réputé pour sa sécurité et ses performances, devient populaire parmi les développeurs embarqués.

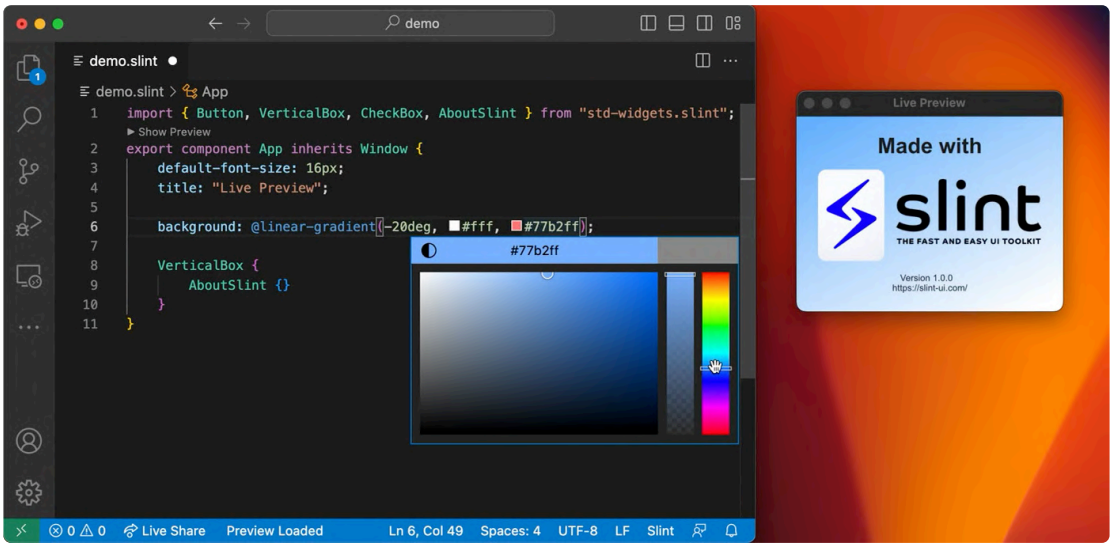


Figure 2. Itérations rapides avec l'aperçu en temps réel de Slint (Live-Preview).

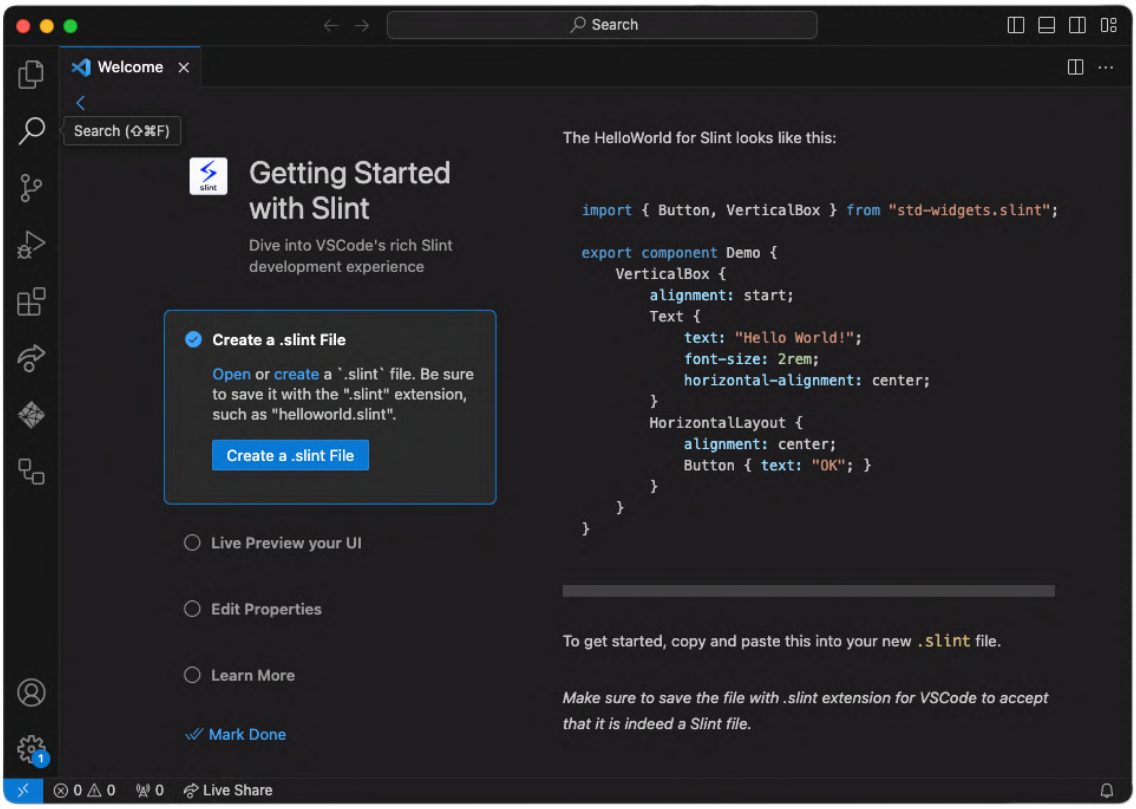


Figure 3.
Démarrer avec Slint.

Slint, la seule boîte à outils offrant des API natives à la fois pour C++ et Rust (**figure 1**), offre aux développeurs le choix : écrivez votre logique métier dans l'un ou l'autre de ces langages. De plus, il offre un chemin de transition pour ceux qui souhaitent passer leur code de C/C++ à Rust.

Séparez l'interface utilisateur de la logique métier

Dans Slint, l'interface utilisateur est définie à l'aide d'un langage similaire à HTML/CSS, favorisant une stricte division entre la présentation et la logique métier. Finalisez votre conception d'interface utilisateur grâce à des itérations rapides avec l'aperçu en temps réel de Slint (Live-Preview) (**figure 2**).

Profitez d'une bonne expérience de développement

La complexité actuelle du développement logiciel exige de bons outils.
Profitez de l'autocomplétion, de la coloration syntaxique, des diagnostics, de l'aperçu en direct, et bien plus encore (**figure 3**). De plus, Slint offre un composant ESP-IDF, simplifiant son intégration avec le framework de développement IoT d'Espressif.

Offrez une expérience utilisateur exceptionnelle

La performance de l'interface graphique est cruciale pour une (UX) exceptionnelle. Profitez de la flexibilité de Slint avec les capacités de rendu ligne par ligne ou sur framebuffer (**figure 4**).

Essayez Slint sur ESP32, visitez [1].

230670-04

Figure 4. Démo Slint sur ESP32.

LIEN

[1] Slint sur ESP32 : https://slint.dev/esp32

M5STACK

développement IoT rapide et facile avec M5Stack

Contribué par M5Stack

En tant que plateforme de développement IoT (Internet of Things, en français IdO = Internet des Objets) de renommée mondiale basée sur ESP32, le M5STACK permet de réaliser des centaines de contrôleurs, de capteurs, d'actionneurs et de modules de communication de façon modulaire, pouvant être connectés à l'aide d'interfaces standards. En empilant des modules de fonctionnalités diverses, les utilisateurs peuvent ainsi rapidement développer et vérifier leurs produits.

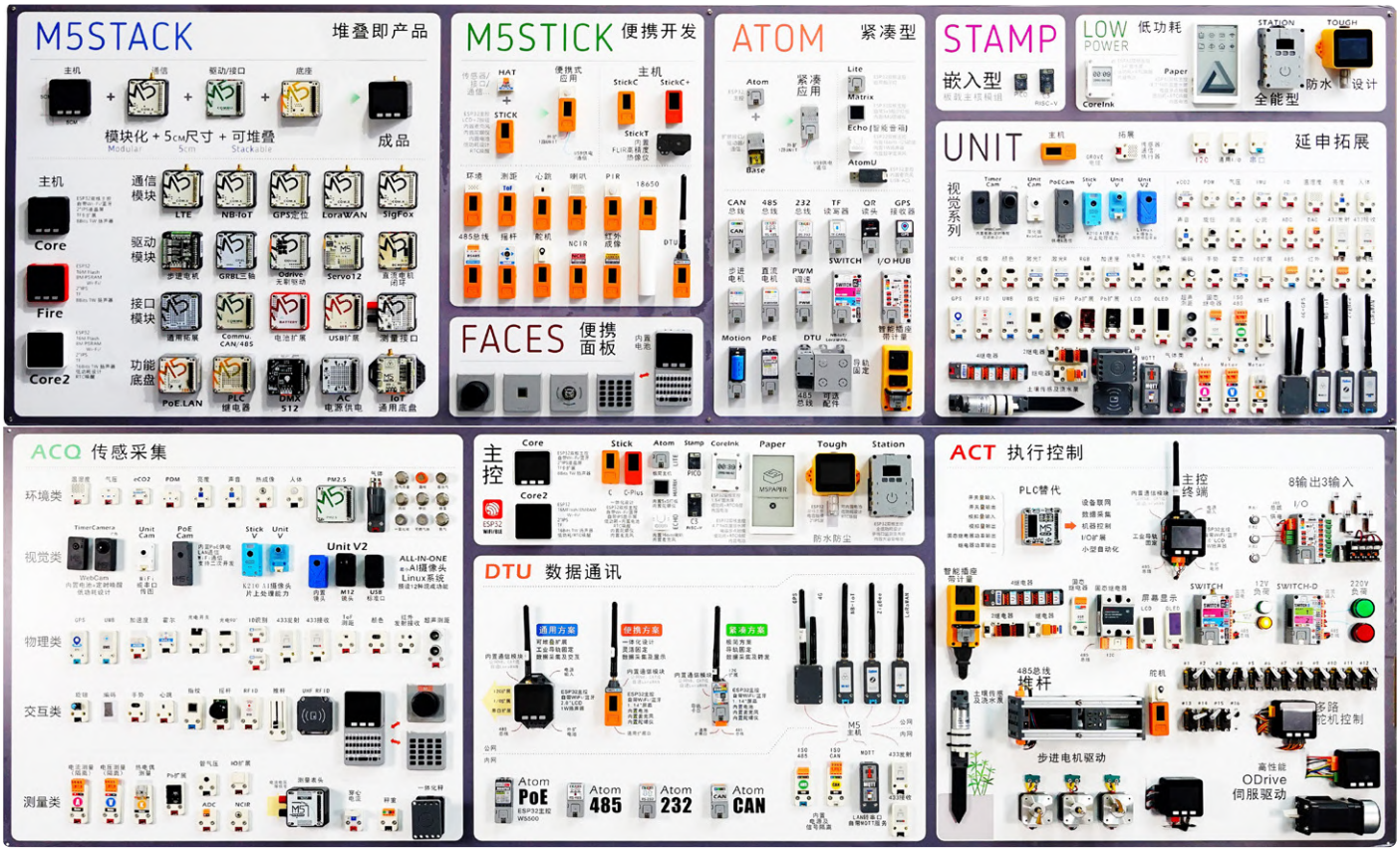


Figure 1. Famille Eco-system M5Stack.

Figure 2. Le M5Dial est adapté à la maison intelligente.

Si vous êtes un fan de l'ESP32, alors le M5Stack est indispensable !

Autant pour les débutants que pour les développeurs professionnels, les modules M5Stack (**figure 1**) [1] peuvent être connectés et gérés par l'IDE (Integrated Development Environment, en français EDI = environnement de développement intégré) de programmation graphique « UIFlow », ne nécessitant que peu de code tout en offrant une expérience optimale de prototypage de projets IoT.

Grâce aux modules matériels empilables ainsi qu'à la plateforme de programmation graphique conviviale, M5Stack propose aux développeurs des secteurs de l'IoT industriel, de la domotique, de la vente numérique, de l'agriculture intelligente et de l'éducation STEM (acronyme de science, technology, engineering, and mathematics, en français STIM = Science, Technologie, Ingénierie et Mathématiques), une programmation efficace et fiable, à l'aide d'une méthode rapide et facile (Quick & Easy).

Nouveau : le M5Dial

Le M5Dial [2], lancé récemment, est un produit très adapté à la maison intelligente (= maison équipée d'un système qui connecte des objets, des appareils et des systèmes entre eux, permettant de les gérer directement depuis une application installée sur un smartphone, une tablette ou une montre intelligente). Le M5Dial, une carte de développement polyvalente, embarquée, intègre les diverses fonctionnalités et capteurs nécessaires pour le contrôle d'une maison connectée (**figure 2**).

Le contrôleur principal du M5Dial est un M5StampS3, un microcontrôleur basé sur la puce ESP32-S3, connue pour ses hautes performances et sa faible consommation d'énergie. Il prend en charge les communications Wi-Fi et Bluetooth, ainsi que de multiples interfaces pour périphériques telles que SPI, I2C, UART, ADC, etc. Le M5StampS3 est également doté d'une mémoire flash intégrée de 8 Mo, offrant ainsi un espace de stockage suffisant pour l'utilisateur.

Le M5Dial dispose d'un écran tactile TFT (Thin-Film-Transistor, en français Transistor à Couche Mince) rond de 1.28 pouces, d'un encodeur rotatif, d'un module de détection RFID (Radio Frequency IDentification, en français IFR = Identification par Fréquence Radio), d'un circuit RTC (Real Time Clock, en français HTR = Horloge Temps Réel), d'un buzzer (petit vibreur), de boutons mécaniques et d'autres fonctionnalités, permettant aux utilisateurs de mettre facilement en œuvre divers projets.

La caractéristique principale du M5Dial est son encodeur rotatif, qui enregistre avec précision la position et l'orientation du bouton, apportant ainsi à l'utilisateur un contrôle interactif amélioré. Avec ce bouton rotatif, l'utilisateur peut régler des paramètres comme le volume, la luminosité, parcourir les menus, ou contrôler des équipements domestiques tels que des éclairages, des rideaux, la climatisation, etc. L'écran intégré de l'appareil peut également afficher différents effets en couleurs. Avec ses dimensions compactes de 45 mm × 45 mm × 32.2 mm et son poids léger de 46.6 g, le M5Dial est très facile à mettre en œuvre.

Qu'il soit utilisé pour contrôler des appareils ménagers dans le cadre de la maison intelligente ou pour surveiller et contrôler des systèmes dans le domaine de l'automatisation industrielle, le M5Dial peut être facilement intégré, en proposant un contrôle intelligent par l'intermédiaire d'une sélection interactive des fonctionnalités disponibles.

230662-04

LIENS

[1] The Innovator of Modular IoT Development Platform | M5Stack: https://m5stack.com
[2] ESP32-S3 Smart Rotary Knob w/ 1.28" Round Touch Screen:
https://shop.m5stack.com/products/m5stack-dial-esp32-s3-smart-rotary-knob-w-1-28-round-touch-screen

prototypage d'un compteur d'énergie basé sur l'ESP32

Figure 1. Rendu test de l'aspect de notre compteur d'énergie monophasé.

Saad Imtiaz (Elektor)

Cet article décrit la conception d'un compteur d'énergie basé sur un ESP32 d'Espressif, en mettant l'accent sur la surveillance de la consommation d'énergie en temps réel et sur la sécurité. Il met en lumière les étapes initiales, les exigences et les considérations qui interviennent lors de la réalisation d'un projet embarqué. Au fur et à mesure de l'avancement du projet, les réalisations futures seront partagées dans les prochaines éditions du magazine Elektor.

Dans le domaine de l'ingénierie, le recours aux bonnes technologies peut conduire à des avancées significatives. Ce projet vise à concevoir un compteur d'énergie en utilisant le microcontrôleur ESP32 d'Espressif et le circuit intégré de mesure d'énergie ATM90E32AS de Microchip. Dans cet article, nous décrivons brièvement le lancement de ce projet, de la sélection des composants au prototypage. L'objectif est simple : créer un système fiable de mesure précise de l'énergie à partir du tableau électrique principal de votre maison ou de votre labo. Ce compteur permettra aux utilisateurs de suivre leur consommation d'énergie en temps réel, offrant des informations qui peuvent conduire à une utilisation plus efficace de l'énergie.

Conception et exigences

Le projet est caractérisé par des objectifs et des exigences de conception clairs : surveiller en temps réel la puissance monophasée avec trois transformateurs de courant (TC), être abordable et facile d'utilisation. Le choix des composants ESP32 et ATM90E32AS IC a été guidé par ces objectifs, offrant à la fois un bon rapport coût-efficacité et des performances fiables. Un autre objectif était de maintenir des dimensions inférieures à 100×80×30 mm (L×l×h) afin de s'assurer qu'il puisse être logé dans un boîtier de disjoncteur. Pour améliorer la convivialité, nous avons également prévu une interface mobile pour la surveillance à distance, ainsi qu'un écran OLED avec des boutons pour l'interaction directe. Le projet permet également des mises à jour logicielles futures, garantissant une utilité à long terme pour le consommateur. La **figure 1** présente le rendu du prototype actuel de l'enceinte.

Choix du microcontrôleur

Le choix du microcontrôleur ESP32 a été fondé sur une analyse détaillée de ses capacités. Cette puce excelle dans plusieurs domaines essentiels à la réussite de ce projet. Tout d'abord, sa facilité d'intégration dans différents types de circuits donne une flexibilité au cours de la phase de développement. Deuxièmement, son rapport coût-efficacité en fait un choix intéressant pour un prototype qui vise à concilier performance et budget. Troisièmement, la compatibilité avec une large gamme de capteurs et de circuits intégrés offre des avantages significatifs. Enfin, le soutien étendu de la communauté pour la puce ESP32 renforce son adéquation à ce projet. La **figure 2** met en évidence les principales caractéristiques et les avantages de l'ESP32-D0WD-V3 qui ont conduit à sa sélection pour ce projet.

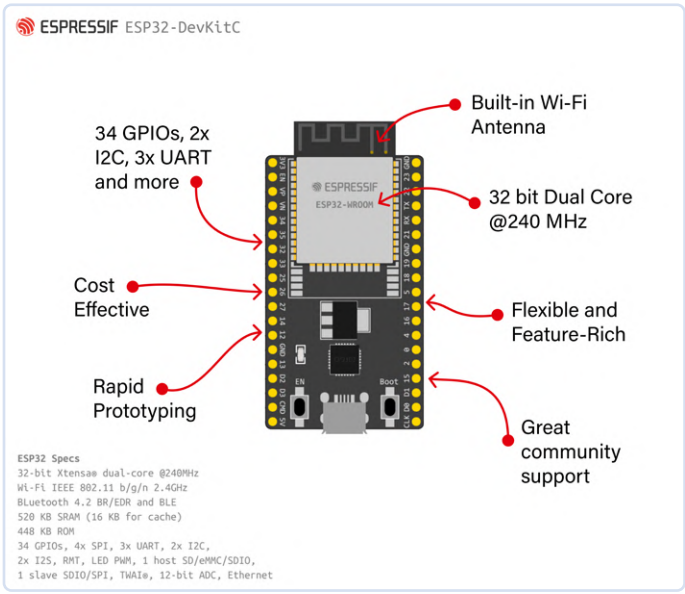


Figure 2. Principales caractéristiques et avantages de l'ESP32.

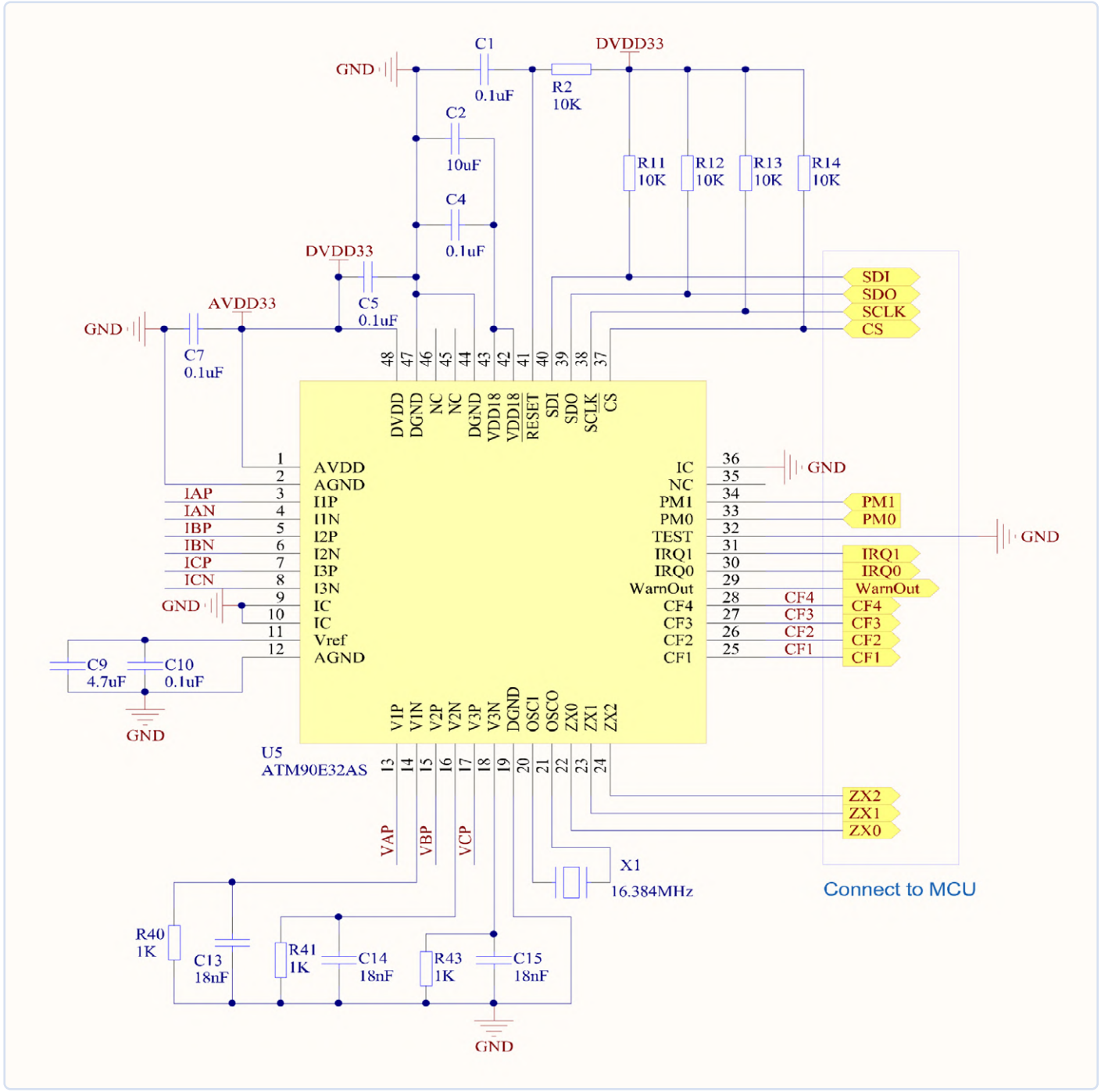


Figure 3. Le compteur d'énergie est basé sur une note d'application d'Atmel [2]. Ici, vous pouvez voir le circuit autour du CI de compteur.

Intégration des CI de compteur

Nous avons intégré le circuit intégré ATM90E32AS IC de Microchip conformément à la note d'application du fabricant ; ce document a servi de pierre angulaire pour garantir que le circuit intégré de mesure de l'énergie communiquait de manière transparente avec le microcontrôleur ESP32. Toutefois, cette phase n'a pas été exempte de difficultés. L'acquisition des bons composants dans le respect des contraintes budgétaires a nécessité une planification rigoureuse, compte tenu des contraintes de disponibilité. La **figure 3** présente la note d'application fournie par Atmel (aujourd'hui Microchip).

Phase de conception et normes de sécurité électrique

La phase de conception est en effet une partie essentielle du processus de fabrication, en particulier lorsque la sécurité est un facteur essentiel. Dans un appareil conçu pour interagir avec les tensions alternatives du secteur, il convient d'accorder une attention méticuleuse à la conformité avec les normes de sécurité établies. La **figure 4** présente le schéma fonctionnel du projet.

Pour garantir la sécurité, nous avons intégré plusieurs composants électriques dédiés dans le circuit de l'appareil. Nous avons utilisé des varistances à oxyde métallique (MOV) pour supprimer les tensions transitoires afin de protéger les circuits contre les pics de tension. En outre, nous avons inclus des fusibles afin d'assurer une sécurité absolue contre les surintensités.

Au-delà de la sélection des composants, nous nous sommes également concentrés sur des considérations de disposition qui respectent les normes de sécurité. Nous avons respecté des lignes de fuite et les distances d'isolement adéquates entre les éléments conducteurs

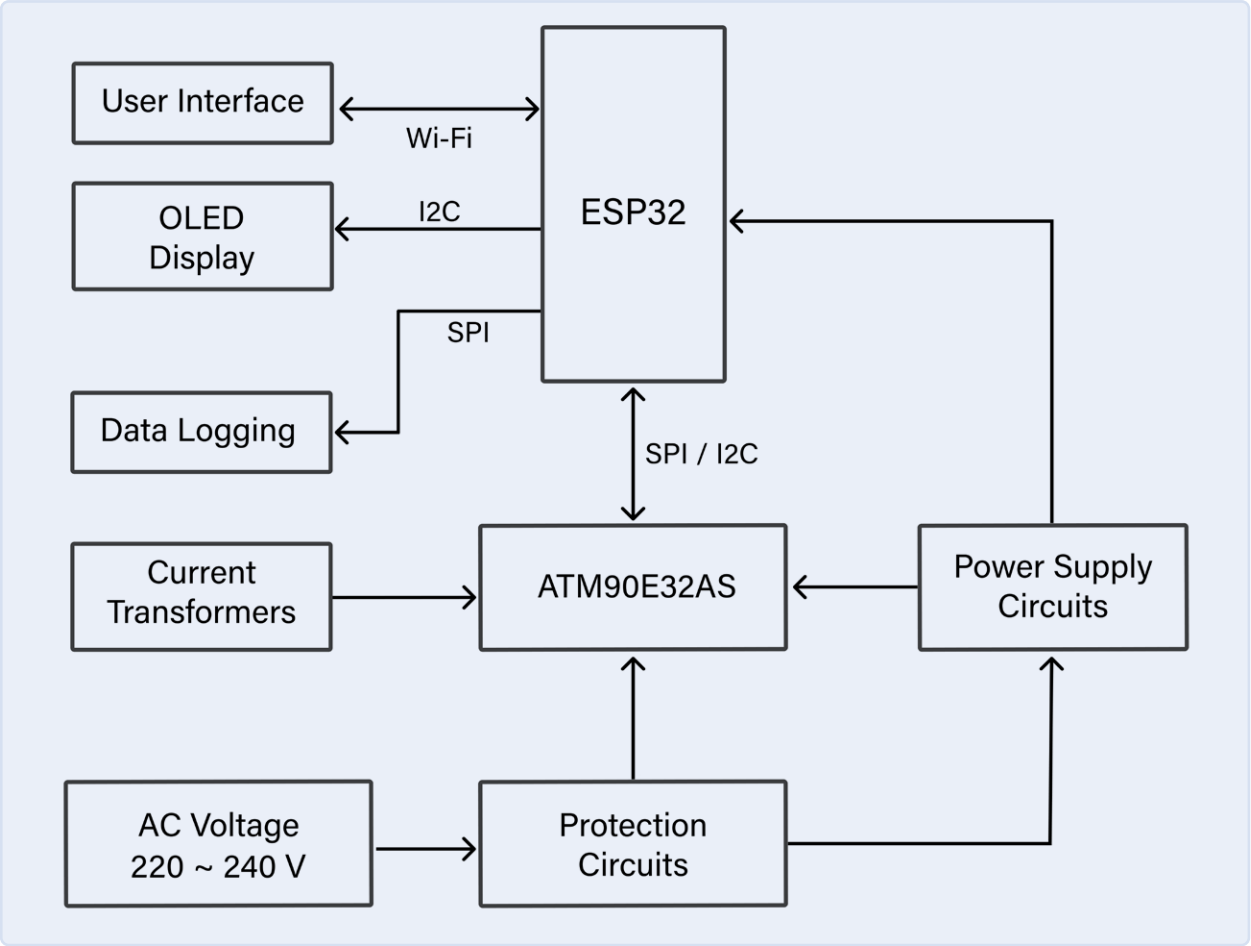


Figure 4. Schéma de principe de notre projet de compteur d'énergie.

sur le circuit imprimé afin d'éviter les arcs électriques. Nous avons calculé avec soin la largeur des lignes de courant alternatif pour tenir compte des valeurs nominales de courant, conformément aux normes IPC-2221 [1]. Cet aspect était essentiel pour garantir les performances thermiques de la carte dans des conditions de pleine charge. Nous avons utilisé un plan de masse solide pour garantir la mise à la masse. Nous avons accordé une attention particulière à la conception des paires différentielles pour assurer l'intégrité des signaux, en veillant à ce que le routage suive une géométrie précise afin de minimiser les interférences électromagnétiques.

Choix du fabricant : JLC PCB

Après avoir consulté plusieurs services d'assemblage de circuits imprimés, nous avons choisi JLC PCB. La raison principale était l'équilibre entre la rentabilité et la fiabilité offertes par ce service. Cette décision était importante pour maintenir le projet dans les limites du budget sans compromettre la qualité de la carte assemblée. Le schéma du prototype et la conception des circuits imprimés sont en cours de finalisation et seront bientôt prêts pour la production.

Réflexion sur le voyage et regard vers l'avenir

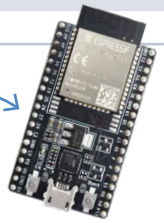
Rétrospectivement, ce projet montre les résultats que l'on peut obtenir lorsqu'une planification minutieuse est associée à une bonne conception technique. Les obstacles que nous avons rencontrés nous ont aidés à améliorer notre projet. Alors que nous passons de la fabrication d'un prototype à une éventuelle production en série, nous espérons que ce projet fera une réelle différence dans la façon dont les gens gèrent l'énergie. Ce projet sera détaillé dans les prochaines éditions de ce magazine – nous sommes encore en train de fabriquer le prototype, de le tester et de travailler sur le logiciel. Il y a encore beaucoup à faire, alors restez à l'écoute pour des mises à jour sur ce projet. Nous approcherons de l'achèvement et partagerons les mises à jour dans l'édition de janvier/février 2024 d'Elektor, qui sera consacrée au thème « alimentations et énergie ».

230646-04

Questions ou commentaires ?
Contactez Elektor (redaction@elektor.fr).

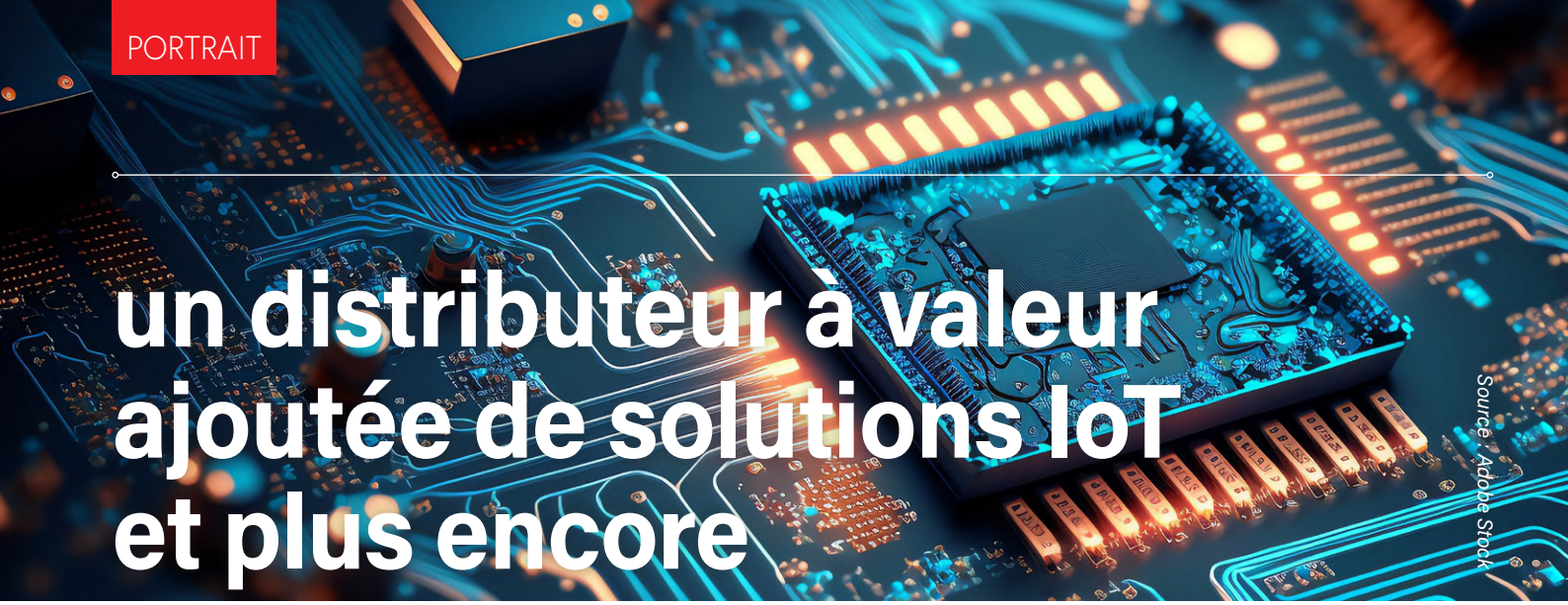
Produits

- **ESP32-DevKitC-32E**
  www.elektor.fr/20518
- **ESP32-C3-DevKitM-1**
  www.elektor.fr/20324



LIENS

[1] Normes IPC-2221A : https://www-eng.lbl.gov/~shuman/NEXT/CURRENT_DESIGN/TP/MATERIALS/IPC-2221A(L).pdf
[2] Note d'application ATM90E32AS : https://ww1.microchip.com/downloads/en/Appnotes/Atmel-46103-SE-M90E32AS-ApplicationNote.pdf



un distributeur à valeur ajoutée de solutions IoT et plus encore

Source : Adobe Stock

contribué par Steliau Technology

Steliau Technology est une entreprise innovante spécialisée dans les solutions électroniques. Elle se distingue par son expertise en ingénierie et sa passion pour l'innovation. Steliau Technology, à travers son partenaire Espressif, fournit des composants électroniques essentiels à la connectivité sans fil et à l'IoT, tels que l'ESP32-C5, l'ESP32-C6, l'ESP32-P4, l'ESP32-S6 et bien d'autres produits.

Fondée en 2018, Steliau Technology [1] se définit comme un distributeur à valeur ajoutée de solutions électroniques. Interfaces Hommes-Machines, écrans et solutions tactiles, connectivité et IoT, autant de domaines d'expertises largement maitrisés par Steliau et qui lui octroient une réputation déjà bien établie sur le marché de l'électronique.

Steliau Technology est reconnue pour ses partenariats stratégiques avec des entreprises de pointe dans l'électronique, lui permettant de renforcer son positionnement dans les domaines de l'IoT et de la connectivité.

Espressif & Steliau Technology entretiennent une relation de partenariat historique, qui ne cesse de se renforcer au fil des années. En tant que distributeur officiel pour la France et l'Italie, Steliau Technology est le guichet unique pour les solutions Espressif.

Steliau Technology est parfaitement équipée pour offrir un support complet de toute la gamme des produits Espressif, à la fois au niveau hardware et software embarqué. L'équipe possède une expertise technique approfondie qui couvre tous les aspects de la connectivité, de la conception matérielle à la programmation logicielle. Cela signifie que Steliau Technology est en mesure de fournir une assistance complète aux clients, garantissant ainsi des solutions de connectivité robustes et performantes. Ce partenariat solide garantit à nos clients un accès privilégié aux meilleures solutions de connectivité sur le marché, comme les dernières générations Espressif : ESP32-C5, ESP32-C6, ESP32-P4, ESP32-S6.

Steliau Technology à travers son partenaire Espressif fournit des composants électroniques essentiels à la connectivité sans fil et à l'IoT dans une variété de marchés et de secteurs, contribuant ainsi à l'évolution constante de la technologie.

Solutions IoT et bien plus encore

Tout d'abord, dans le domaine de l'Internet des objets (IoT), les microcontrôleurs Wi-Fi et Bluetooth d'Espressif sont largement utilisés. Ces composants sont essentiels pour les applications domestiques intelligentes, comme les thermostats connectés, les caméras de sécurité, et les dispositifs de contrôle d'éclairage. Elles jouent un rôle prépondérant dans l'interopérabilité dans les produits connectés de la maison avec des solutions MATTER compatibles (au travers le Wi-Fi, le thread notamment). Les solutions d'Espressif sont également utilisées dans le secteur industriel pour la surveillance à distance, la collecte de données, et le contrôle des machines. En outre, les produits d'Espressif sont présents dans le secteur de la santé, où ils alimentent des appareils médicaux connectés et des dispositifs de suivi de la condition physique.

En tant que partenaire électronique global, Steliau a la capacité de proposer des solutions intégrant les produits d'Espressif pour le contrôle des écrans tactiles. Forts de son expertise, Steliau Technology est en mesure de concevoir des solutions intégrées dans lesquelles les produits d'Espressif contrôlent l'écran tactile, avec à notre actif plusieurs succès stories, notamment pour des tailles d'écran de 7 pouces. Notre capacité à offrir une solution globale est renforcée par un support technique spécifique couvrant l'ensemble de ces domaines.

230661-04

LIEN

[1] Steliau Technology : https://steliau-technology.com

Des demandes ?

Steliau est à la disposition des clients pour toutes demandes. Merci de contacter remi.krief@steliau-technology.com pour toute demande relative aux solutions Espressif.

en détail :
entretien avec Arduino sur le Nano ESP32

Alessandro Ranellucci et Martino Facchin abordent la collaboration avec Espressif

Questions de Saad Imtiaz et de Clemens Valens (Elektor)

À l'été 2023, Arduino a lancé la carte Nano ESP32. Basée sur l'ESP32-S3 d'Espressif, la nouvelle carte offre une connexion Wifi 2,4 GHz, 802.11 b/g/n, et une connectivité Bluetooth 5 (basse consommation) à longue portée dans un format Nano. Le Nano ESP32 n'est pas la première carte Arduino équipée d'un processeur d'Espressif, mais cette fois-ci, il s'agit de l'élément principal et non d'un simple module de communication sans fil supportant un autre microcontrôleur. Elektor a interviewé Alessandro Ranellucci (Directeur, Arduino Makers Division) et Martino Facchin (Responsable Hardware & Firmware, Arduino) à propos de cette collaboration entre Espressif et Arduino.

Elektor : pouvez-vous expliquer pourquoi vous avez choisi l'ESP32 pour la nouvelle série Arduino Nano plutôt qu'un autre microcontrôleur ?

Alessandro Ranellucci : la série Arduino Nano représente un groupe de cartes conçues pour être compatibles les unes avec les autres en termes de forme, de disposition de broches et de caractéristiques techniques essentielles. Cette uniformité est appréciée par notre communauté de *makers* car elle permet une interchangeabilité sans faille au niveau de la série Nano. Cependant, nous n'avions pas encore introduit de carte basée sur la très répandue architecture ESP32. Nous avons estimé qu'il était nécessaire de fournir une option efficace au sein de la série Nano qui utilise l'ESP32, ce qui nous a amenés à prendre cette décision.

Martino Facchin : oui, au cours des quatre dernières années à peu près, nous avons développé la série Nano en y intégrant des microcontrôleurs de divers fournisseurs de puces avec lesquels nous n'avions pas travaillé auparavant. Nous avons commencé par intégrer des microcontrôleurs de Nordic, puis de Raspberry Pi. Il s'agit essentiellement de notre terrain d'expérimentation pour tester des architectures nouvelles et répandues, tout en ajoutant une utilité qui n'est pas disponible ailleurs. Par exemple, sur la puce simple ESP32, il n'y avait pas d'USB. Dans la C3, il y a un port USB, mais il est associé à un identifiant de fournisseur et de produit fixe, de sorte qu'il n'est pas possible de faire la distinction entre les différentes cartes. La S3 est la première carte à disposer de cette fonctionnalité, c'est pourquoi nous l'avons utilisée, plutôt qu'une carte plus ancienne.

Elektor : quels sont les avantages techniques uniques de l'ESP32 par rapport aux autres options ?

Martino Facchin : oui, en effet, nous avons eu l'occasion de collaborer directement avec u-blox pour acquérir une puce spécialisée qui n'est pas disponible à l'achat, qui comprend de la PSRAM intégrée dans la puce elle-même. Nous proposons une puce avec PSRAM et la plus grande quantité possible de mémoire flash externe. Ces détails sont des approfondissements techniques, pour dire ainsi, conçus pour les passionnés et les geeks. Les caractéristiques de la carte poussent l'ESP32 dans ses derniers retranchements les plus élevés disponibles. Ensuite bien sûr, elle inclut le WiFi, le Bluetooth à basse consommation (BLE), le traitement double-coeur, et toutes les autres fonctionnalités qui viennent avec l'ESP32. Nous

Martino Facchin

Alessandro Ranellucci

avons choisi de ne pas inclure de fonctionnalités supplémentaires sur l'Arduino Nano ESP32 comme nous l'avons fait avec d'autres cartes. Celle-ci est conçue comme un bloc de construction. Contrairement à l'Arduino BLE, ce n'est pas un dispositif que vous pouvez utiliser dès sa sortie de la boîte pour des applications conséquentes. Vous devez attacher des composants supplémentaires à la carte, tels que des capteurs ou des modules. Elle est équipée d'une LED RVB, mais en plus de cela, l'utilisateur a la liberté de la compléter avec d'autres composants, en l'utilisant comme une base bien adaptée pour ses projets.

Elektor : avez-vous rencontré des difficultés pour intégrer la puce ESP32 dans le format Arduino Nano, et si oui, comment avez-vous résolu ces problèmes ?

Alessandro Ranellucci : en effet, le premier défi consistait à former un partenariat avec Espressif. Ils ont créé un excellent noyau Arduino pour les cartes ESP32 pour de nombreuses années, un noyau qui est profondément intégré dans le système Arduino, mais qui a également respecté leurs préférences techniques distinctes. Le plus difficile était de trouver comment combiner nos efforts pour créer une expérience utilisateur plus cohérente et améliorée. C'est ainsi que nous avons mis en place cette collaboration. Je pense que Martino peut également mentionner d'autres défis, comme, par exemple, la numérotation des broches.

Martino Facchin : vraiment pas tant que cela, car du point de vue matériel, il n'y a pas eu de problèmes. Bien sûr, c'est facile par rapport aux autres cartes que nous avons fabriquées ces deux dernières années. D'un point de vue logiciel, c'est très différent car la numérotation des broches était très liée au monde des ESP. Les broches étaient numérotées exactement de la même manière que sur le support du processeur. Sur le dessous de la carte, vous pouvez voir des étiquettes d'autres fabricants avec des numéros comme 31, puis à côté un 4, puis un 55, et ainsi de suite. Cela ne convient pas au Nano, ce n'est pas la solution que nous recherchions. Nous avons donc développé un moyen de convertir les numéros de broches logiques en numéros de broches internes. Nous avons fait accepter cette solution au centre de la communauté du noyau ESP32. À l'heure actuelle, toutes les cartes équipées d'un ESP32, quel que soit le fabricant, peuvent utiliser la même logique si elles souhaitent adopter notre philosophie, et il s'agit d'une contribution directe à un noyau dont nous n'assurons pas la maintenance. Cela a été difficile parce que la date de sortie de cette fonctionnalité devait correspondre parfaitement à la date de sortie de la carte. Cependant, nous avons finalement réussi.

C'était un défi, car c'était la première fois que nous sortions quelque chose que nous ne contrôlions pas entièrement, et c'était difficile, mais nous l'avons fait.

Alessandro Ranellucci : nous avons tiré des enseignements importants de l'ensemble du processus de validation car, comme l'a indiqué Martino, tout fabricant peut désormais opter pour l'utilisation de numéros de broches logiques. Il est plus convivial de numéroter les broches de manière séquentielle que d'utiliser la numérotation des broches du contrôleur, comme PA1 et PD1.

Elektor : le développement du logiciel ESP32, l'écosystème et le soutien de la communauté ont-ils influencé votre décision de l'utiliser dans le nouvel Arduino Nano ?

Martino Facchin : non. Nous aurions choisi de l'utiliser même s'il n'y avait pas eu de soutien de la part de la communauté. Il est certain qu'avec le travail considérable de la communauté et le développement de nos projets, nous bénéficions grandement de leurs contributions, en particulier pour le gestionnaire de bibliothèque. Certaines bibliothèques étaient déjà compatibles avec notre carte, ce qui était avantageux. D'autres étaient exclusives à l'ESP32, ce qui était un inconvénient, mais nous pouvons maintenant les utiliser également. Cette compatibilité a été un facteur dans notre décision, mais nous aurions continué avec l'ESP32 de toute manière.

Alessandro Ranellucci : en ce qui concerne le logiciel, nous avons la possibilité de créer un nouveau noyau, comme nous l'avons fait pour d'autres produits, tels que le RP2040, pour lequel nous avons développé notre propre noyau afin d'avoir un contrôle total sur le logiciel. Mais Espressif a réalisé un excellent travail au

> *Pin numbering was a challenge because the numbers were very tied to the ESP32 world. But we found a solution – logical pin numbers.*
>
> Martino Facchin

fil des ans et dispose d'une communauté solide. C'est pourquoi nous avons choisi de collaborer avec eux, une décision influencée par la force de l'écosystème.
Nous aspirons à rester neutres en matière de technologie, en évitant de nous engager exclusivement dans la gamme de microprocesseurs d'un seul fabricant. Notre objectif est d'offrir une plateforme Arduino polyvalente et interopérable, car c'est ainsi que les utilisateurs considèrent Arduino et leurs attentes envers la plateforme. C'est pourquoi nous expérimentons et recherchons en permanence de nouveaux produits à développer.
Martino Facchin : les personnes qui passent à Arduino à partir d'autres environnements, tels que les BSP (*Board Support Package*) ou les environnements intégrés avec des processus d'installation et de configuration laborieux, disent souvent qu'Arduino fonctionne, tout simplement. C'est, je crois, notre plus grande valeur, lorsqu'un utilisateur peut commencer à utiliser Arduino de la manière la plus simple et la plus rapide possible. Et comme Alessandro l'a mentionné, nous ne sommes pas fermés au monde extérieur. Il y a six ans, je ne pouvais pas affirmer cela parce qu'Atmel était notre principal sponsor, mais aujourd'hui nous sommes complètement indépendants.

Elektor : pouvez-vous nous parler des améliorations ou des changements que vous avez apportés à la plate-forme ESP32, pour la rendre plus compatible avec les objectifs de l'Arduino Nano ESP32 ?
Martino Facchin : nous avons modifié le processus de téléchargement habituellement utilisé pour l'ESP32, qui obligeait traditionnellement les utilisateurs à passer au module USB natif pour utiliser l'outil ESP, un processus qui n'était pas intégré de manière optimale à notre IDE. Désormais, lors du téléchargement d'un sketch, une méthode standard OTS (*Off The Shelf*) est employée. Le programme est téléchargé via *Device Firmware Update* (DFU), ce qui le déplace vers la deuxième partition par le biais d'une mise à jour OTA (*Over-The-Air*). Le programme d'amorçage est ensuite chargé de tenter un redémarrage à partir de cette deuxième partition. En cas de succès, le nouveau sketch est chargé, et dans le cas contraire, le sketch d'origine est conservé. Cette implémentation est une amélioration significative, car Espressif avait déjà envisagé plusieurs cas d'utilisation. Nous avons adapté leur approche pour développer un système plus rapide et plus fiable avec un mode de récupération.
Nous avons intégré le double clic (vous pouvez entrer en mode *recovery* en appuyant deux fois sur le bouton de réinitialisation), ce qui n'existait pas sur la carte ESP. Ce sont quelques modifications qui ont été bien acceptées par la communauté en raison de l'effort fourni. Même si le noyau communautaire est soutenu par Espressif, il s'agit d'un effort de la communauté, donc tout le monde était content de cela.

Alessandro Ranellucci : il y a deux autres contributions que nous avons faites à l'écosystème dans son ensemble, donc pas spécifiquement au noyau. Une partie du travail que nous avons effectué était du débogage, et nous avons également ajouté la prise en charge de MicroPython pour l'ESP32.

Elektor : en ce qui concerne le débogage, est-ce sur cette carte qu'il y a un accès à l'ESP par des points de soudure, ou est-ce que c'est sur une autre carte ?
Martino Facchin : non, vous pouvez déboguer cette carte directement via USB, Espressif a eu la gentillesse de le proposer. Via l'USB, vous avez un port série, le CDC, les choses habituelles, et aussi une interface JTAG. Vous pouvez interagir avec l'interface JTAG en utilisant la liaison USB habituelle en même temps que la communication normale. Il n'est pas vraiment difficile de connecter un débogueur externe, mais ce n'est pas nécessaire. En fait, il est déjà intégré dans l'EDI . Il suffit de connecter la carte, de sélectionner « *Debug mode* (Hardware CDC) » dans le menu et d'appuyer sur le gros bouton *Debug*. En plus, un tas de choses impliquant OpenOCD et GDB seront exécutées, mais c'est transparent du point de vue de l'utilisateur.
Alessandro Ranellucci : notre contribution a été de concevoir une expérience utilisateur pour cela. Avec Arduino, c'est beaucoup plus facile de tester, de documenter, de sensibiliser, de diffuser auprès des utilisateurs et d'interagir ensuite avec Espressif que d'utiliser des outils de débogage professionnels complets.

Elektor : les détails de la mise en œuvre me rappellent des questions posées en ligne. Il y avait une confusion au sujet des ports lorsque l'on branchait la carte, car il y avait deux ports et les gens ne savaient pas quel port utiliser.
Martino Facchin : oui, ce n'est pas facile à expliquer, parce que la principale façon dont nous avons toujours dit aux gens de faire les choses a été de commencer par les bases. Par exemple, vous le faites de cette façon, et il y a un port série, et ainsi de suite, et ensuite vous passez à un sujet avancé. Les sujets plus complexes doivent être très bien expliqués. C'est pourquoi nous disposons d'une documentation sur la manière de procéder correctement au débogage. Nous ne nous contentons pas de laisser la fonctionnalité en place et de laisser les gens expérimenter. De plus, les gens ne lisent généralement pas la documentation, c'est pourquoi nous faisons de notre mieux pour éviter ces inconvénients et mettons tout à leur disposition. La documentation devrait toujours être lue, et tout y est très bien expliqué.
Alessandro Ranellucci : la version la plus récente de l'EDI Arduino, sortie il y a un mois, a apporté de nombreuses améliorations à la sélection des ports et de choix des cartes. La confusion concernant les ports multiples ou les ports qui changent après que vous avez effectué des opérations, etc., est gérée de manière plus conviviale.

Martino Facchin : nous avons également ajouté ce menu DFU, de sorte que nous disposons désormais d'une fonction de *pluggable discovery*. Cette détection à l'insertion est un concept très intéressant car vous pouvez découvrir des choses via le port série ou par WiFi via MDNS. Paul Stoffregen a ajouté *discovery* via HID à la version 1 de l'EDI Arduino parce que son programme d'amorçage était basé sur HID, mais il s'agissait toujours d'un correctif externe pour le programme d'installation Arduino. Nous avons décidé de rendre cela disponible pour tout le monde. Il a donc porté son *discoverer* sur la version 2 de l'EDI Arduino. Ensuite, nous avons eu un autre problème avec le Minima, car lorsque vous passez en mode *bootloader*, il ne montre pas de port série. C'était déroutant de ne pas le voir dans le menu. C'est pourquoi nous avons donc décidé d'ajouter notre *discoverer* au port DFU. L'ESP32 dispose d'un DFU pendant l'exécution, en raison de la façon dont vous téléchargez le code. Occasionnellement, il peut apparaître sur votre ordinateur, mais, comme je l'ai mentionné précédemment dans la documentation, tout est expliqué. Ce port est destiné à vos téléchargements et vous pouvez choisir l'un ou l'autre. Cela ne change rien au fait qu'il accepte les téléchargements.

Elektor : y a-t-il eu des changements de compatibilité en termes de bibliothèques de codes existants, lors de la transition vers la puce ESP32 pour le nouveau modèle Nano ?
Alessandro Ranellucci : oui c'est le cas, chaque fois que nous ajoutons une nouvelle architecture à la famille. Il y a les bibliothèques officielles de base qui doivent être portées, en particulier lorsqu'elles sont hautement optimisées, de sorte qu'elles fonctionnent avec un code de bas niveau. Dans ce cas, nous avons dû étendre certaines bibliothèques de base. Mais l'écosystème des bibliothèques pour l'ESP32 était beaucoup plus développé.

Elektor : pouvez-vous mettre en évidence les caractéristiques de sécurité ou les considérations qui vous ont amené à choisir l'ESP32 pour l'Arduino Nano ESP32, en particulier pour les applications IdO ?
Alessandro Ranellucci : je ne dirais pas que nous avons choisi l'ESP32 pour des raisons de sécurité. Bien sûr, l'ESP32 a quelques capacités, que nous utilisons partiellement pour le moment.

Martino Facchin : vraiment en partie, car nous n'utilisons pas de cryptage ou ce qu'ils appellent le « secure boot » parce que cela rend la vie de l'utilisateur extrêmement difficile. Vous devez signer chaque code binaire que vous produisez, et vous devez ensuite le modifier pour chaque carte, sinon cela n'a pas de sens. Nous ne l'appliquons donc pas, mais nous l'autorisons, bien sûr. Du point de vue de l'intégrateur, lorsque vous prenez ce produit et que vous voulez le rendre vraiment sûr, il a toutes les fonctionnalités pour le faire. Mais ce n'est pas la raison pour laquelle nous l'avons choisi.
Alessandro Ranellucci : habituellement, sur toutes nos cartes, nous avons un élément matériel sécurisé séparé. C'est la méthode que nous avons choisie pour tous nos produits. Dans le cas présent, nous ne l'avons pas mise en œuvre parce que le microcontrôleur principal, en théorie, possède ces capacités. Pour l'instant, nous avons décidé d'utiliser cette puce telle quelle, par souci de simplicité. Rien ne nous empêche de poursuivre le développement, bien entendu.
Martino Facchin : la même chose s'est produite avec le Portenta H7, par exemple, lorsque nous avons lancé le programme d'amorçage sécurisé (*secure bootloader*) et l'infrastructure de démarrage MCO pour fournir des mises à jour OTA et sécurisées. Il s'agissait d'une option à confirmer, et non de quelque chose que nous fournissons directement. À terme, je suis certain que nous fournirons également une sorte de documentation par le biais d'un menu.

Elektor : est-il prévu d'utiliser la puce de Espressif dans davantage de cartes Arduino, telles que la gamme PRO ?
Alessandro Ranellucci : je ne peux pas répondre pour la gamme PRO, mais en général, oui.
Martino Facchin : nous aimons le facteur de forme *u-blox* parce qu'il s'adapte vraiment bien aux Nano. Il est très petit. Nous avons utilisé leurs modules sur presque toutes les cartes Nano, soit comme puce d'accompagnement pour le WiFi, soit comme microcontrôleur principal pour le Nano BLE. Nous sommes donc très satisfaits de leur qualité en tant que fabricant. En même temps, bien sûr, sur la UNO R4, nous avions le module Espressif normal. Et oui, nous prévoyons de faire d'autres choses.

Elektor : pouvez-vous nous donner plus de détails sur la collaboration ou les contacts que vous avez eus avec l'équipe de développement de l'ESP32, pour garantir une intégration transparente dans l'écosystème Arduino ?
Alessandro Ranellucci : il y a une bonne équipe de développement chez Espressif. Elle fait un excellent travail en impliquant la communauté dans le processus de développement. Nous avons donc eu des réunions individuelles avec eux, ainsi que des appels récurrents. Nous avons également partagé un canal de communication sur Slack ou d'autres plateformes de communication, afin d'avoir un moyen direct et quotidien de se mettre à jour, de s'informer mutuellement des problèmes et d'arriver à un consensus avant de rendre le projet public sur GitHub. Il s'agissait d'une collaboration très étroite. Nous pouvons donc citer de nombreuses personnes d'Espressif qui nous ont aidés dans cette collaboration.
Alessandro Ranellucci : en fait, à un niveau plus élevé, Ivan Grokhotkov (vice-président de Software Platforms, Espressif) nous a aidés à faciliter toute la communication, et Pedro Minatel (Developer Advocate) a été d'une aide précieuse en nous aidant à parler à la bonne personne.

Elektor : à l'avenir, imaginez-vous l'évolution du partenariat entre Arduino et Espressif, et quelles sont les possibilités d'innovation et de collaboration que vous anticipez ?
Alessandro Ranellucci : Je dirais qu'au-delà des produits matériels, les deux équipes sont d'accord sur la nécessité de travailler étroitement à la normalisation de l'API. L'API est désormais plus interopérable entre les mondes Arduino et ESP32. C'est sur ce point que nous aimerions poursuivre notre collaboration.
Martino Facchin : de plus, Espressif s'est diversifié dans de nombreux domaines, tels que Rust. De nombreux développeurs travaillent sur Rust, qui n'est pas une priorité pour nous, mais nous pourrions obtenir des idées très intéressantes à partir du travail qu'ils font dans ce domaine. Il y a aussi le système d'exploitation Zephyr, dans lequel nous sommes tous deux partenaires. Au sein du comité de pilotage technique, nous pouvons prendre des décisions, tout comme Espressif. Tout le développement que nous entreprenons tous les deux est axé sur l'amélioration des prochains outils pour les générations futures. En fin de compte, nous travaillons dans le même but.

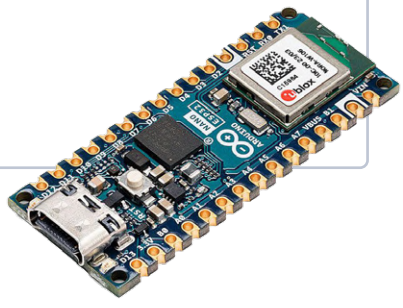
Elektor : lorsque vous comparez l'ESP32 à d'autres puces, pourriez-vous préciser les critères de référence, ou les mesures de performances, qui vous ont amené à conclure qu'il s'agissait du choix idéal pour l'Arduino Nano ESP32 ?
Martino Facchin : l'ESP32-S3 est une bonne puce. Sa consommation d'énergie n'est pas incroyablement faible, mais elle n'est pas mauvaise. Les caractéristiques de consommation ultra-faible existent si vous voulez vraiment les utiliser, mais nous n'allons pas dire à nos utilisateurs de tout mettre en veille profonde et d'utiliser la très faible consommation, même si c'est agréable de voir que ces choses existent. Je dirais que la performance n'est pas la principale raison d'être de la famille Nano. C'est plutôt la facilité d'utilisation. Nous avons eu l'occasion d'effectuer une comparaison avec le Portenta H7, par exemple, en ce qui concerne les capacités d'apprentissage automatique. Le Portenta H7 est environ sept fois plus rapide que l'ESP32, même à des vitesses d'horloge comparables. L'ESP32 a une vitesse d'horloge deux fois inférieure à celle de le Portenta H7. Le Nano existe pour la facilité d'utilisation, l'environnement, la disponibilité des bibliothèques, le facteur de forme, etc.

Elektor : compte tenu des capacités du système d'exploitation en temps réel de l'ESP32, comment cela est-il pris en compte dans la conception de l'ESP32 et dans son potentiel multitâche ?
Alessandro Ranellucci : je dirais qu'il s'agit d'un besoin croissant, avec comment utiliser plusieurs cœurs, le multitâche, etc. Ce besoin n'est pas actuellement très demandé par les *makers*, mais un écosystème devra être mis en place.
Martino Facchin : nous avons fait un effort de standardisation il y a un peu plus d'un an, avec ce que l'on appelle les threads Arduino. Si vous regardez sur GitHub, nous avons essayé de développer cette idée d'avoir des threads cachés derrière différents onglets dans votre EDI. Ainsi, vous avez un onglet avec `setup()` et `loop()`, comme d'habitude. Ensuite, vous avez un autre onglet avec un autre `setup()` et `loop()`, et cela représente votre deuxième thread. Ensuite, il y a un troisième, un quatrième, tout ce qui est nécessaire. Il reste le problème de la synchronisation des variables, où vous avez la restriction habituelle du *RTOS* (système d'exploitation en temps réel) qui vous oblige à utiliser des noms de variables différents entre deux threads. Nous l'avons résolu à un niveau supérieur. Nous avons donc caché cette complexité en utilisant *Mbed*, mais il ne s'agissait pas en fait d'un problème propre à *Mbed*. Nous voulions le porter sur plusieurs systèmes d'exploitation.
Si cela a réussi, la communauté des *makers* n'a pas du tout réagi à ce projet. Ce n'est donc probablement pas quelque chose dont les *makers* ont vraiment besoin. Le threading, c'est bien, mais cela complique aussi les choses. Et en même temps, bien sûr, vous pouvez utiliser FreeRTOS. Vous pouvez faire ce que vous voulez si vous êtes assez compétent, mais nous nous n'insistons pas pour cela, car nous avons toujours privilégié la facilité d'utilisation.

VF : Laurent Rauber — 230524-04

Produit

› Arduino Nano ESP32
www.elektor.fr/20562



ARDUINO CLOUD

What Arduino Cloud is

Develop from anywhere

- **NO CODE**
  With ready-to-use templates
- **LOW-CODE**
  Automatically generated sketches
- **FULL ARDUINO EXPERIENCE**
  Either offline with the UDE2 or online with the Cloud Editor
- **STORE YOUR SKETCHES ONLINE**
  Use your code in your favourite Arduino development environment

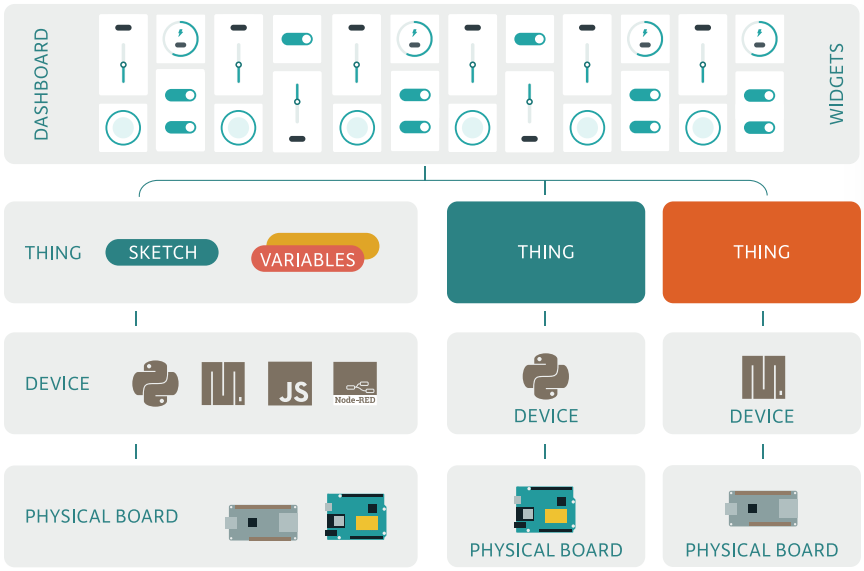
Program/Deploy

- **CABLE**
  Traditional USB programming
- **OVER-THE-AIR (OTA) UPDATES**
  Deploy your firmware wirelessly to your devices
- **MASS SCALE & AUTOMATION**
  With the Arduino Cloud CLI

Monitor & Control

- **CUSTOM DASHBOARDS**
  Using drag and drop widget
- **INSIGHTFUL WIDGETS**
  Interact with the devices and get real-time and historical data with dozens of widgets
- **MOBILE APP**
  Visualise your data in real-time from your phone with the IoT remote app

How does it work?



Compatible hardware

WITHIN ARDUINO DEVELOPMENT ENVIRONMENTS

ARDUINO
Cloud Applications can be developed using the Arduino Cloud Editor or Arduino IDE 2.

ESP32/ESP8266
+70% Of Arduino Cloud active users use ESP-based boards.

OUTSIDE ARDUINO DEVELOPMENT ENVIRONMENTS

Use your favourite programming environment and language to connect your devices to the Cloud.

Third party platform integration

TRIGGER ACTIONS ON THIRD PARTY PLATFORMS

Connect your Arduino Cloud devices to external platforms such as IFTTT, Zapier and Google Services using webhooks and unlock endless possibilities.

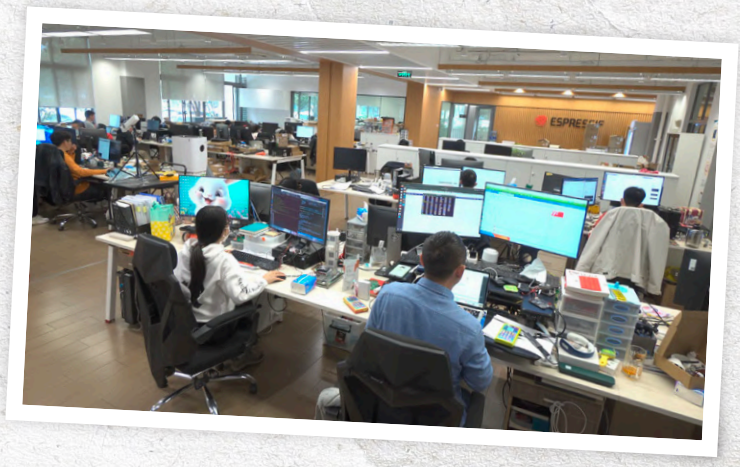
Seamlessly integrate your IoT devices with over 2 000 apps, enabling tasks like recieving phone notifications, automating social media updates, streamlining data logging to external files, creating calendar events, or sending e-mail alerts.

Get 30% off on the yearly Maker plan with code ELEKTOR30*

cloud.arduino.cc/elektor

innovation des puces IdO

les idées de ESPRESSIF



La communauté d'Elektor est composée d'ingénieurs, de makers et d'étudiants intéressés par un large éventail de sujets liés à l'électronique. Plusieurs de ces membres ont récemment rédigé des questions pour les ingénieurs et les dirigeants d'Espressif.

Teo Swee Ann a fondé Espressif en 2008. Qu'est-ce qui l'a poussé à le faire ? Quelle était sa vision de l'entreprise à cette époque comment a-t-elle évolué depuis ? - Margriet Debeij (Pays-Bas)

Lors de la création d'Espressif, l'objectif initial de Teo était de créer un produit permettant d'automatiser la conception analogique. Cependant, l'entreprise a découvert sa véritable vocation dans la conception de puces dédiées pour l'internet des objets (IdO). Teo a établi une stratégie claire qui met l'accent sur la création de solutions IdO innovantes, abordables et économes en énergie. L'importance qu'il accorde à l'innovation a joué un rôle déterminant dans le succès d'Espressif Systems. Il a également compris la valeur de l'engagement et a activement soutenu la croissance de la communauté des développeurs d'Espressif. En favorisant un écosystème de collaboration et de partage des connaissances, il a veillé à ce qu'Espressif reste à la pointe de l'innovation. Alors que le paysage technologique évoluait, Teo a conduit Espressif Systems à adopter des technologies émergentes telles que l'intelligence artificielle, l'apprentissage automatique et l'informatique de pointe.

Comment l'ESP-NOW se différencie-t-il du ZigBee ou du Matter ? - Clemens Valens (France)

L'ESP-NOW et le ZigBee peuvent tous deux être utilisés pour construire des dispositifs IdO à faible consommation, mais l'alliance de l'ESP-NOW et du Wifi peut offrir un large éventail de possibilités que les applications peuvent exploiter pour fournir des solutions uniques. L'ESP-NOW peut également offrir une plus grande portée et une plus grande largeur de bande, en particulier dans les environnements extérieurs. Il fonctionne même en l'absence d'un réseau d'infrastructure Wifi et est un protocole exclusif d'Espressif, tandis que Matter et ZigBee sont des normes définies par la Connectivity Standards Alliance. Matter est un protocole de couche d'application qui prend en charge Thread et le Wifi en tant que protocoles de communication sans fil. Le ZigBee et l'ESP-NOW peuvent faire partie du réseau Matter grâce à l'utilisation d'un pont Matter. Nous avons des solutions pour le pont Matter ESP-NOW et le pont Matter Zigbee.

Espressif a une collaboration avec Arduino qui a donné naissance à la carte Arduino Nano ESP32. Est-il possible que vous vous associiez à Raspberry Pi et que vous proposiez un produit commun ? - Ferdinand te Walvaart (Pays-Bas)

Nous apprécions notre collaboration avec Arduino. Elle inclut le support officiel de l'EDI Arduino (environnement de développement intégré) pour l'ESP-IDF à travers les différents chipsets d'Espressif. Pour aller plus loin, nous avons annoncé l'Arduino UNO R4 Wifi et l'Arduino Nano ESP32, tous deux incluant le module ESP32-S3. Nous pensons que nous partageons tous, Arduino, Raspberry Pi et Espressif, une communauté et un état d'esprit axé sur l'open-source qui fait de la collaboration une voie naturelle. Nous pensons que Raspberry Pi se concentre principalement sur le segment des microprocesseurs à ce jour, et nous n'avons pas de produit à offrir dans ce segment. Cela dit, avec le lancement de la série Pico, il pourrait y avoir la possibilité de collaborer et de travailler ensemble à l'avenir.

Quels sont les défis posés par le marketing auprès des professionnels par rapport à un public plus général ? - Erik Jansen (Pays-Bas)

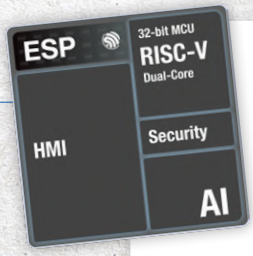
Les ingénieurs sont un groupe qui s'intéresse de très près aux détails techniques. Nous devons donc trouver le juste milieu entre la précision et la compréhension. Notre objectif est de leur montrer comment nos produits répondent à leurs besoins et défis spécifiques. Nous veillons donc à leur fournir toutes les informations techniques dont ils ont besoin pour prendre des décisions en connaissance de cause. Nous savons également qu'ils passent du temps sur des forums et des plateformes spécialisés, et nous voulons donc être là où ils sont. Notre marketing doit faire l'effet d'un « phare » dans ces espaces. Les ingénieurs apprécient les décisions logiques, c'est pourquoi notre message se concentre sur la façon dont nos produits apportent de la valeur, des avantages et des innovations techniques à leurs projets. Il s'agit de parler leur langage et de leur montrer que nous sommes là pour rendre leur parcours plus fluide et passionnant.

Des compétences particulières sont nécessaires pour développer et gérer les technologies IA/IdO. Comme il y a une pénurie de professionnels qualifiés pour tous les marchés techniques, cela limite-t-il la croissance des produits et des revenus d'Espressif ? - Ruud Schaay (Pays-Bas)

Cette pénurie pose effectivement des problèmes. Nos produits s'adressent aux développeurs, en proposant des plateformes agréables soutenues par des ressources étendues. Cela permet à des passionnés d'horizons divers de s'engager efficacement dans les technologies de l'IA/IdO. Nous collaborons également activement avec des établissements d'enseignement du monde entier par l'intermédiaire de nos équipes et de nos communautés en ligne, en proposant des ateliers et des ressources pour combler le déficit de compétences. Cette approche soutient non seulement la croissance des produits d'Espressif, mais contribue également au paysage technologique dans son ensemble. L'importance que nous accordons à l'accessibilité et à l'éducation permet aux professionnels en herbe de s'épanouir, ce qui favorise une culture de l'innovation et du développement des compétences. Bien que la pénurie de talents soit préoccupante, nous nous engageons à doter les individus des outils et des connaissances nécessaires pour prospérer dans l'arène de l'IA/IdO, propulsant à la fois Espressif et l'industrie vers l'avant.

Combien d'ingénieurs travaillent pour Espressif ? Parlez-nous de la culture d'entreprise. Quel est l'environnement de travail chez Espressif ? - C. J. Abate (États-Unis)

Espressif est au sens propre une entreprise axée sur la technologie. Plus de 75% de notre personnel est composé d'ingénieurs, soit environ 450 employés dans les différents départements de recherche et développement. Nos employés viennent d'horizons divers et trente nationalités différentes se côtoient dans nos bureaux situés dans cinq différents pays, dont la Chine, l'Inde, la Tchéquie, Singapour et le Brésil. Nous sommes fiers d'avoir une culture d'entreprise qui encourage les employés à explorer des idées innovantes, à prendre des risques calculés et à améliorer continuellement nos produits et services. Tout le monde est accessible, un état d'esprit de collaboration est instillé, et les employés sont encouragés à poursuivre ce en quoi ils croient et ce qui les passionne. Cela permet de créer des solutions différenciées et innovantes.



Espressif semble avoir l'une des offres RISC-V les plus solides pour les microcontrôleurs. Tous les futurs appareils utiliseront-ils RISC-V ? - Stuart Cording (Allemagne)

Nous veillons à ce que notre matériel soit largement accessible et à ce que nos logiciels soient disponibles en open-source. Il s'agit d'une philosophie fondamentale pour Espressif. L'adoption des processeurs RISC-V dans notre gamme de microcontrôleur était une progression naturelle. Nous sommes fiers de ce que nous avons pu réaliser en offrant à nos clients une transition simple entre les différents produits dans le même cadre de développement. Nous sommes déterminés à poursuivre l'adoption de RISC-V dans notre éventail et notre premier produit RISC-V à double cœur (ESP32-P4) sera bientôt sur le marché. Cela nous offre une grande souplesse de mise en œuvre et enrichit notre propriété intellectuelle, ce qui est essentiel pour fournir des solutions plus avancées et plus abordables à nos clients.

Alors que la 5G fait évoluer le marché vers des réseaux déterministes, avec une vitesse de calcul élevée et une réponse en temps réel, il devient également très sensible à la consommation d'énergie. Comment Espressif s'y prend-elle pour résoudre cette équation apparemment insolvable ? - Roberto Armani (Italie)

Il est vrai que les économies d'énergie et l'empreinte carbone deviennent des objectifs de plus en plus importants pour nos clients et leurs consommateurs finaux. Nous en sommes conscients et continuons à améliorer nos produits et nos solutions pour répondre à ce besoin croissant. Par exemple, nous avons réorganisé notre approche du mode veille légère qui, dans les produits les plus récents tels que l'ESP32-C6 et l'ESP32-H2, permet à l'application de mettre hors tension la plupart des périphériques, ce qui peut réduire la consommation de courant de 80%. Nous proposons également des solutions de produits pour répondre aux besoins croissants en matière de faible consommation d'énergie. Par exemple, l'interrupteur ESP-NOW switch basé sur l'ESP32-C2, qui permet d'appuyer 10 000 fois sur l'interrupteur avec une seule pile bouton. Nous continuerons à innover et à nous concentrer sur la réduction de la consommation globale de nos produits.

Les solutions Espressif sont utilisées dans des milliers de conceptions électroniques, des applications professionnelles aux projets maisons. Votre équipe d'ingénieurs doit avoir une petite liste de projets préférés développés par la communauté. Pouvez-vous nous en partager trois des plus passionnants ou innovants basés sur Espressif que vous avez vus naître au sein de la communauté ? - C. J. Abate (États-Unis)

Il y a de nombreux projets captivants au sein de notre communauté inventive. En effet, cette édition présente une sélection de ces projets remarquables, couvrant à la fois le matériel et les microprogrammes. Parmi eux figurent des cartes d'évaluation basées sur l'ESP32, pas plus grandes que des piles bouton, ainsi que des projets ingénieux faisant tourner Linux sur l'ESP32-S3. En outre, nous avons rencontré des projets simulant des consoles de jeu sur des produits ESP et fabriquant des horloges numériques synchronisées avec plusieurs cartes ESP. Certains se sont même lancés dans la création de cartes VGA avec des produits Espressif ou dans le portage de pilotes audio sur l'ESP32. Il serait injuste de se limiter à trois projets remarquables.

Combien de cartes/modules avez-vous vendus jusqu'à présent ? - Muhammed Söküt (Allemagne)

Nous avons lancé notre premier SoC IdO en 2014 avec la sortie du populaire ESP8266. Il s'agissait d'un produit IdO qui a révolutionné l'industrie en intégrant la connectivité sans fil et le microcontrôleur sur la même puce. L'ESP8266 et le produit phare ESP32, sorti en 2016, ont conduit à une vente cumulée de 100 millions d'unités en 2017. Depuis lors, nous avons continué à susciter de l'intérêt et avons évolué en tant qu'entreprise, en proposant de nouveaux produits et des solutions innovantes. Rien qu'en 2021, nous avons expédié plus de 200 millions d'unités. Cumulativement, nous avons récemment annoncé avoir dépassé le milliard d'unités expédiés.

Le succès d'Espressif a commencé avec l'ESP8266, abordable et facile à utiliser, qui servait principalement à établir des connexions Wifi. Aujourd'hui, les puces et les SoC d'Espressif sont souvent utilisés pour la connexion Wifi, avec d'autres contrôleurs sur la même carte hébergeant l'application principale. L'ESP32-P4 sort du lot, car il s'agit d'un processeur performant à lui tout seul. Espressif ira-t-il encore plus loin dans cette direction ? Verrons-nous, par exemple, un ESP64 ? - Jens Nickel (Allemagne)

Nous avons créé l'ESP32-P4 après avoir constaté que de nombreuses conceptions utilisaient l'ESP32 sans aucune connectivité, et nous pensions pouvoir fournir une solution plus puissante, mais optimisée, pour répondre à ce besoin. Nos optimisations de la mise en œuvre du RISC-V nous ont permis de créer un microcontrôleur multicœur à hautes performances avec une extension IA. Nous avons continuellement amélioré l'ensemble des périphériques. Grâce à tout cela, nous sommes bien équipés pour créer rapidement de nouvelles définitions de systèmes sur puce (SoC). Toutefois, il est peut-être trop tôt pour se prononcer sur les offres spécifiques de microcontrôleurs que nous proposerons à l'avenir.

Quels sont vos projets pour les futurs microcontrôleurs (par exemple, les successeurs de l'ESP32) ? Peut-on s'attendre à des éléments tels que l'USB intégré, le Wi-Fi 5 GHz ou le Bluetooth 5 x ? - Dr. Thomas Scherer (Allemagne)

Depuis l'ESP32 en 2016, nous avons lancé une série de produits sur les séries ESP32-C, ESP32-S et, plus récemment, ESP32-H et ESP32-P. Ces produits sont destinés à répondre aux divers besoins des différentes applications et industries. La plupart de ces produits nouvellement lancés prennent en charge la technologie Bluetooth Low Energy 5. L'ESP32-S2/S3 et l'ESP32-C6 offrent l'USB OTG. L'ESP32-H2, dont la production de masse a commencé au début de cette année, prend en charge la connectivité 802.15.4, ce qui lui permet d'être utilisé dans des applications connectées par Zigbee et Thread. Nous avons également annoncé récemment l'ESP32-C5, qui prendra en charge le Wi-Fi 6 bibande (2,4 GHz et 5 GHz) et sera bientôt disponible. Le futur ESP32-P4 prendra en charge certaines interfaces homme-machine (IHM) avancées et des périphériques médias tels que MIPI-DSI et MIPI-CSI avec un processeur de traitement d'image intégré (ISP), un encodeur H.264, etc. et permettra une pléthore d'applications nouvelles et diverses.

Les produits d'Espressif sont largement utilisés dans des produits tels que les appareils électroménagers, les ampoules, les haut-parleurs intelligents, l'électronique grand public et les terminaux de paiement. Espressif prévoit-il d'étendre l'utilisation de ses produits au-delà ? - Alina Neacșu (Allemagne)

Chez Espressif, notre objectif est de démocratiser l'accès aux technologies IdO et de diversifier le marché avec des solutions de connectivité sans fil innovantes, centrées sur les développeurs et abordables. La série de puces ESP32 continuera d'évoluer et offrira une meilleure connectivité, une puissance de calcul accrue, une sécurité renforcée, un ensemble de périphériques de plus en plus performants et une consommation d'énergie réduite. De plus, Espressif s'est également développé en tant que fournisseur de solutions complètes, identifiant les difficultés des clients et les abordant efficacement avec des solutions qui vont au-delà des kits de développement matériels et logiciels (SDK) traditionnels. Les modules ESP RainMaker, ESP Insights et ESP ZeroCode en sont de bons exemples. Ces solutions et produits ne nous limitent pas à un segment ou à une industrie particulière.

Le secteur industriel exige des microcontrôleurs fiables et robustes. Comment Espressif compte-t-il tirer parti de ses avancées et de ses capacités pour devancer ses concurrents et répondre aux besoins spécifiques du marché industriel ? - Saad Imtiaz (Pakistan)

Le marché industriel requiert quelques exigences spécifiques qui ne sont pas courantes dans le domaine des consommateurs. Il s'agit de conditions de fonctionnement plus strictes, telles que des températures plus élevées, une plus grande fiabilité (faible taux de défaillance) et une plus grande longévité, c'est-à-dire que le produit doit pouvoir être utilisé pendant plusieurs années. Nos modules et SoC supportent des températures allant jusqu'à 105°C, ce qui les rend aptes à être utilisés dans la plupart des applications industrielles. Nos produits sont soumis à des tests de fiabilité approfondis afin de garantir à nos clients les taux de défaillance les plus bas possibles.

De nombreux produits Espressif contiennent un émetteur sans fil d'un type ou d'un autre, souvent pour les bandes ISM et incorporant une antenne embarquée. Comment Espressif s'assure-t-il que la puissance RF, la largeur de bande et les émissions parasites sont conformes aux spécifications et aux limites légales ? - Jan Buiting (Pays-Bas)

Lorsque nous concevons nos produits, nous tenons compte des lignes directrices et des spécifications relatives aux différents protocoles et supports de communication définis par les organes directeurs et les alliances, et nous les respectons scrupuleusement. Nos produits et modules sont soumis à des tests et certifications réglementaires dans des laboratoires tiers, ce qui garantit leur conformité aux spécifications et limites définies par les différentes régions géographiques. Vous trouverez ces certifications sur notre site web. Elles peuvent être utilisées par nos clients pour accélérer les processus de certification de leurs produits.

VF : Maxime Valens — 230604-04

rationaliser la **programmation des microcontrôleurs** avec **ESP-IDF Privilege Separation**

Harshal Patil, Espressif

Cet article présente la séparation des privilèges dans les applications de microcontrôleurs avec l'ESP-IDF d'Espressif Systems. Cette méthode permet de répartir les micrologiciels dans un noyau protégé et dans une application utilisateur, ce qui simplifie le développement. Il décrit les étapes à suivre pour commencer, afin de rendre le développement des microcontrôleurs plus efficace.

En général, les applications sur microcontrôleurs (MCU) sont développées sous forme de micrologiciels monolithiques. Cependant, dans un système d'exploitation à usage général, il existe deux modes de fonctionnement : le mode noyau et le mode utilisateur. En mode noyau, le programme a un accès direct et illimité aux ressources du système, alors qu'en mode utilisateur, le programme d'application n'a pas d'accès direct aux ressources du système. Pour accéder aux ressources, un appel système doit être effectué.

L'idée principale de cette séparation est de faciliter le développement de l'application finale sans se préoccuper des changements sous-jacents dans le système, tout comme les applications de bureau ou mobiles : le système d'exploitation sous-jacent gère les fonctionnalités critiques, et l'application finale peut utiliser l'interface présentée par le système d'exploitation.

Séparation de privilèges de l'ESP

En général, toute application ESP-IDF sur un SoC Espressif est construite sous la forme d'un seul micrologiciel monolithique sans aucune séparation entre les composants de noyau centraux (système d'exploitation, réseau, etc.) et la logique « applicative » ou « commerciale ». Dans le cadre de la séparation des privilèges de l'ESP, nous divisons l'image du micrologiciel en deux fichiers binaires séparés et indépendants : l'application protégée et l'application utilisateur.

Pour commencer

Commençons à construire notre premier projet, Blink. La LED clignotante est l'équivalent de « Hello World » du monde des systèmes embarqués, démontrant la chose la plus simple que vous pouvez réaliser avec un microcontrôleur pour voir une sortie réelle. C'est parti !

Étape 0 : matériel requis pour le projet

- Une carte de développement ESP32-C3 ou ESP32-S3 avec une LED intégrée. Il est possible d'utiliser les kits de développement ESP32-C3-DevKitM-1 ou ESP32-S3-DevKitC-1. Nous choisirons l'ESP32-C3-DevKitM-1 pour notre projet.
- Un câble USB 2.0 (Standard-A vers Micro-B)
- Un ordinateur fonctionnant sous Windows, Linux ou macOS

Étape 1 : obtenir le projet ESP Privilege Separation

- Clonez le dépôt du projet ESP Privilege Separation.

```
git clone https://github.com/espressif/esp-privile-
ge-separation.git
```

Étape 2 : configuration de l'environnement ESP-IDF

- Clonez le dépôt du projet ESP-IDF.

```
git clone -b v4.4.3 --recursive https://github.com/
espressif/esp-idf.git
```

- Installer les outils.

```
cd esp-idf
./install.sh
```

- Définissez les variables d'environnement.

```
source ./export.sh
```

- Appliquez le correctif ESP Privilege Separation spécifique à ESP-IDF.

```
git apply -v /idf-patches/
privilege-separation_support_v4.4.3.patch
```

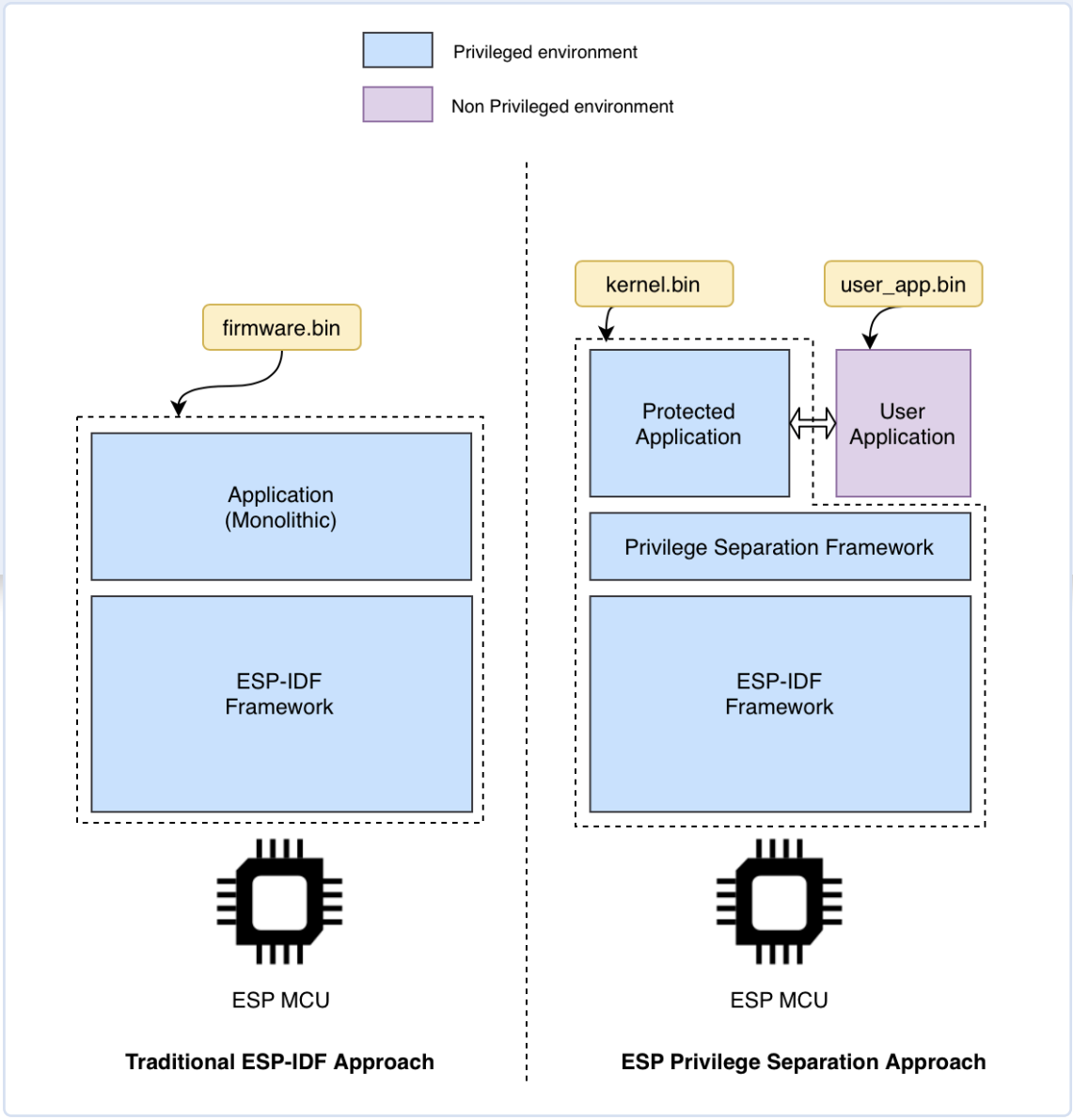


Figure 1. Comparaison d'un ESP-IDF traditionnel composé d'un micrologiciel monolithique unique par rapport au cadre ESP Privilege Separation, où l'image du micrologiciel est divisée. (Source : Espressif)

Étape 3 : exécution de l'exemple *Blink*

- Construisez l'exemple.

```
cd /examples/blink
idf.py set-target esp32c3
idf.py build
```

- Flashez et exécutez l'exemple.

```
idf.py flash monitor
```

Plonger dans l'implémentation

Notre LED clignote maintenant, mais pourquoi ce projet se distingue-t-il de tous les autres projets de clignotement ? Réponse : La séparation des privilèges. Essayons de comprendre l'implémentation :

- L'exemple est divisé en deux applications : l'application sécurisée et l'application utilisateur.
- L'application sécurisée recherche une partition utilisateur dans la table des partitions et charge l'application dans les sections définies par l'utilisateur.
- Elle configure ensuite les zones mémoire pour la région avec des privilèges inférieurs (`WORLD1`) et fournit l'accès tel que configuré par le développeur.
- Enfin, elle lance une tâche distincte à faible privilège avec le point d'accès de l'utilisateur.
- Une fois l'application utilisateur chargée, une tâche est lancée pour faire clignoter la LED connectée à GPIO8 (ESP32-C3-DevKitM-1).

Intégration avec ESP RainMaker

Nous avons maintenant notre devkit de clignotement de LED, mais nous ne sommes pas encore en mesure de contrôler le clignotement de la LED. Pour le qualifier d'appareil véritablement « connecté » et contrôler la LED, nous devons l'intégrer à ESP RainMaker. ESP RainMaker est un logiciel Cloud AIoT léger qui permet aux utilisateurs de construire, développer et déployer des solutions AIoT personnalisées avec un minimum de code et un maximum de sécurité. Commençons par construire l'exemple `rmaker_switch` présent dans le projet ESP Privilege Separation.

Étape 0 : exigences

- ESP-IDF pour ESP Privilege Separation a déjà été configuré avant la réalisation de tout exemple.
- Application mobile ESP RainMaker (Android/iOS).
- Réseau Wifi.

Étape 1 : installation d'ESP RainMaker

- Clonez le dépôt du projet ESP RainMaker.

```
git clone --recursive https://github.com/espressif/
esp-rainmaker
git checkout 00bcf4c0c30d96b8954660fb396ba313fb6c886f
export RMAKER_PATH=
```

Étape 2 : exécution de l'exemple rmaker_switch

- Construisez l'exemple.

```
cd /examples/rmaker_switch
idf.py set-target esp32c3
idf.py build
```

- Flashez et exécutez l'exemple.

```
idf.py flash monitor
```

Étape 3 : approvisionnement de l'appareil

- Une fois l'exemple exécuté, un code QR apparaît dans le terminal. Le code QR peut être utilisé pour l'approvisionnement sur Wifi.
- Connectez-vous à l'application mobile ESP RainMaker et cliquez sur *Add Device*. Cela ouvrira l'appareil photo pour scanner le code QR.
- Suivez le processus d'approvisionnement pour que votre appareil puisse se connecter à votre réseau Wifi.

Étape 4 : contrôle du commutateur de LED

- Enfin, vous verrez un appareil *Switch* ajouté à l'écran d'accueil de l'application mobile.
- Vous pouvez maintenant essayer de basculer l'icône du commutateur pour contrôler votre LED.

Mais comment avons-nous réalisé cette intégration ESP RainMaker transparente ? Nous introduisons un composant `rmaker_syscall` dans notre exemple Blink, qui révèle les appels système pour le framework ESP Rainmaker, et, lorsque l'application utilisateur démarre, il initialise le service ESP Rainmaker dans l'application protégée, et crée un appareil de commutation ESP RainMaker. Comme le composant ESP RainMaker est placé dans l'application protégée, des appels système sont nécessaires pour toutes les API publiques qu'il expose. Les fonctions d'extensibilité simples d'ESP Privilege Separation permettent d'ajouter des appels système personnalisés spécifiques à l'application, et cette couche utilise les données stockées dans le contexte de l'appareil pour exécuter les rappels de lecture et d'écriture dans l'espace utilisateur.

Mises à jour OTA du micrologiciel avec ESP Privilege Separation

La mise à jour du micrologiciel à distance (OTA) est l'une des fonctions les plus importantes pour tout appareil connecté. Elle permet aux développeurs de fournir de nouvelles fonctions et des corrections de bogues en mettant à jour l'application à distance. Dans ESP Privilege Separation, il y a deux applications – *protected_app* et *user_app*, pour lesquelles le framework permet des mises à jour indépendantes des deux fichiers binaires. L'application protégée, ayant des privilèges plus élevés, peut être mise à jour en suivant simplement l'interface OTA normale fournie par ESP-IDF, tandis que la mise à jour OTA de l'application utilisateur ayant des privilèges moins élevés est rendue possible car l'ESP Privilege Separation expose un appel système pour l'application utilisateur, `usr_esp_ota_user_app()`, pour exécuter le processus OTA.
Essayons l'exemple de mise à jour OTA de l'application utilisateur du projet ESP Priviledge Separation.

Étape 0 : exigences

- ESP-IDF pour ESP Privilege Separation a déjà été configuré avant la construction de tout exemple.
- Une URL publique de l'image de la nouvelle application utilisateur hébergée.

Étape 1 : exécution de l'exemple `rmaker_switch`

- Construisez l'exemple.

```
cd /examples/esp_user_ota
idf.py set-target esp32c3
idf.py build
```

- Flashez et exécutez l'exemple.

```
idf.py flash monitor
```

Étape 2 : démarrer la mise à jour OTA

Saisissez l'URL de l'OTA lorsque l'application utilisateur accepte l'OTA en utilisant la commande `user-ota` via la console. Cela déclenche une OTA et la nouvelle application utilisateur démarre une fois l'OTA terminée. Nous avons maintenant un appareil connecté avec des fonctions clés prêtes à être déployées.
Vous pouvez scanner le code QR pour regarder l'exemple ESP Privilege Separation, démontrant un cas d'utilisation réel de mise à jour OTA d'une application utilisateur à l'aide d'ESP Rainmaker et d'ESP Privilege Separation.

Comme nous l'avons dit, le succès d'un produit est déterminé par ses utilisateurs, et c'est vous ! Nous aimons savoir ce que vous pensez, alors n'hésitez pas à nous faire part de vos expériences, de vos suggestions et de vos commentaires. Vos commentaires alimentent notre innovation et nous aident à affiner notre produit pour mieux répondre à vos besoins. Visitez notre site web [1], envoyez-nous un courriel, connectez-vous avec nous sur les réseaux sociaux pour partager vos idées ou ouvrez un nouveau problème dans le dépôt GitHub du projet [2]. Si vous avez un besoin spécifique qui, selon vous, s'inscrit bien dans ce cadre, ou si la résolution de tels problèmes vous passionne, nous serions heureux de discuter avec vous d'une éventuelle collaboration.

230598-04

Questions ou commentaires ?

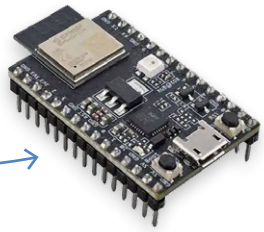
Envoyez un courriel à l'auteur (harshal.patil@espressif.com) ou contactez Elektor (redaction@elektor.fr).

À propos de l'auteur

Harshal Patil est ingénieur logiciel chez Espressif Systems depuis un an. Il est passionné par la résolution de problèmes concrets en construisant des systèmes sécurisés et connectés. Passionné de technologie et de sport, il aime explorer de nouveaux gadgets innovants et jouer au football dans son temps libre.

Produits

- **ESP32-C3-DevKitM-1**
  www.elektor.fr/20324
- **ESP32-S3-DevKitC-1**
  www.elektor.fr/20697



LIENS

[1] Espressif : https://espressif.com
[2] Le dépôt du projet sur GitHub : https://github.com/espressif/esp-privilege-separation/issues

SDK

ESPRESSIF

ESP-IDF
Espressif IoT Development Framework

ESP-IDF est le SDK de développement officiel d'Espressif qui prend en charge tous les SoC des séries ESP32, ESP32-S, ESP32-C et ESP32-H. C'est le meilleur point de départ pour construire tout ce qui ne nécessite pas un kit de développement dédié. L'ESP-IDF comprend le noyau FreeRTOS, des pilotes de périphériques, des piles de stockage flash, des piles de protocoles Wi-Fi, Bluetooth, BLE, Thread, Zigbee, une pile TCP/IP, TLS, des protocoles de niveau des applications (HTTP, MQTT, CoAP) et de nombreux autres composants et outils logiciels.
Il fournit également des fonctions de haut niveau couramment utilisées, telles que l'OTA et le provisionnement du réseau. En plus du support de la ligne de commande, il prend en charge les intégrations Eclipse et VSCode IDE. Notez que vous trouverez de nombreux exemples d'applications pour vous aider à démarrer rapidement. ESP-IDF a une période de support et de maintenance bien définie, et la dernière version stable est recommandée pour le développement de nouveaux projets.

https://github.com/espressif/esp-idf

un serveur de reconnaissance de la parole open-source...

...et l'ESP BOX

Kristian Kielhofner (États-Unis)

Beaucoup d'entre nous apprécient l'Alexa Echo et les appareils similaires, mais toujours avec des inquiétudes concernant la vie privée et l'utilisation des données. La plateforme Willow est une alternative gratuite et open-source. Outre le Willow Inference Server, il faut une interface utilisateur vocale de haute qualité pour capturer l'audio et la nettoyer. Avec un prix attractif, l'ESP BOX d'Espressif procure non seulement les microphones et la puissance de traitement audio, mais elle s'accompagne également d'un puissant écosystème logiciel.

Depuis le lancement de la plateforme Alexa il y a huit ans, Amazon a vendu plus de 500 millions d'appareils Echo. En revanche, il demeure des inquiétudes et des controverses concernant la vie privée, l'utilisation des données et les tentatives de plus en plus ennuyeuses d'Amazon pour faire débourser davantage aux utilisateurs d'Echo. Willow [1] est une plateforme qui procure une interface utilisateur vocale concurrente d'Alexa, conviviale pour les makers, gratuite et open source, sans sacrifier la qualité ni se ruiner.

Matériel

Raspberry Pi

Une interface utilisateur vocale de haute qualité doit s'intégrer dans le monde physique et interagir avec lui. L'écosystème open-source a tenté depuis longtemps de créer des interfaces utilisateur vocales en utilisant le Raspberry Pi, divers microphones, etc. (**figure 1**), mais cette approche présentait des inconvénients significatifs :

- Le prix. En achetant un Raspberry Pi, un écran LCD tactile, un réseau de microphones de haute qualité, un haut-parleur, un boîtier adapté, etc., vous atteignez un prix au moins trois fois supérieur à celui d'un appareil Echo (50 € ou moins). Cette approche nécessite l'approvisionnement des composants, un assemblage minutieux, la création d'un boîtier adapté, le développement de logiciels, etc. Répéter cette opération pour plusieurs appareils dans votre entourage augmente considérablement le temps et le coût nécessaires.
- La disponibilité. La situation s'est améliorée récemment mais, y compris pour le Raspberry Pi, la chaîne d'approvisionnement de ces composants a connu des problèmes.

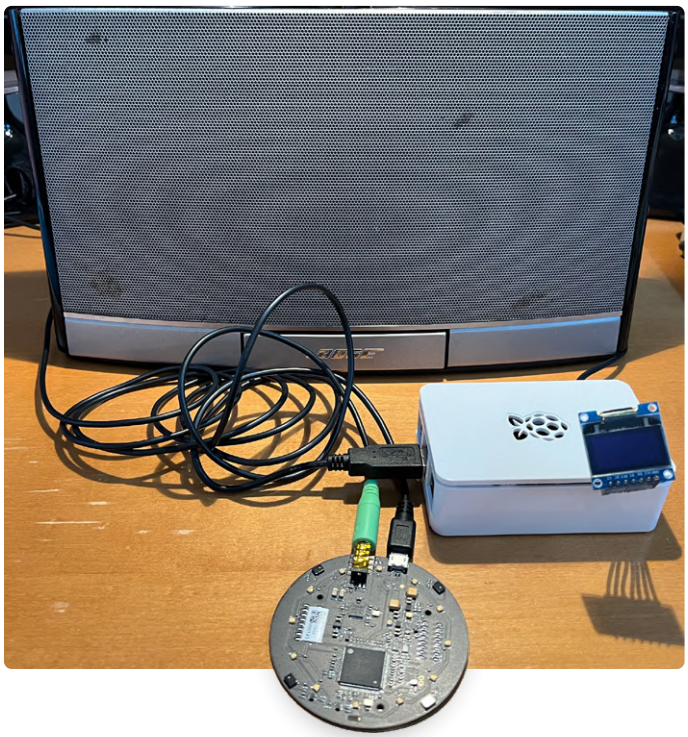


Figure 1. D'après les tentatives antérieures de l'auteur avec le Raspberry Pi.

Au cours des dernières années, obtenir ces composants tenait de l'impossible ou de l'inutilement coûteux à cause des coûts de distribution.

- La gestion. Les solutions à base de Raspberry Pi nécessitent souvent une distribution Linux complète qui prenne en charge les composants matériels, ainsi qu'un ensemble conséquent de logiciels nécessaires pour une interface utilisateur vocale. La gestion d'une demi-douzaine d'ordinateurs Linux peut s'avérer difficile pour de nombreux utilisateurs.
- La qualité. Amazon (et d'autres) ont investi des ressources considérables dans la conception d'une interface utilisateur vocale capable de fonctionner dans des environnements acoustiquement difficiles avec une grande variété de locuteurs humains. Il ne s'agit pas simplement d'installer un microphone et un haut-parleur sur un Raspberry Pi.
- L'écosystème. Une fois que vous avez obtenu une transcription précise d'une commande vocale, vous devez en faire quelque chose et fournir un retour d'information à l'utilisateur.
- La performance. Avec les implémentations de reconnaissance de la parole les plus optimisées, un Raspberry Pi 4 peut effectuer une reconnaissance presqu'en temps réel avec la qualité de modèle la plus basse. La précision laisse à désirer et le « temps réel » n'est tout simplement pas assez rapide - surtout si l'on considère qu'une erreur de transcription vous obligera à répéter la commande, ce qui élimine l'aspect pratique d'une interface utilisateur vocale.

Le **tableau 1** montre qu'un Raspberry Pi est légèrement plus rapide que la reconnaissance vocale en temps réel avec le plus bas niveau de qualité de modèle disponible.

Il est intéressant de noter que les modèles de reconnaissance de la parole ont une performance globale en temps réel plus élevée avec des segments de parole plus longs. Malheureusement, les commandes vocales des assistants vocaux sont très courtes, ce qui entraîne des pertes de performance significatives. Ce benchmark favorise en fait la performance de reconnaissance de la parole en temps réel du Raspberry Pi.

Nous connaissons et apprécions tous le Raspberry Pi – il est fantastique pour un large éventail d'applications et de cas d'usage. Malheureusement, un assistant vocal avec reconnaissance et synthèse de la parole n'en fait pas partie.

Comme le montre un exemple de capture d'écran (**figure 2**), la reconnaissance de la parole de Willow avec Willow Inference Server a pris 222 millisecondes de la fin du discours à la confirmation de l'action effectuée par le système domotique Home Assistant. Cela représente environ 25 % du temps nécessaire au Raspberry Pi 4 pour effectuer la reconnaissance de la parole seule, tout en utilisant un modèle de reconnaissance de la parole nettement plus précis que le « petit » modèle qui est à peine en temps réel sur le Raspberry Pi.

Comme beaucoup d'autres, j'ai tenté à plusieurs reprises avec différentes approches au fil des ans de créer une interface utilisateur vocale. Malheureusement, le résultat a toujours été le même : beaucoup de travail et de coûts pour une démo intéressante, mais totalement impraticable et inutilisable dans le monde réel à cause des problèmes mentionnés ci-dessus (plus d'autres).

Tableau 1. Performances de reconnaissance de la parole du Raspberry Pi 4.

Modèle	Taille du faisceau	Durée de la parole (ms)	Temps d'inférence (ms)	Rapport au temps réel
tiny	1	3.840	3.333	1,15×
base	1	3.840	6.207	0,62×
medium	1	3.840	50.807	0,08×
large-v2	1	3.840	91.036	0,04×

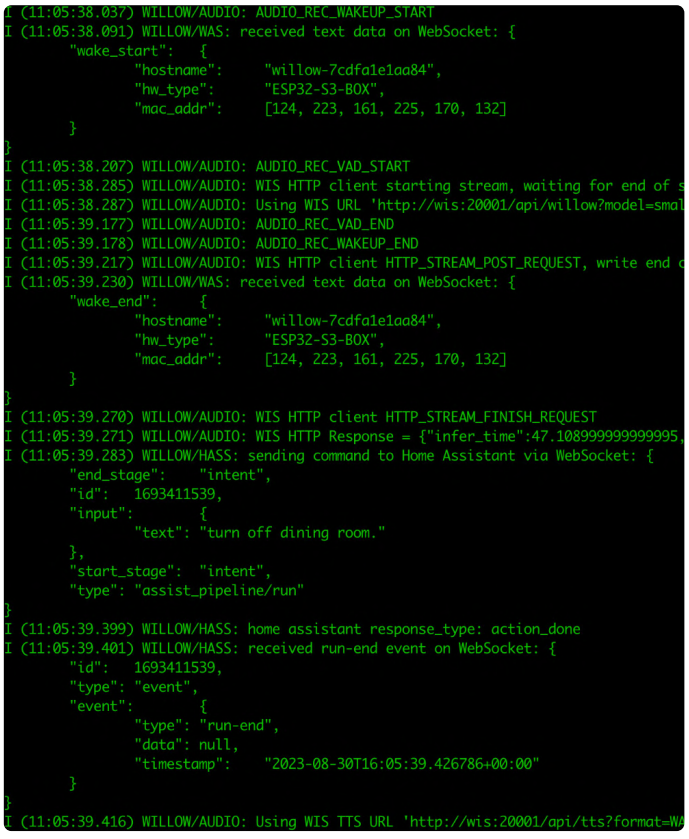


Figure 2. Performances de reconnaissance de la parole de Willow.

ESP BOX

J'ai alors découvert la plateforme de développement ESP BOX d'Espressif (**figure 3**). Comme beaucoup d'entre vous, j'utilise des appareils Espressif pour diverses applications depuis une dizaine d'années et je sais à quel point le matériel est robuste, riche en fonctionnalités, rentable et facilement disponible. Je sais également, grâce à des projets antérieurs, qu'Espressif fournit de nombreuses solutions logicielles et une documentation de grande qualité pour son matériel et ses logiciels.

L'ESP BOX d'Espressif est une plateforme de développement spécialement conçue pour ce cas d'usage. Pour environ 50€ chez votre détaillant ou distributeur d'électronique de bricolage préféré, vous obtenez :

Figure 3. L'ESP32-S3-BOX-3 est équipée d'un écran de 2,4 pouces, de deux microphones et d'un haut-parleur.

- Un ESP32-S3 avec 16 Mo de mémoire flash et 16 Mo de RAM connectée par SPI à haute vitesse
- Un écran tactile capacitif de 2,4 pouces
- Deux microphones (très important – voir plus loin)
- Un haut-parleur pour la sortie audio
- Un bouton de mise en sourdine du matériel
- De nombreuses possibilités d'extension avec les nouveaux ensembles et composants ESP32-S3-BOX-3

Le tout prêt à l'emploi dans un boîtier esthétique et acoustiquement optimisé. Théoriquement, avec l'ESP BOX et 50€, je pourrais avoir un appareil que je pourrais sortir de la boîte, flasher et placer dans ma cuisine, ma chambre, mon bureau, etc.

Logiciel

En plus de fabriquer ce matériel presque parfait, Espressif est depuis longtemps un champion de l'open-source. J'ai été ravi de voir que l'ESP BOX est prise en charge par une sélection de bibliothèques libres et gratuites :

- ESP IDF comme environnement de développement
- ESP ADF pour les applications audio
- ESP SR pour la reconnaissance de la parole
- ESP DSP pour des routines de traitement du signal hautement optimisées (FFT, calcul vectoriel, etc.)
- Un composant LVGL pour piloter des écrans LCD (tactiles compris)

Avec l'ESP BOX et ces bibliothèques, j'ai développé en l'espace d'une semaine une preuve de concept rudimentaire, capable de capturer la parole, de l'envoyer à mon implémentation de reconnaissance puis de fournir la transcription à une plateforme pour qu'elle réagisse.

Audio

Comme je l'avais appris lors de mes précédentes tentatives infructueuses, une interface utilisateur vocale commence par un mot de réveil. Vous devez pouvoir vous adresser à cette interface en prononçant un mot spécifique, pour qu'elle se réveille et commence à capturer la parole. Le mot de réveil est l'équivalent d'un bouton d'alimentation – il ne doit pas s'allumer au hasard, et lorsque vous appuyez sur le bouton, il doit fonctionner à chaque fois. Pour les applications d'interface vocale, si vous devez vous répéter plusieurs fois pour que l'appareil s'active, vous vous retrouvez rapidement dans un scénario où il est plus rapide, plus facile et beaucoup moins frustrant de sortir votre téléphone de votre poche et de faire ce que vous essayez d'y faire.
Heureusement, le canevas ESP Speech Recognition fournit non seulement un moteur de mots de réveil, mais aussi plusieurs mots de réveil. J'ai trouvé que l'implémentation du réveil était extrêmement fiable – réveillant systématiquement tout en minimisant les fausses activations. Grâce à ESP SR, je disposais d'un « bouton d'alimentation » vocal fiable.
Le défi suivant consiste à obtenir un son propre. Une fois de plus, l'ESP SR vient à la rescousse. L'ESP SR inclut l'AFE (audio front end) d'Espressif. On pourrait beaucoup dire sur l'AFE, mais en résumé, il fournit une couche de traitement audio performante qui réalise, entre l'entrée du microphone et la capture audio:

L'annulation de l'écho acoustique (AEC). Les propriétés acoustiques de l'environnement physique constituent l'un des nombreux défis de la reconnaissance de la parole en champ lointain. La distance, les surfaces réfléchissantes dures, les chemins audios alambiqués (coins, objets sur le parcours), etc. peuvent produire beaucoup d'écho. La mise en œuvre de l'AEC dans l'ESP SR élimine une grande partie de cet écho, ainsi que l'écho dans les scénarios où il y a de l'audio bidirectionnel (comme dans une application de téléphone à haut-parleur).

La séparation aveugle des sources (BSS). Dans les environnements bruyants, il est important d'éliminer les bruits non vocaux qui peuvent contribuer à une mauvaise qualité de la reconnaissance de la parole. L'implémentation ESP SR BSS, qui utilise plusieurs microphones, peut en particulier « focaliser » la capture audio sur la direction du son entrant afin de réduire de manière significative le bruit de fond capturé.

La suppression du bruit (NS). Dans les cas où il n'y a qu'un seul microphone (ou un seul actif), la suppression du bruit peut réduire fortement la capture de sons non humains. Pour les applications où du matériel personnalisé est utilisé, des microphones uniques peuvent réduire significativement les coûts de la nomenclature et la complexité de la conception.

La détection de l'activité vocale (VAD). Un mot de réveil fiable n'est qu'une pièce du puzzle. Lorsque nous nous réveillons et commençons à capturer de l'audio, nous devons arrêter la capture lorsque la personne a fini de parler. La VAD est capable de détecter le début et la fin de la parole, de sorte que l'utilisateur n'a pas besoin de mettre fin manuellement à l'enregistrement.

Willow Inference Server

Maintenant que nous pouvons réveiller, capturer de la parole propre et nous arrêter à la fin de celle-ci, nous devons l'envoyer quelque part. Heureusement, Espressif fournit son environnement de développement audio (ADF) pour cette tâche. Dans ma démonstration rapide de faisabilité, j'ai utilisé l'exemple de pipeline de flux HTTP de l'ESP ADF pour fournir des séquences audio capturées post-AFE via HTTP POST à un point de terminaison HTTP. J'ai pu rapidement adapter l'implémentation de mon serveur d'inférence de reconnaissance de la parole pour mettre en mémoire tampon les trames audios entrantes de l'ESP-BOX, attendre un marqueur de fin (grâce à VAD) et transmettre immédiatement cette mémoire tampon au modèle de reconnaissance de la parole sous-jacent. Lorsque le serveur d'inférence de reconnaissance revient avec la transcription de la parole, celle-ci est fournie en tant que réponse HTTP JSON à l'ESP-BOX pour être exécutée en suivant. Cette implémentation du serveur d'inférence de reconnaissance vocale est connue sous le nom de *Willow Inference Server (WIS)*.

L'ESP-BOX avec Willow est capable de prendre la transcription de la parole et de l'envoyer à Home Assistant ,configuré par l'utilisateur, à OpenHAB ou à un point de terminaison HTTP REST personnalisé. Willow affichera sur l'écran la transcription de la reconnaissance de la parole et la réponse du dispositif de commande. En fonction de la configuration de l'utilisateur, il émettra également une tonalité de réussite ou d'échec ou utilisera la synthèse vocale de Willow Inference Server pour énoncer le texte de sortie résultant de la commande vocale.

Aujourd'hui, avec l'ESP-BOX, Willow et Willow Inference Server, je suis capable de prononcer un mot de réveil, de saisir une commande et de l'envoyer à Home Assistant – avec une latence entre la fin de la parole et la réalisation de l'action bien inférieure à 500 ms (**figure 4)**, en fonction du matériel et de la configuration du WIS.

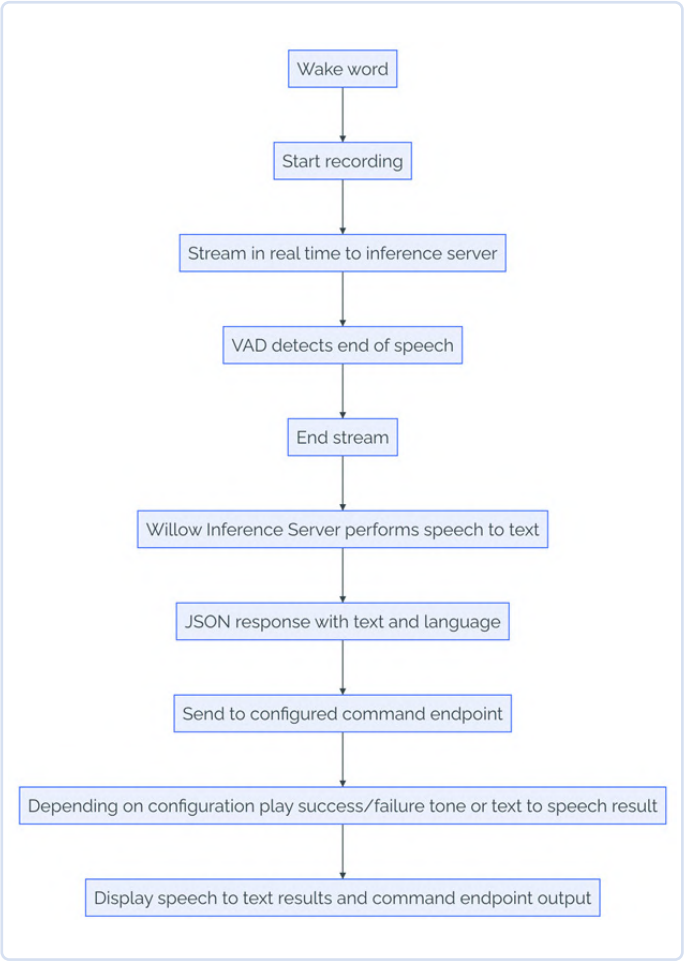


Figure 4. Flux du serveur d'inférence Willow..

Multinet : Pas de serveurs ou de matériel supplémentaires

Mais la magie de l'ESP SR ne s'arrête pas là. En plus des modèles de mots de réveil fournis, ESP SR inclut un modèle de reconnaissance de la parole sur l'appareil appelé *MultiNet* pour une reconnaissance des commandes vocales entièrement sur l'appareil. L'ESP SR permet de reconnaître jusqu'à 400 commandes vocales prédéfinies entièrement sur l'appareil (sans Willow Inference Server). Willow prend en charge MultiNet et, lorsqu'il est utilisé avec Home Assistant, il peut même extraire les noms des entités configurées pour générer automatiquement cette grammaire (**figure 5)** - sans aucun composant supplémentaire et avec des performances et une précision comparables à celles de Willow Inference Server pour ces commandes prédéfinies.

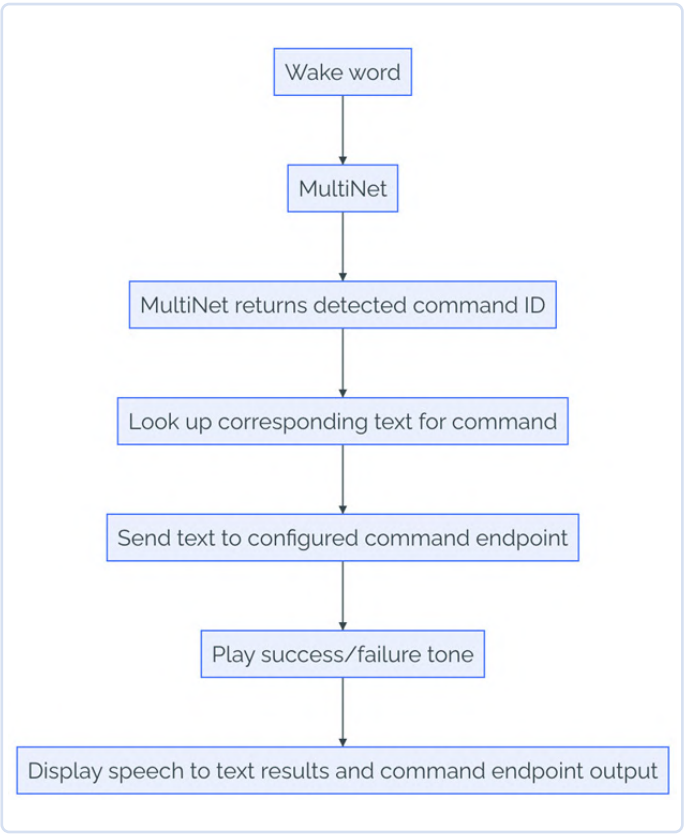


Figure 5. Flux du mode MultiNet.

Toujours à l'écoute des créateurs et des passionnés

L'ESP-BOX avec Willow peut remplacer les appareils Alexa en quelques minutes, tout en restant fidèle à ses racines pour maker d'Espressif.

L'ESP-BOX supporte une large gamme [2] de composants avec divers capteurs, GPIO, interface hôte USB et plus encore via l'interface d'extension. Les interfaces série et JTAG sont bien sûr disponibles via le port USB C de l'unité principale.

Willow est écrit en C avec ESP-IDF mais l'ESP-BOX prend en charge également des plateformes comme Arduino, PlatformIO, et CircuitPython.

Avec Willow et l'ESP-BOX, nous avons maintenant ce qui a longtemps été le « Sacré Graal » des interfaces vocales open-source : une expérience comparable à Alexa à un prix similaire avec plus de flexibilité, de contrôle et une confidentialité totale. Le tout avec des logiciels open-source et des interfaces matérielles de maker que nous apprécions tous !

VF : Denis Lafourcade — 230564-04

Questions ou commentaires ?

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

À propos de l'auteur

Kristian Kielhofner est le fondateur de Willow, un projet open-source visant à créer un assistant vocal local et autonome apte à rivaliser avec Amazon Echo/Google Home. Depuis son enfance passée à jouer avec l'Apple IIe, Kristian a toujours été un passionné de technologie et a fondé de nombreux projets open-source et startups dans les domaines de la voix et de l'apprentissage automatique. Kristian consacre aujourd'hui son temps à Willow et à d'autres projets open-source, tout en conseillant des entreprises technologiques en phase de démarrage.

Produit

- **ESP32-S3-BOX-3**
  www.elektor.fr/20627

LIENS

[1] Plate-forme Willow : https://heywillow.io
[2] ESP32-S3-BOX-3 : https://www.espressif.com/en/news/ESP32-S3-BOX-3

SDK

ESPRESSIF

ESP-IoT-Solution

pilotes de périphériques, cadres de développement et plus encore pour vos systèmes IdO !

ESP-IoT-Solution est un dépôt où vous trouverez de nombreuses implémentations de capteurs, d'écrans, d'appareils audio, d'entrées et de pilotes d'actionneurs pour les SoC Espressif.
En plus, il offre certains des cadres de haut niveau qui sont généralement requis, notamment un pilote de bouton-poussoir capable de détecter les pressions longues et courtes. Il comprend également des exemples d'applications spécifiques pour les modes basse consommation, le stockage et la sécurité.
Si vous souhaitez effectuer des mises à jour OTA directes d'un appareil BLE depuis un téléphone, c'est ici qu'il faut se rendre. Beaucoup de ces fonctions sont disponibles sous forme de composants IDF individuels que vous pouvez importer directement dans votre projet.

https://github.com/espressif/esp-iot-solution

l'œil qui réfléchit

reconnaissance faciale et plus encore, utilisant l'ESP32-S3-EYE

Tam Hanna (Hongrie)

La carte ESP32-S3-EYE d'Espressif est une plateforme spécialement conçue pour évaluer les applications de reconnaissance faciale et autres applications d'Intelligence Artificielle. Elle est fournie avec un écosystème de logiciels et de nombreux exemples, provenant de l'ensemble d'outils ESP-WHO du fabricant en ce qui concerne la reconnaissance faciale, et de TensorFlow pour les opérations manuelles. Allons les découvrir !

Les progrès et la sophistication grandissante des algorithmes d'Intelligence Artificielle (IA) ont été fulgurants ces dernières années. Il existe maintenant des systèmes très performants qui fournissent des résultats impressionnants, comparables à ceux d'un être humain (en particulier dans le cas de l'IA hébergée dans le nuage). La loi de Moore, et l'accroissement constant de la demande des utilisateurs, imposent aux fabricants d'inclure des composants optimisés pour l'IA dans leur offre de dispositifs. En ce qui concerne Espressif, il s'agit de l'ESP32 dans sa variante S3 qui, grâce à ses instructions vectorisées, offre toutes sortes de moteurs d'accélération d'Intelligence Artificielle et un ensemble spécifique d'instructions dédiées à l'IA.
Ces derniers mois, un écosystème robuste et évolutif s'est développé autour de l'ESP32-S3, facilitant la création de diverses applications d'IA par les développeurs. Dans cet article, nous allons détailler et expérimenter les possibilités de cette technologie.

Débuts sans appréhension

Espressif est conscient des besoins des développeurs. Depuis le lancement de la carte ESP32-LyraT, prévue pour les applications audio-phoniques, il y a quelques années, le fabricant à constamment introduit des cartes d'évaluation pour des «besoins spécifiques». Nous allons utiliser l'ESP32-S3-Eye pour exécuter l'ensemble des exemples décrits dans les paragraphes suivants. La **figure 1** est un schéma des blocs fonctionnels des composants de la carte à circuit-imprimé. La présence

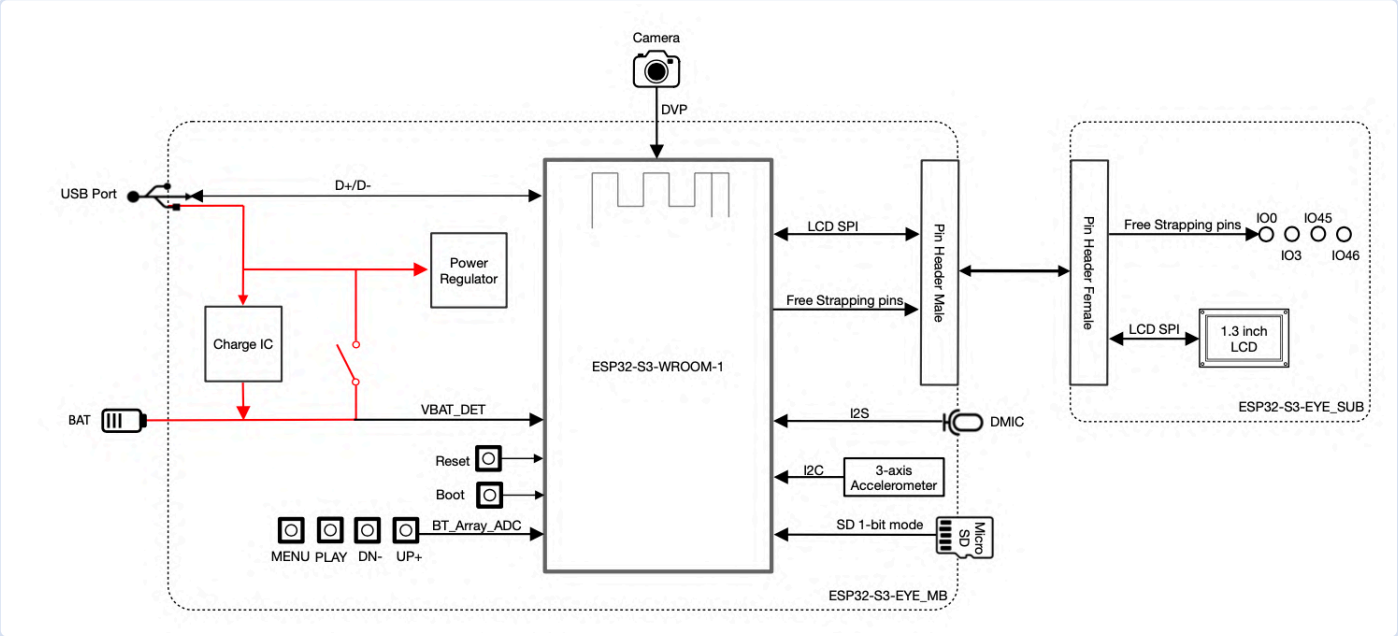


Figure 1. Blocs fonctionnels de l'ESP32-S3-EYE (Source : [10]).

sur la carte d'une caméra et d'un petit écran LCD [1] m'a convaincu d'utiliser cette carte d'évaluation dont le prix est d'environ 42 €. Cet écran couleurs se révèle être très utile, permettant de voir directement les résultats du processus d'apprentissage de ce dispositif, sans nécessiter l'utilisation d'un PC. Un microphone numérique est situé à la surface du circuit-imprimé, sous l'écran. Il peut être utilisé pour tester divers logiciels de reconnaissance du langage. Il est important de noter que les expérimentations entreprises peuvent également être faites en utilisant d'autres plateformes matérielles. Le schéma de l'ESP32-S3-Eye se trouve en [2], il peut servir de base pour la réalisation de circuits personnalisés. Les composants utilisés par Espressif sont en général disponibles chez plusieurs distributeurs.

Disponibilité des composants ES8388

Se procurer les composants recommandés par Espressif nécessite parfois une approche différente. En ce qui concerne les codecs audio ES8388, je n'ai pas réussi à trouver un distributeur, basé aux USA ou en Europe, qui ne l'ait en stock. Une demande directe à son fabricant m'a toutefois permis d'être orienté vers Newtech Component Ltd, un distributeur qui a été suffisamment coopératif pour m'en procurer gratuitement 30 exemplaires. Pour cela, il convient de prévoir un transporteur pouvant expédier les échantillons à une adresse locale. Pour cela, l'auteur utilise TipTrans.

La carte est fournie avec le micrologiciel installé ce qui évite tout problème durant la procédure de mise en œuvre du système. Si la carte ESP32-S3-EYE est à l'état usine, vous n'avez qu'à la mettre sous tension par le port micro-USB et attendre quelques secondes. L'application de reconnaissance faciale apparaîtra ensuite, comme illustré sur la **figure 2**.
Il est important de remarquer qu'à cette étape, les algorithmes d'IA et d'apprentissage (ML ou Machine Learning), font preuve d'une excellente résilience relativement à la qualité des données capturées. La photographie de la figure 2 a été prise en conservant le film de protection de l'objectif de la caméra. La réduction de contraste qui en résulte (et malgré que je ne sois pas rasé), n'a pas pris en défaut le processus d'IA de reconnaissance faciale.

Retour vers le futur

Parfois, vous souhaiterez peut-être réinitialiser votre ESP32-S3-EYE à «l'état usine». Les fichiers .bin précompilés se trouvent à l'adresse https://github.com/espressif/esp-who/tree/master/default_bin. Ils peuvent être "flashés" sur la carte de façon identique à tout autre fichier binaire.

Première expérimentation

La présence, quasiment illimitée, de capital-risque, a conduit à la création d'une multitude de plateformes d'IA. La logique impose que ces sociétés ne considèrent pas uniquement les PC, mais incluent également des microcontrôleurs dans leurs listes de plateformes supportées. Une revue détaillée de l'état actuel du marché se situerait bien au-delà

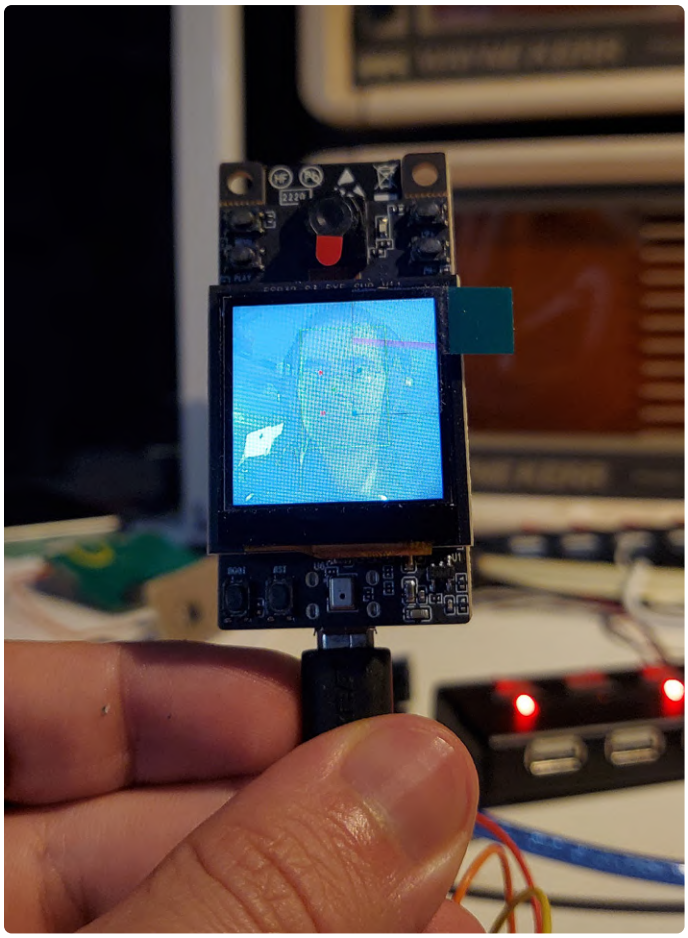


Figure 2. Bien que mon laboratoire soit sombre, la caméra fonctionne malgré la présence du film de protection de l'objectif !

des objectifs de cet article, c'est pourquoi, pour cette première étape, nous nous limiterons à expérimenter le processus de développement du traitement d'images sur la plateforme *ESP-WHO* proposée par Espressif. Il s'agit d'une plateforme «optimisée» pour certains types de reconnaissance d'images animées. Son diagramme structurel est illustré **figure 3**.

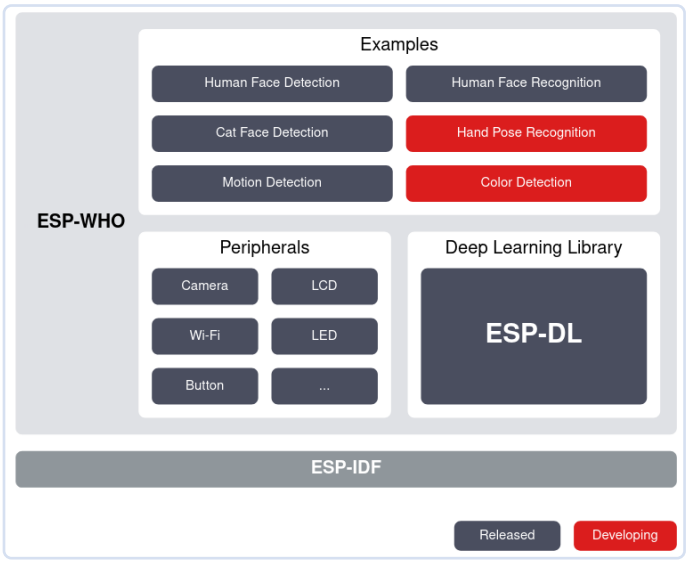


Figure 3. L'ESP-WHO utilise quelques autres composantes de l'écosystème Espressif (Source : [11]).

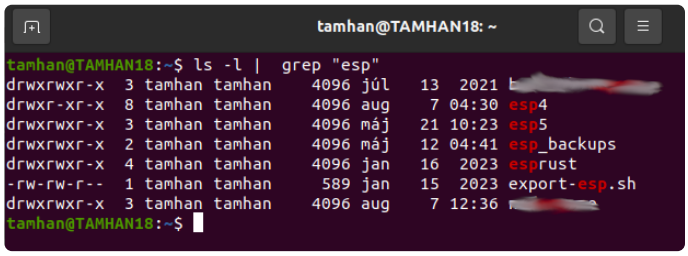


Figure 4. Différentes versions de l'ESP-IDF peuvent en général coexister sur la même station de travail.

La bibliothèque *ESP Deep Learning Library* (ESP-DL), disponible en [3], est une ressource importante. Il s'agit essentiellement d'un ensemble optimisé offrant diverses techniques d'IA et réduisant la latence lorsqu'on l'utilise conjointement avec un accélérateur matériel. Il convient de noter que, tels que présents dans l'ensemble, les composants WHO fonctionnent sous la version 4.4 de l'*ESP-IDF* et ne permettent pas encore un fonctionnement sous la version 5.0. Pour les projets de mes activités de consulting, j'utilise généralement la version 4.4, mais je dispose également de plusieurs variantes d'environnements de développement ESP32, installés sur mon ordinateur comme le montre le **figure 4**.
La version IDF utilisée dans les étapes suivantes, est identifiée ainsi :

```
tamhan@TAMHAN18:~/esp4/esp-idf$ idf.py --version
ESP-IDF v4.4.4-dirty
```

Nous sommes maintenant prêts pour le téléchargement du code *ESP-WHO.* La commande `git` suivante le permet :

```
tamhan@TAMHAN18:~/esp4$ git clone --recursive https://
github.com/espressif/esp-who.git
. . .
```

L'exécution de la commande de compression d'objets `Compressing objects:` peut nécessiter un temps d'attente important, les dépôts sont parfois assez volumineux et lents à manipuler. Après avoir terminé cette étape, il est recommandé de procéder à l'initialisation du sous-ensemble, selon le processus suivant :

```
tamhan@TAMHAN18:~/esp4$ cd esp-who/
tamhan@TAMHAN18:~/esp4/esp-who$ git submodule update
--recursive --init
```

Dans la plupart des cas, la commande `git submodule update --recursive --init` ne produira aucun affichage à l'écran. Cela signifie que l'image en cours de traitement est « complète ».

Quelques exemples

Le logiciel de reconnaissance faciale *ESP-WHO* fourni par Espressif comporte trois options d'interface. Des exemples de projets sont présents sur la **figure 6**.

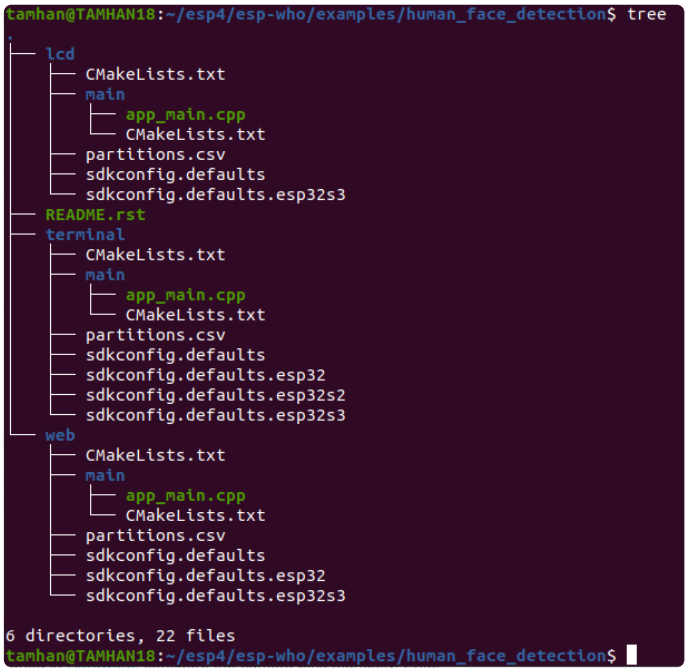


Figure 5. Variantes ESP-IDF dont je dispose !

Dans les étapes qui suivent, je vais utiliser la variante lcd des exemples `~/esp4/esp-who/examples/human_face_detection`. Les résultats obtenus seront donc directement affichés sur le petit écran présent sur la carte ESP32. La variante *Terminal* utilise le moniteur de communication idf-py, alors que la variante *Web* met en jeu un serveur web. Pour commencer, vous devez spécifier le type de contrôleur ESP32 que vous souhaitez utiliser selon la procédure suivante. Si, comme moi, vous utilisez une carte ESP32-S3-EYE, la chaine `esp32s3` est transmise, par la commande :

```
tamhan@TAMHAN18:~/esp4/esp-who/examples/human_face_detec-
tion/lcd$ idf.py set-target esp32s3
```

Le code source actuel de l'exemple est d'une simplicité impressionnante, il se trouve dans le fichier *main/app_main.cpp*. Pour en avoir une vue globale meilleure, je vais tout d'abord l'imprimer complètement, avant d'y ajouter des commentaires (voir le **listage 1)**.
Le cœur du moteur ESP-WHO utilise l'objet du noyau *queue* présent dans FreeRTOS, décrit en détail en [4]. Les deux déclarations statiques créent une file d'attente (queue) d'entrée et sortie qui sera ensuite utilisée pour manipule les données via le pipeline de reconnaissance (recognition pipeline).
Le reste du code, contribution du développeur, se consacre principalement à la mise en œuvre d'un pipeline par l'appel à trois fonctions. Les analogies au pipeline utilisé dans l'ESP-ADF la plateforme audio [5] sont évidentes.
Pour continuer la mise en œuvre, vous devrez entrer la commande `idf.py menuconfig` dans la première étape, afin de charger l'environnement correspondant à la configuration habituelle *menuconfig,*

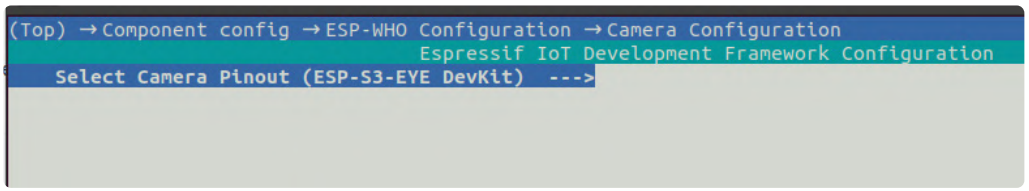


Figure 6. Si vous utilisez un module caméra existant, vous devez le reconfigurer ici !

Listage 1. Code source de l'exemple.

```
#include "who_camera.h"
#include "who_human_face_detection.hpp"
#include "who_lcd.h"

static QueueHandle_t xQueueAIFrame = NULL;
static QueueHandle_t xQueueLCDFrame = NULL;

extern "C" void app_main()
{
    xQueueAIFrame = xQueueCreate(2, sizeof(camera_fb_t *));
    xQueueLCDFrame = xQueueCreate(2, sizeof(camera_fb_t *));

    register_camera(PIXFORMAT_RGB565, FRAMESIZE_240X240, 2, xQueueAIFrame);
    register_human_face_detection(xQueueAIFrame, NULL, NULL, xQueueLCDFrame, false);
    register_lcd(xQueueLCDFrame, NULL, true);
}
```

utilisée dans les autres projets ESP32 (ainsi que dans le noyau Linux). Rendez-vous ensuite, dans la section *Component config → ESP-WHO Configuration*. Dans le champ *Camera Configuration,* assurez-vous que la caméra de votre carte d'évaluation est préconfigurée comme le montre la **figure 6.**

Enregistrez la configuration créée dans menuconfig, puis exécutez la commande habituelle `idf.py build` pour lancer la compilation. La création de l'image exécutable prendra un peu plus de temps, car approximativement 1200 fichiers doivent être compilés par la plate-forme de développement.

Avant de charger le logiciel dans la carte par la commande `idf.py flash` suivie du numéro de port, vous devrez mettre la carte en mode bootloader. Utilisez pour cela les deux boutons situés à proximité du connecteur USB. Maintenez appuyé le bouton BOOT tout en appuyant momentanément sur le bouton RST, relâchez ensuite les deux boutons. Vous pouvez maintenant continuer par le chargement du programme dans l'ESP32, selon la procédure habituelle. Vous pourrez vérifier la réussite du chargement du mode bootloader dans *dmesg*, comme l'indique la **figure 7**. Appuyez à nouveau sur le bouton RST afin de lancer l'exécution de la nouvelle variante de la reconnaissance faciale familiale (figure 2).

Examen succinct du code

L'utilisation de l'ESP-WHO est une introduction relativement simple destinée aux développeurs qui commencent à expérimenter des applications d'Intelligence Artificielle, sans devoir fournir un effort important. Espressif fournit également le code source de certains composants [6], permettant d'avoir une connaissance des routines de reconnaissance faciale.

Il utilise le détecteur préexistant `HumanFaceDetectMSR01`, qui reçoit certains paramètres transmis lors de la procédure de paramétrisation :

```
static void task_process_handler(void *arg) {
  camera_fb_t *frame = NULL;
  HumanFaceDetectMSR01 detector
      (0.3F, 0.3F, 10, 0.3F);
```

Le travail réel est effectué par le modèle ML (apprentissage machine) dans une boucle infinie qui reçoit les images individuelles à traiter de la queue créée préalablement :

```
while (true) {
   bool is_detected = false;
   if (xQueueReceive(xQueueFrameI,
     &frame, portMAX_DELAY)) {
     std::list<dl::detect::result_t> &detect_results =
       detector.infer((uint16_t *)frame->buf,
      {(int)frame->height, (int)frame->width, 3});
```

Les appels à la fonction `detector.infer()` permettent de s'assurer que le process d'inférence est exécuté. Si l'objet retourné `results` comporte des résultats de détection, le programme fait appel à deux fonctions de sortie :

```
    if (detect_results.size() > 0) {
     draw_detection_result((uint16_t *)frame->buf,
     frame->height, frame->width, detect_results);
     print_detection_result(detect_results);
     is_detected = true;
    }
   }
```

```
[ 8792.171038] usb 1-1.5: new full-speed USB device number 6 using ehci-pci
[ 8792.301169] usb 1-1.5: New USB device found, idVendor=303a, idProduct=1001, bcdDevice= 1.01
[ 8792.301175] usb 1-1.5: New USB device strings: Mfr=1, Product=2, SerialNumber=3
[ 8792.301178] usb 1-1.5: Product: USB JTAG/serial debug unit
[ 8792.301181] usb 1-1.5: Manufacturer: Espressif
[ 8792.301183] usb 1-1.5: SerialNumber: 34:85:18:8C:50:D8
[ 8792.301642] cdc_acm 1-1.5:1.0: ttyACM0: USB ACM device
tamhan@TAMHAN18:~$
```

Figure 7. Ce message d'état indique une connexion réussie.

Performance Comparison

A quick summary of ESP-NN optimisations, measured on various chipsets:

Target	TFLite Micro Example	without ESP-NN	with ESP-NN	CPU Freq
ESP32-S3	Person Detection	2300ms	54ms	240MHz
ESP32	Person Detection	4084ms	380ms	240MHz
ESP32-C3	Person Detection	3355ms	426ms	160MHz

Figure 8. L'activation de l'accélérateur d'IA permet d'obtenir un résultat plus rapidement. (Source : [9]).

Le reste de l'ESP-WHO est généralement constitué de composants logiciels préexistants, comme par exemple ESP-Cam [7], permettant l'accès à la caméra. Je n'irai pas plus loin dans le logiciel ESP-WHO car les exemples fournis sont explicites.

Approfondissons en utilisant TensorFlow

Ceux qui souhaitent un contrôle plus approfondi du comportement de leur système seront tentés de partir de zéro pour la programmation de l'ensemble, cela n'est toutefois pas une solution pratique. Sauf si une grande attention y est apportée, le système pourrait rapidement devenir incontrôlable et conduire à des coûts élevés de maintenance du logiciel au cours de son existence.

La bibliothèque Google *TensorFlow*, disponible à [8], est devenue quasi standard et existe, depuis pas mal de temps, dans des versions optimisées, destinées à divers microcontrôleurs. Il est important de remarquer que dans GitHub, elle se trouve dans deux dépôts ; la raison en est surprenante, car Google est passé de la distribution de code générique et spécifique à un fabricant, à la bibliothèque *TensorFlow,* en cours du développement de cette plateforme. La version de cette bibliothèque se trouve en [9].

Étant donné que TensorFlow accède à diverses composantes en arrière-plan de la plateforme ESP-IDF, le déploiement à partir GitHub devra se faire en incluant le paramètre `--recursive` qui délègue à la ligne de commande la nécessité d'indiquer l'endroit où se trouve le code et de s'assurer que l'on dispose du dépôt principal et l'ensemble des sous-modules.

```
tamhan@TAMHAN18:~/esp4$ git clone --recursive https://
github.com/espressif/tflite-micro-esp-examples.git
```

Pour un test initial de mise en œuvre, l'exemple *~/esp4/tflite-micro-esp-examples/examples/hello_world* peut être utilisé. Il utilise un modèle d'apprentissage (ML) optimisé par la prédiction des valeurs d'une fonction sinusoïdale; alternative à l'algorithme conventionnel CORDIC, et d'un modèle qui fonctionne avec moins de données en entrée. Dans l'étape suivante, entrez à nouveau la commande `idf.py set-target esp32s3` afin d'optimiser le squelette de l'application pour l'ESP-32-S3.

Selon la version de l'ESP-IDF disponible sur votre station de travail ou votre PC, vous pourrez avoir des messages d'erreur lors du paramétrage du code téléchargé depuis le dépôt (*Invalid manifest...)*. Ces erreurs proviennent de composantes partiellement obsolètes dans la chaîne de compilation. Pour résoudre ce problème, lancez l'exécution du programme suivant à partir de la ligne de commande :

```
/home/tamhan/.espressif/python_env/idf4.4_py3.8_env/bin/
python -m pip install --upgrade idf-component-manager
```

Ouvrez ensuite une fenêtre *Nautilus* pointant sur le répertoire contenant le squelette du projet, en entrant la commande suivante. La suppression du répertoire de préparation de l'exécutable doit être faite manuellement de telle façon que la commande `set-target` puisse à nouveau créer les fichiers de travail de la compilation :

```
tamhan@TAMHAN18:~/esp4/tflite-micro-esp-examples/examples/
hello_world$ nautilus .
tamhan@TAMHAN18:~/esp4/tflite-micro-esp-examples/examples/
hello_world$ idf.py set-target esp32s3
tamhan@TAMHAN18:~/esp4/tflite-micro-esp-examples/examples/
hello_world$ idf.py menuconfig
```

Notez l'entrée *ESP-NN* dans *Menuconfig.* Elle permet le fonctionnement de la configuration de la bibliothèque de chargement dynamique (DL ou Dynamic Loader). La sélection de l'option *Optimized versions* dans la section *Optimization for neural network functions* est fortement recommandée car elle active l'accélérateur. Les performances indiquées sur la **figure 8** montrent l'accroissement signifiant que l'on obtient quand cette option est validée.

```
I (292) cpu_start: Starting scheduler on PRO CPU.
I (0) cpu_start: Starting scheduler on APP CPU.
x_value: 0.000000, y_value: 0.000000

x_value: 0.314159, y_value: 0.372770

x_value: 0.628319, y_value: 0.559154

x_value: 0.942478, y_value: 0.838731

x_value: 1.256637, y_value: 0.965812

x_value: 1.570796, y_value: 1.042060

x_value: 1.884956, y_value: 0.957340

x_value: 2.199115, y_value: 0.821787

x_value: 2.513274, y_value: 0.533738

x_value: 2.827433, y_value: 0.237217

x_value: 3.141593, y_value: 0.008472

x_value: 3.455752, y_value: -0.304993

x_value: 3.769912, y_value: -0.533738

x_value: 4.084070, y_value: -0.779427

x_value: 4.398230, y_value: -0.965812

x_value: 4.712389, y_value: -1.109837
```

Figure 9. Un réseau neuronal fonctionne également avec des données sinusoïdales.

La compilation et l'exécution vont maintenant se dérouler comme prévu. Vous pouvez voir la sortie des résultats obtenus sur la **figure 9**.

Analyse du code TensorFlow

Ne soyez pas surpris si des images chaotiques apparaissent sur l'écran de l'ESP-EYE, l'écran LCD intégré a son propre contrôleur d'image, il ne visualise que la dernière image enregistrée jusqu'à ce qu'une nouvelle mise à jour intervienne.
Si vous visualisez le fichier main.cc dans votre éditeur de texte de votre choix, vous trouverez le fragment de code suivant :

```
#include «main_functions.h»

extern «C» void app_main(void) {
  setup();
  while (true) {
   loop();
  }
}
```

Vos premières impressions ne vous trompent pas, TensorFlow est proche de l'environnement Arduino. La définition des fonctions `setup()` et `loop()` se trouve dans le fichier *main_functions.cc*.
La constitution de la fonction `loop()` est particulièrement intéressante, car elle est responsable de l'exécution des échanges. Sa première tâche consiste à générer les données d'entrée qui seront transmises au prédicteur sinusoïdal :

```
void loop() {
  float position =
   static_cast<float>(inference_count) /
   static_cast<float>(kInferencesPerCycle);
  float x = position * kXrange;
```

À l'étape suivante, l'information est quantifiée afin de la rendre plus « digeste » par le réseau neuronal. Il s'agit d'une approche habituelle en matière d'apprentissage machine (Machine Learning). La normalisation de toutes les données d'entrée entre 0 et 1 permet de contrôler la complexité des résultats algorithmiques :

```
  int8_t x_quantized =
    x / input->params.scale +
     input->params.zero_point;
  input->data.int8[0] = x_quantized;
```

Les calculs et la quantification interviennent dans le bloc suivant :

```
  TfLiteStatus invoke_status =
      interpreter->Invoke();
  if (invoke_status != kTfLiteOk) {
   MicroPrintf(«Invoke failed on x: %f\n»,
    static_cast<double>(x));
   return;
  }
  int8_t y_quantized = output->data.int8[0];
  float y = (y_quantized -
     output->params.zero_point) *
     output->params.scale;
```

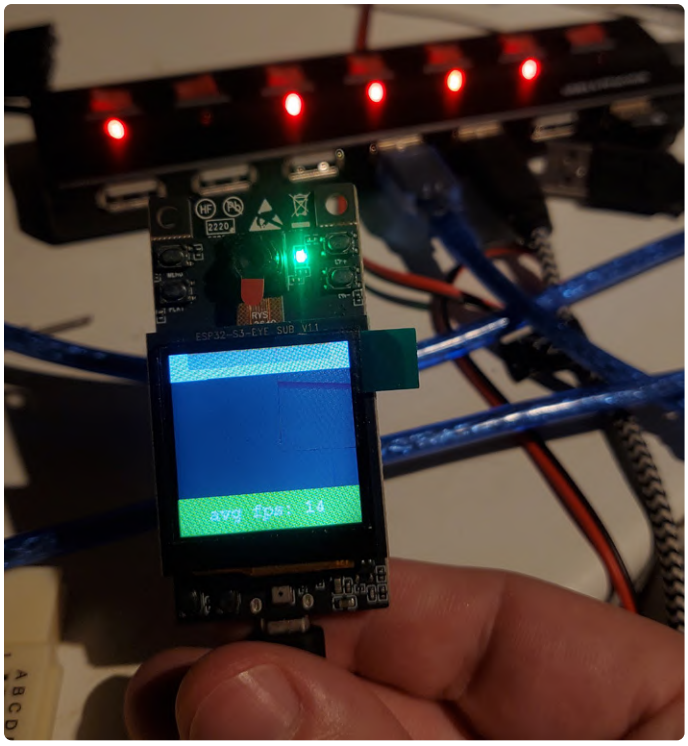


Figure 10. Le bandeau vert en bas d'écran indique une reconnaissance réussie

Pour terminer, un nettoyage final est nécessaire. Il est intéressant de noter que selon la documentation de TensorFlow, la fonction `HandleOutput()` en charge de la sortie des données, doit être fournie par le développeur :

```
  HandleOutput(x, y);
  inference_count += 1;
  if (inference_count >= kInferencesPerCycle)
      inference_count = 0;
}
}
```

Expérimentation des modules plus évolués

Si vous analysez en détail les exemples contenus dans le répertoire *~/esp4/tflite-micro-esp-examples/examples,* fourni par Google, vous constaterez qu'un exemple de reconnaissance de la parole nommé *micro_speech* y est présent, ainsi qu'un exemple de reconnaissance faciale appelé *person_detection.* Pour obtenir ces exemples et en lancer l'exécution, il est nécessaire de suivre les trois étapes habituelles des procédures de paramétrage, vérifier les paramètres de *menuconfig,* suivie du téléversement du code machine de l'ESP32.
Il est important de noter que ces exemples avancés fonctionnent en utilisant les échanges par la ligne de commande. Si vous souhaitez ajouter la possibilité de sorties sur l'écran des données du détecteur d'identité, vous devrez adapter le fichier *esp_main.h* comme il suit :

```
// Enable this to do inference on embedded images
//#define CLI_ONLY_INFERENCE 1
#define DISPLAY_SUPPORT 1
```

Après recompilation, le contenu de l'écran représenté sur la **figure 10** apparaîtra. L'étude des problèmes relatifs à l'affichage pourraient faire l'objet d'un autre article. Une reconnaissance réussie est indiquée lorsque le bas de l'écran devient vert.

Deux méthodologies

Les expérimentations menées ici démontrent que la plateforme ESP32-S3 est capable de réaliser des tâches de reconnaissance d'objets et faciale de différentes façons. Tandis que l'utilisation des possibilités offertes par l'ESP-WHO permet d'obtenir rapidement des résultats appréciables, la méthode intégrant l'écosystème TensorFlow permet d'utiliser des techniques avancées dans le domaine de l'apprentissage machine.

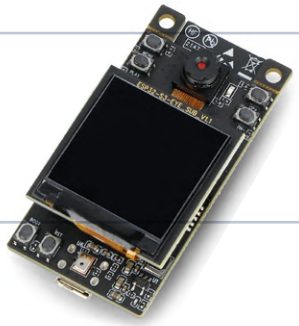
VF : Jean Boyer — 230556-04

Questions ou commentaires ?

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

Produit

› **ESP32-S3-EYE**
www.elektor.fr/20626



LIENS

[1] Recherche de distributeurs de l'ESP32-S3-EYE : https://oemsecrets.com/compare/ESP32-S3-EYE
[2] Schéma de l'ESP32-S3-EYE : https://tinyurl.com/esp32eyeschematics
[3] Bibliothèque d'apprentissage profond pour l'ESP : https://github.com/espressif/esp-dl
[4] FreeRTOS Queues : https://freertos.org/a00018.html
[5] T. Hanna, « les signaux audios et l'ESP32 », Elektor 1-2/2023 : https://www.elektormagazine.fr/magazine/elektor-291/61422
[6] Exemples d'IA avec l'ESP-WHO : https://github.com/espressif/esp-who/tree/master/components/modules/ai
[7] Contrôle d'une caméra avec l'ESP32 : https://github.com/espressif/esp32-camera
[8] TensorFlow : https://tensorflow.org
[9] TensorFlow Lite Micro : https://github.com/espressif/tflite-micro-esp-examples
[10] Schéma des blocs fonctionnels de l'ESP32-S3-EYE : https://github.com/espressif/esp-who/blob/master/docs/en/get-started/ESP32-S3-EYE_Getting_Started_Guide.md
[11] ESP-WHO sur GitHub : https://github.com/espressif/esp-who

SDK

Arduino-ESP32

ESPRESSIF

tout le support dont vous avez besoin pour ESP32, ESP32-S2, ESP32-S3 et ESP32-C3

Le support Arduino pour les puces ESP32, ESP32-S2, ESP32-S3, et ESP32-C3 est disponible dans ce dépôt. Vous pouvez utiliser l'EDI standard Arduino ou l'EDI PlatformIO pour développer des applications Arduino sur ces circuits intégrés.
Ce dépôt contient de nombreuses bibliothèques utiles, y compris les pilotes de périphériques couramment utilisés, la prise en charge des protocoles Wifi, BLE et ESP-NOW, et des fonctions de haut niveau telles que ESP-Insights et ESP-RainMaker.
Toutes ces bibliothèques contiennent des exemples qui permettent de démontrer les fonctionnalités. De nombreuses cartes de développement basées sur les circuits intégrés mentionnés sont également prises en charge.

https://github.com/espressif/arduino-esp32

commutateur alimenté par pile bouton, basé sur un ESP32-C2

évaluation de la conception et des performances

Li Junru and Zhang Wei, Espressif

Ce commutateur à pile bouton, basé sur un ESP32-C2 est une solution domotique. Il offre une communication directe, stable et bidirectionnelle, avec les dispositifs équipés de microcontrôleurs ESP auxquels il transmet les commandes de contrôle. De taille compacte, l'autonomie de sa pile peut atteindre cinq ans. Cet article traite de sa réalisation matérielle et logicielle, soulignant ses bénéfices pour les solutions de domotique et les applications de l'Internet des Objets.

Figure 1. Le commutateur à pile bouton basé sur l'ESP32-C2 dans son boitier.

En raison du développement rapide du marché de la domotique, la demande de commutateurs à faible consommation énergétique mais de grande efficacité, se fait croissante. Cet article présente une solution de pointe, un commutateur alimenté par une pile bouton, réalisé autour d'un ESP32-C2 (**figure 1**), dont l'objectif est de répondre aux impératifs de rapidité de réaction et à la nécessité de passerelles supplémentaires, que l'on rencontre souvent avec les autres solutions sans-fil, basées sur des technologies telles que Bluetooth LE (Low Energy ou à faible consommation) et ZigBee.

Le commutateur à pile bouton basé sur l'ESP32-C2, peut se targuer d'offrir plusieurs avantages, par rapport aux autres commutateurs :

- Communication directe avec les luminaires intelligents ou les interrupteurs muraux équipés de circuits intégrés ESP, éliminant ainsi la nécessité d'une passerelle supplémentaire.
- Communication bidirectionnelle et stable, garantissant un taux de réussite des transmissions Wifi par paquets.
- Alimenté par une pile bouton, la taille du dispositif est minimisée, permettant une adaptation aux formats des autres produits tels que les interrupteurs auto-adhésifs, les interrupteurs à commandes multiples, les interrupteurs sensitifs et les commutateurs rotatifs.
- L'ESP32-C2 est maintenu non alimenté lorsqu'il est inutilisé, permettant ainsi à une seule pile bouton CR2032 d'atteindre une durée de vie de 5 années (en considérant 10 manœuvres par jour).

Dans cet article, nous étudierons en détail la réalisation du commutateur à pile bouton utilisant une puce ESP, démontrant sa capacité à répondre aux exigences de la technologie domotique moderne.

Les piles bouton, comme par exemple la pile CR2032 largement répandue, sont souvent utilisées comme source d'énergie des dispositifs de l'Internet des objets (IdO). Réputées pour leur taille réduite et leur légèreté, les piles bouton conviennent parfaitement aux dispositifs électroniques miniaturisés sans les alourdir. Leur densité énergétique importante leur permet de stocker davantage d'énergie dans un volume restreint, apportant en consé-

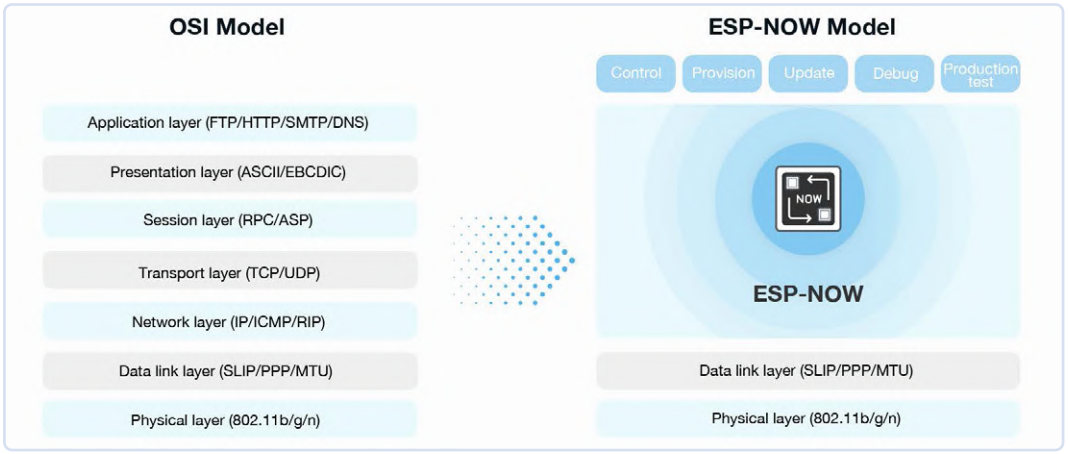


Figure 2. Le modèle ESP-NOW réduit à une couche les cinq couches supérieures du modèle OSI.

quence une plus grande autonomie d'utilisation. De plus, les piles boutons bénéficient de caractéristiques d'autodécharge faibles, leur permettant de conserver leur capacité durant de longues périodes de non-utilisation. Conservant une tension relativement stable durant leur décharge, les piles bouton ont un rôle vital, assurant un fonctionnement correct des dispositifs. Toutefois, en raison de leur conception, elles procurent une intensité de sortie moins élevée, les rendant inutilisables pour les dispositifs exigeant une puissance importante.

Dans les divers domaines de leurs applications, les appareils IdO ont vu leur utilisation augmenter, grâce à la couverture étendue des réseaux Wifi. Grâce à l'élimination des liaisons filaires traditionnelles entre les appareils, la technologie Wifi permet un déploiement convenable, plus souple, des dispositifs de l'Internet des Objets. De plus, la technologie supporte les connexions simultanées entre plusieurs dispositifs, les rendant particulièrement intéressants pour les scénarios de l'IdO impliquant de nombreux appareils interconnectés. Le marché offre une pléthore de composants et dispositifs supportant le Wifi, réduisant significativement les coûts de développement et de production des équipements de l'Internet des Objets. Il est néanmoins important de considérer que la consommation des dispositifs Wifi est relativement élevée, en comparaison avec les autres technologies sans fil à faible puissance. En conséquence, la technologie Wifi convient moins bien à certains dispositifs IdO devant être alimentés par des piles. À titre d'exemple, l'ESP32-C2 exige un courant maximum de 370 mA durant la transmission, et 65 mA durant une opération de réception.

Les caractéristiques de consommation énergétique des dispositifs Wifi imposent un double challenge dans leur utilisation dans les domaines d'application à faible puissance. D'une part, l'intensité en réception rend difficile le maintien permanent du fonctionnement de ces circuits en mode réception. D'autre part, le courant instantané important nécessaire durant la transmission par paquets pourrait impacter la stabilité de la tension d'alimentation du chip, conduisant à une possible réinitialisation accidentelle du composant. Dans cet article, nous présentons un commutateur alimenté par une pile bouton, basé sur l'utilisation de l'ESP32-C2, qui fait face à ces difficultés par la combinaison optimisée de logiciel et matériel aboutissant à une impressionnante durée de vie de la pile.

L'article est divisé en trois sections principales. En premier, nous analysons en détail le logiciel développé, apportant une vue détaillée de la stratégie de programmation et ses bénéfices. En second, nous étudions la réalisation matérielle correspondante, en particulier un guide pour la sélection des composants. Pour terminer, dans la partie expérimentale, nous évaluerons les performances énergétiques obtenues ainsi que le temps de latence du dispositif, afin de fournir au lecteur une évaluation objective et complète de cette solution.

Conception du logiciel

Choix de la couche protocole : la première décision cruciale concernait le choix du protocole approprié. De façon conventionnelle, le Wifi fonctionne en mode connecté, c'est-à-dire que l'équipement maintient une connexion permanente au réseau sauf en cas de déconnexion manuelle intentionnelle ou en cas de situation hors de portée. Cette connexion permanente permet au dispositif de rester en permanence prêt à communiquer, sans avoir besoin de se reconnecter à chaque utilisation. Cependant, le maintien de la connexion consomme une quantité d'énergie significative.

Pour résoudre cette difficulté, nous avons adopté une solution de communication plus simple. En ce qui concerne le logiciel, nous y sommes parvenus en transmettant des messages au niveau de la couche « data link » , permettant à l'équipement récepteur de retrouver facilement l'information pour la retransmettre. De plus, nous avons défini un temps de réception prédéterminé, afin de minimiser autant que possible la consommation énergétique. Pour ce faire, le protocole

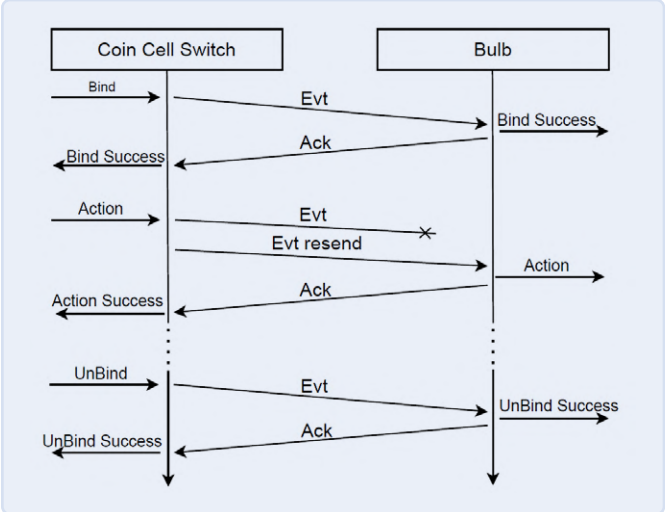


Figure 3. Mécanisme d'accusé-réception implémenté dans la couche application.

Tableau 1. Consommation d'énergie avant la mise en œuvre de l'optimisation.

Opération	Durée (ms)	Puissance Moyenne (mW)	Consommation d'énergie (mJ)
Chargement initial	364,7	58	21,16
Initialisation Wifi	115,2	69,8	8,03
Démarrage Wifi	56,6	303,9	17,2

Tableau 2. Consommation d'énergie après la mise en œuvre de l'optimisation.

Opération	Durée (ms)	Puissance Moyenne (mW)	Consommation d'énergie (mJ)
Chargement initial	37,52	53,14	2,0
Initialisation Wifi	6,55	72,62	0,48
Démarrage Wifi	19,12	164,0	3,13

ESP-NOW, développé par Espressif Systems s'est révélé être un choix idéal. ESP-NOW est un protocole de communication sans fil basé sur la couche « data link » . il réduit les cinq couches supérieures du modèle OSI en une seule couche (**figure 2**).
L'identification du dispositif est déterminée par son adresse MAC unique, éliminant la nécessité d'un passage successif des données, au travers de couches complexes comme la couche réseau, la couche transport, la couche session, la couche présentation et la couche application. Cela élimine également le besoin d'ajouter, puis retirer des en-têtes à chaque couche, allégeant les temps d'attente résultants de la congestion du réseau et les délais dus aux pertes de paquets, aboutissant à un temps de réponse élevé. De plus, ESP-NOW peut coexister avec le Wifi, Bluetooth LE (faible énergie) et il supporte de multiples séries de composants SoC (System on Chip ou Système sur une puce) intégrant des fonctionnalités Wifi.

Méthodologie : afin de garantir une communication fiable ; nous avons créé un mécanisme d'accusé de réception au niveau de la couche application **(figure 3)**. Le processus de communication débute lors de l'initialisation de celle-ci par le commutateur à pile bouton. Après avoir reçu un message, le dispositif récepteur répond par un ACK (*acknowledgement* ou accusé de réception), indiquant la réussite de la réception du message, la communication est alors terminée. Si l'accusé de réception n'est pas reçu dans un temps spécifié, le commutateur tente une retransmission du message de façon répétitive. Afin de maintenir la simplification du protocole, tous les paquets sont diffusés en utilisant le protocole ESP-NOW au niveau de la couche « data link ».
L'engagement (*binding*) et le désengagement (*unbinding*) du dispositif sont accomplis par la vérification de l'adresse MAC, garantissant la sécurité et la simplicité du protocole ainsi défini.

Stratégie de changement de canal : les canaux de transmission et réception de ESP-NOW correspondent au canal utilisé par le point d'accès auquel le dispositif est relié. Le commutateur à pile bouton ne peut communiquer avec le dispositif contrôlé que s'ils utilisent le même canal. Lorsque le dispositif contrôlé est déjà connecté au point d'accès Wifi, le canal qu'il utilise suit les changements de canal du point d'accès. Le commutateur à pile bouton doit déterminer le canal en cours d'utilisation par l'équipement contrôlé. Dans un environnement réseau relativement stable, le canal du point d'accès ne change pas fréquemment.
Ainsi**,** le dispositif transmet d'abord sur le canal utilisé lors de la dernière transmission réussie. Si de multiples tentatives d'envoi échouent et qu'il est établi que l'élément récepteur a changé de canal, il répétera la transmission en utilisant chacun de canaux Wifi existants.

Modulation utilisée pour la transmission des paquets : durant la transmission des paquets Wifi, le courant instantané peut atteindre 300 mA, bien au-delà de ce qu'une pile bouton peut fournir. Pour résoudre ce problème, nous avons implémenté une stratégie de transmission intermittente des paquets. Après la transmission de chaque paquet, un délai est introduit, durant lequel le chip est mis à l'état de veille économique en énergie. Dans cet état, le courant consommé par le chip n'est que de quelques µA et la pile bouton est principalement utilisée pour recharger les condensateurs, avant de procéder à la transmission du paquet suivant.
Si l'on considère que le courant moyen consommé durant la transmission est I_0, pendant une durée t_0, et que le courant moyen correspondant à l'état de veille est I_1, pendant une durée t_1, l'intensité moyenne globale peut se calculer ainsi :

$$I = \frac{I_0 t_0 + I_1 t_1}{t_0 + t_1}$$

En faisant varier correctement la durée de I_0 et I_1, on peut faire en sorte que le courant moyen absorbé se situe en deçà de l'intensité de fonctionnement normale de la pile bouton. Cette stratégie contribue à l'obtention d'un fonctionnement plus efficace et moins énergivore du commutateur à pile bouton.

Optimisation du processus d'initialisation : le processus d'initialisation d'un dispositif Wifi nécessite plusieurs étapes depuis la mise sous tension jusqu'à l'établissement de la transmission. Nous avons réalisé une analyse profonde de chacune de ces étapes et de la durée de leur accomplissement. Par défaut, le démarrage du chip comprend l'amorçage (boot), l'initialisation du Wifi et le démarrage du Wifi. Le chargement initial est le processus le plus long, et le démarrage du Wifi celui qui consomme le plus d'énergie. Afin de réduire l'énergie consommée, nous avons effectué les optimisations suivantes :

- Désactivation de tous les enregistrements des évènements non essentiels durant le chargement initial (boot).
- Vérification flash : nous avons désactivé la vérification de la mémoire flash celle-ci n'étant pas essentielle pour cette application.
- Informations de calibration Wifi : afin d'éviter de trop fréquentes calibration Wifi, nous avons sauvegardé les informations de calibration dans la mémoire non volatile (NVS).

Le commutateur alimenté par pile bouton, basé sur un ESP32-C2 apporte une solution convenante et souple au contrôle des dispositifs domotique.

Observez les résultats ur les **tableaux 1** et **2**. Ayant implémenté ces optimisations, nous avons pu réduire la consommation moyenne durant le processus d'initialisation (chargement + Wifi init + démarrage Wifi) de 46,39 mJ à 5,61 mJ. Par ailleurs, le temps d'initialisation a diminué, passant de 536,5 ms à 63,19 ms. Pour plus d'information détaillée sur la configuration, merci de vous référer à la démonstration du commutateur à pile bouton [1].

Conception du circuit

L'ESP32-C2 nécessite une tension de fonctionnement de 3,3 V, plus élevée que la tension délivrée par la pile bouton. De ce fait, un circuit élévateur de tension doit-être conçu pour augmenter la tension. La stabilité de la tension d'alimentation impacte directement les performances de transmission des paquets et la stabilité générale de l'appareil. Un régulateur de tension bien conçu permet d'améliorer les performances de la transmission par radiofréquences et la durée de vie de la pile. Il est impératif de prendre en considération les coûts de production et les performances requises lors de la conception d'un tel circuit. Le circuit élévateur de tension doit-être étudié avec soin afin de permettre une conversion de tension efficace avec un minimum de pertes de puissance. De plus, il doit fournir une alimentation fiable et stable au circuit ESP32-C2, lui permettant un fonctionnement optimal durant les phases actives et la veille. Par ailleurs, la conception du circuit doit prendre en compte l'intensité consommée, la dissipation de chaleur et l'efficacité permettant d'atteindre un compromis correct entre performances et consommation énergétique. L'objectif ultime de la conception du circuit est l'obtention d'une solution robuste et économique respectant les besoins en alimentation de l'ESP32-C2 tout en optimisant les performances du dispositif et la durée de vie de la pile. La collaboration entre des ingénieurs experts en conception matérielle, l'utilisation des composants et techniques adaptés ont conduit à la conception réussie d'un circuit qui répond aux besoins spécifiques d'un commutateur Wifi alimenté par une pile bouton.

Conception globale du circuit

Choix du convertisseur de puissance : la pile bouton peut être considérée comme étant une source d'alimentation dont la résistance interne augmente avec sa décharge. Initialement, sa résistance interne est d'environ 10 Ω, mais elle peut atteindre une centaine d'Ohms à l'approche de la fin de son cycle de décharge. L'ESP32-C2, en tant que dispositif de sortie, nécessite une source d'alimentation pouvant fournir un courant de 500 mA, voire davantage. L'ondulation (bruit) de la tension d'alimentation peut impacter de façon importante les performances des transmissions par radiofréquences (RF). Lors de l'évaluation des ondulations parasites de l'alimentation, il est essentiel de les mesurer dans les conditions normales de transmission des paquets.

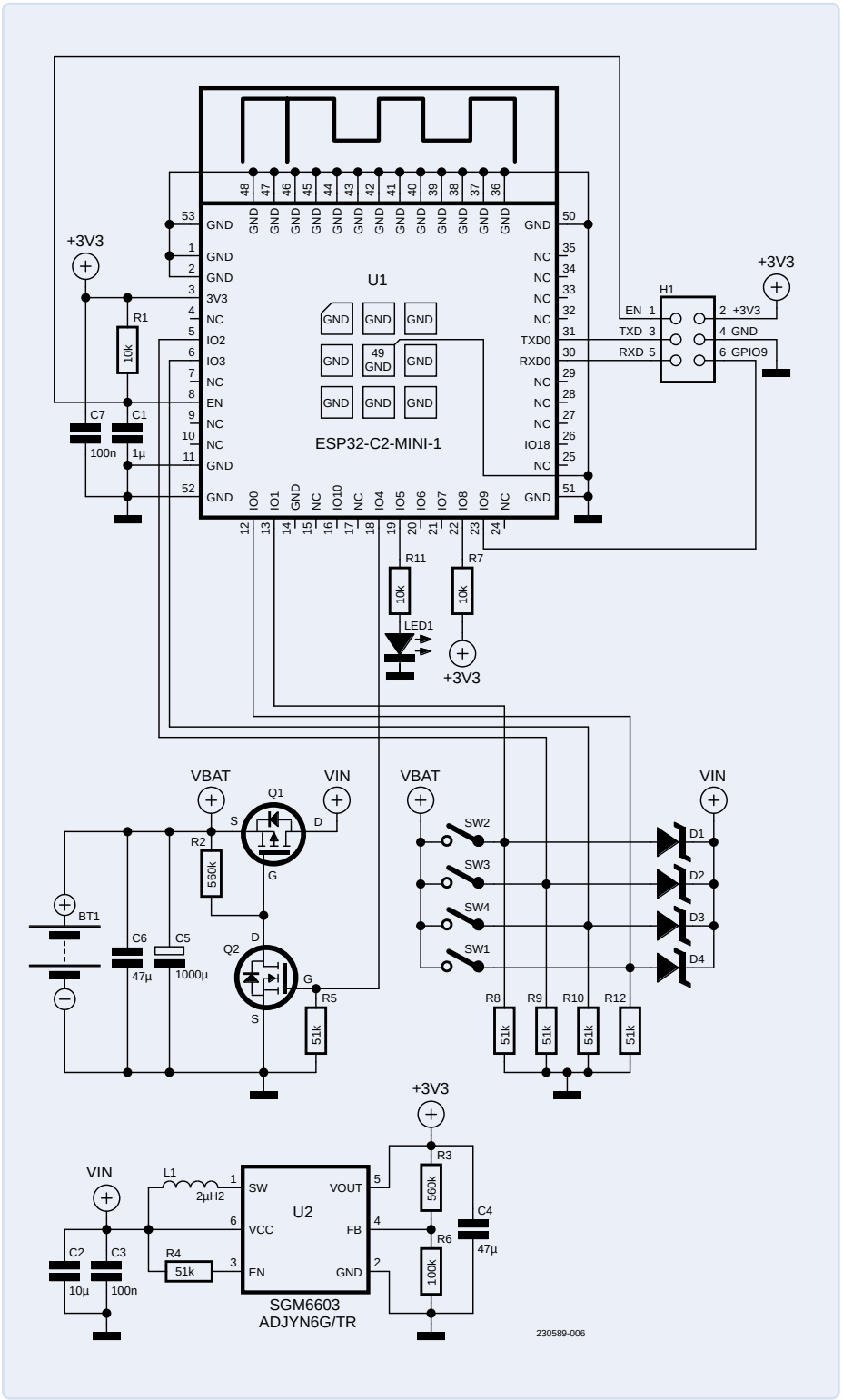


Figure 4. Schéma du dispositif commutateur à pile bouton.

Le bruit de l'alimentation peut varier selon les changements de puissance fournie. Une transmission importante de paquets peut conduire à la présence d'un bruit plus important. Pour atténuer l'impact de la résistance interne élevée de la pile bouton, la tension minimale requise par le convertisseur de tension doit-être la plus basse possible, tout en maintenant une efficacité élevée.
Vous trouverez le schéma général de cette réalisation sur la **figure 4**, le circuit élévateur de tension SGM6603 a été choisi. Il accepte une tension minimale d'entrée de 0,9 V, et délivre un courant de commutation de 1,1 A, ce qui lui permet de convenir pour la réalisation de la conversion de tension de la pile bouton en respectant les besoins d'alimentation de l'ESP32-C2.

Choix des condensateurs : il y a deux séries de condensateurs en relation avec l'alimentation : les condensateurs situés avant le

Figure 5. Oscillogrammes de la tension et du courant absorbé durant la transmission d'un paquet de données standard.

convertisseur de tension et les condensateurs situés à sa sortie. Les condensateurs placés en aval du convertisseur de tension sont reliés en parallèle avec l'alimentation du module Wifi, assurant la stabilisation de la tension de sortie et réduisant sa chute durant la transmission des paquets. L'utilisation de condensateurs de fortes valeurs assure une variation lente de la tension d'alimentation du chip. Par ailleurs, la tension aux bornes de ces condensateurs correspond à la tension d'alimentation du circuit intégré. Lorsque le chip Wifi est alimenté, les condensateurs sont chargés, lorsque le chip n'est plus alimenté, les condensateurs sont totalement déchargés. Toutefois, l'utilisation de condensateurs de valeur trop élevée, peut diminuer l'efficacité globale du circuit. Il est essentiel de rechercher un compromis entre la valeur des condensateurs et l'efficacité du système.

Les condensateurs en amont du convertisseur de tension sont reliés en parallèle avec la pile. Ils ont pour rôle principal la réduction des pics de courant fourni par la pile. Durant les périodes où l'intensité est élevée, les condensateurs deviennent la source principale d'alimentation, alors que durant les périodes de faible consommation, la pile devient la source principale, assurant la charge des condensateurs. Lorsque le convertisseur de tension n'est pas opérationnel, la faible consommation résiduelle du circuit est due au courant de fuite de ces condensateurs. Si l'on considère le volume et le courant de fuite, les condensateurs électrolytiques solides ou à l'aluminium sont un excellent choix. Par exemple, un condensateur électrolytique solide de 1 000 µF présente un courant de fuite typique d'environ 1 µA sous une tension de 3 V.

Conception de la commande d'alimentation : pour les applications dans lesquelles la durée de vie opérationnelle des dispositifs se mesure en années, le courant de veille (courant de fuite) lorsqu'ils ne fonctionnent pas devient un facteur prépondérant affectant l'autonomie globale de l'appareil. Pour répondre à cette préoccupation, le circuit comprend un interrupteur d'alimentation contrôlé, constitué de deux transistors MOSFET Q1 et Q2 sur le schéma de la figure 4. Ce circuit permet au chip d'assurer ou rompre la connexion de la pile, en déconnectant effectivement l'alimentation du module de transmission RF lorsque le dispositif n'est pas utilisé. Lorsque le dispositif doit être alimenté, un bouton poussoir est appuyé, assurant le passage du courant vers le chip à alimenter. Simultanément, le chip utilise la détection de la tension par le convertisseur CAN (Convertisseur analogique Numérique ou ADC) pour identifier le bouton qui a été pressé.

L'utilisation de ce circuit de commande de l'alimentation permet d'obtenir un contrôle énergétique efficace, réduisant toute consommation inutile durant les périodes d'inactivité, prolongeant ainsi la durée de vie de la pile de l'appareil. En déconnectant totalement l'alimentation du module radiofréquence, en dehors des périodes d'utilisation, le courant de veille du dispositif est minimisé, optimisant sa longévité dans diverses applications. En plus des composants cités, le circuit comprend également l'ESP32-C2 et les voyants à LED.

Estimation de la durée de vie de la pile

Toutes les optimisations étant effectuées, une transmission par paquets a été analysée, les tensions d'entrée et le courant du convertisseur élévateur de tension ont été enregistrées (**figure 5**). Selon les données obtenues, la transmission d'un paquet complet dure 240 ms et l'intensité absorbée durant l'opération est de 25,8 mA.

Afin d'estimer correctement la durée de vie de la pile, considérant que le rendement du convertisseur de tension durant la charge est incertain, une évaluation pratique a été menée. Dans cette étude, les transmissions de paquets ont été évaluées dans le mode spécifié. Chaque cycle a été mesuré, depuis l'instant où le dispositif est déclenché (appui du poussoir), jusqu'à la transmission réussie du signal et la réception de l'accusé réception, prenant ainsi en compte la durée totale de l'opération.

Ce test a permis de déterminer qu'une pile bouton CR2032 pouvait supporter environ 65 000 cycles de transmission par paquet. De plus, l'intensité en veille mesurée, ne dépassait pas 1 µA. En considérant que le dispositif transmette des paquets 10 fois par jour, la durée de vie effective de la pile CR2032 serait d'environ 5 ans.

Cette évaluation démontre l'efficacité de la gestion énergétique et le faible courant absorbé par le dispositif, lui permettant de convenir à des applications grand public. Grâce à l'optimisation de son mode d'alimentation et sa conception évoluée, la pile de ce dispositif a une durée de vie surprenante, lui permettant d'offrir un fonctionnement fiable et de longue durée aux applications de l'IdO et de Domotique.

Conclusions

Le commutateur alimenté par pile bouton muni d'un ESP32-C2 apporte une solution

de contrôle de dispositifs connectés souple et adéquate. En optimisant la flexibilité du contrôle de l'alimentation et les protocoles utilisés, cette solution Wifi alimentée par une simple pile bouton facilite la communication avec les autres dispositifs équipés de chips ESP. Dans nos projets futurs, nous avons l'intention d'intégrer cette technologie dans les standards Matter (précédemment connu comme Project CHIP), permettant un contrôle facile de multiples appareils alimentés par des sources conventionnelles.

N'oubliez pas de consulter la plateforme de développement GitHub d'Espressif [2] où vous trouverez davantage d'informations et des démonstrations des solutions open source ESP-IoT-Solution ainsi qu'en ce qui concerne ESP-NOW.

Ce commutateur alimenté par pile bouton est une solution améliorant l'applicabilité et l'efficacité des applications du monde de la Domotique et de l'IdO. De par sa conception optimisée et l'efficacité de son contrôle énergétique, il procure une solution domotique durable et fiable. .

VF : Jean Boyer —230589-04

Questions ou commentaires ?

Contactez librement les auteurs (zhang.wei@espressif.com ou lijunru@espressif.com) ou l'équipe éditoriale d'Elektor (redaction@elektor.fr).

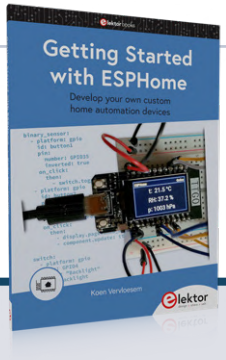
À propos des auteurs

Zhang Wei est ingénieur d'application chez Espressif Systems. Ingénieur chevronné en développement logiciel ayant une expérience de plus de dix ans dans les systèmes embarqués, les réseaux sans fil et le développement IdO, il aime apporter des solutions simples et claires aux problèmes. Diplômé en ingénierie électronique et informatique, il est titulaire d'un Master en « knowledge engineering » de l'université d'état de Singapour. Il a enrichi son expérience en étant employé chez STMicroelectronics, Greenwave Systems et Domarka Digital. En dehors de ses occupations professionnelles, Zhang Wei aime le football et les voyages.

Li Junru est ingénieur d'applications chez Espressif à Shangai. Les systèmes embarqués ont toujours été sa passion. Il a pour mission d'apporter aux passionnés l'aide nécessaire à la libération de leur créativité en utilisant l'ESP32. Le message qui guide Li Junru est le suivant : « Collaborons et nous rendrons passionnant le développement des systèmes embarqués ».

Produits

- **Espressif ESP32 range**
  www.elektor.fr/espressif
- **Getting Started with ESPHome**
  www.elektor.fr/19738



LIENS

[1] Demonstration : ESP Now coin_cell_demo [GitHub] : https://tinyurl.com/espnowcoincell
[2] Dépôt GitHub d'Espressif : https://github.com/orgs/espressif/repositories

SDK

ESP-ADF

ESPRESSIF

Espressif Audio Development Framework

Si vous construisez un appareil qui doit enregistrer ou lire de l'audio, ESP-ADF (Audio Development Framework) est le SDK qu'il vous faut. ESP-ADF fournit non seulement un support de base pour le pipelining audio, mais aussi divers encodeurs et décodeurs audio, des analyseurs de conteneurs audio, des égaliseurs et des downmixers.

En outre, différents protocoles de haut niveau, tels que DLNA, RTSP, RTCP et Bluetooth A2DP sont pris en charge, ainsi que leurs exemples correspondants. Plusieurs kits de développement offrent une prise en charge audio. Découvrez les cartes des séries ESP32-S3-Korvo et ESP32-Lyra qui facilitent le prototypage.

https://github.com/espressif/esp-adf

la maison intelligente évolue avec

libérer le potentiel de l'IdO pour les maisons intelligentes

Kedar Sovani, Espressif

Les consommateurs et les développeurs ont souvent été frustrés par les produits connectés - dans la plupart des cas, des appareils qui nécessitent des applications propriétaires pour la configuration et le contrôle et qui ne permettent ni une expérience utilisateur cohérente ni une communication interopérable. Le protocole Matter offre à l'utilisateur final un moyen sécurisé de configurer, de découvrir et de contrôler les appareils connectés. Depuis son lancement en octobre 2022, plus d'un millier de produits provenant d'un large éventail d'entreprises ont été certifiés Matter.

Vers la fin de l'année 2022, le protocole Matter pour la maison intelligente a été lancée - le résultat de plusieurs années de développement collaboratif par les principaux acteurs de l'industrie. Ces derniers mois, de nombreux appareils fonctionnant avec cette norme universelle interopérable ont été lancés. Dans cet article, nous examinons les avantages offerts par Matter, les progrès accomplis et les perspectives.

L'ère pré-Matter

En tant que consommateur - mais aussi en tant que développeur - de produits connectés, vous connaissez peut-être la frustration associée. Dans la plupart des cas, il s'agissait d'appareils propriétaires dont la configuration et le fonctionnement nécessitaient des applications dédiées et qui n'offraient ni une expérience utilisateur cohérente, ni une communication interopérable. La seule caractéristique commune était la commande vocale de divers écosystèmes qui disposaient d'un support intégré pour ces protocoles propriétaires.

Du côté des développeurs/fabricants d'appareils, cela signifiait que le coût global de création d'un produit de qualité était beaucoup plus élevé, car les développeurs devaient se conformer à différentes organisations et obtenir des certifications de leur part.

Matter entre en scène

Plusieurs acteurs du secteur de la maison intelligente se sont rendu compte que ces problèmes pénalisaient la valeur et la croissance de la maison intelligente. Cette collaboration à l'échelle du secteur a abouti au lancement du protocole Matter à la fin de l'année 2022.

Qu'offre exactement Matter ? Matter offre un moyen sécurisé pour les appareils connectés d'être configurés, découverts et contrôlés par l'utilisateur final.

Depuis son lancement en octobre 2022, plus de 1 000 produits ont

été certifiés Matter, tandis que la Connectivity Standards Alliance (CSA) se targue de compter plus de 300 entreprises contribuant à l'effort commun de développement et de déploiement de Matter. Grâce à cet énorme effort, de nombreux appareils connectés prennent en charge Matter immédiatement.
Il est possible de configurer et de contrôler les appareils compatibles avec des dispositifs communément appelées contrôleurs Matter (téléphones, haut-parleurs, écrans). Plusieurs écosystèmes, tels qu'iOS, Android, Alexa et SmartThings, prennent déjà en charge les contrôleurs Matter par défaut. Cela permet aux consommateurs de commencer à configurer et à contrôler les appareils connectés compatibles Matter sans avoir à acheter un contrôleur Matter séparé ou à installer une application distincte (**figure 1**).

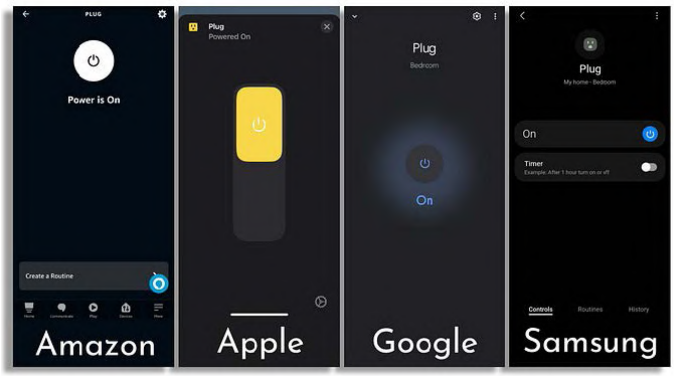


Figure 1. Un Matter Plug tel qu'il apparaît dans différentes apps de l'écosystème.

Sécurité et vie privée

Pour de nombreux appareils connectés que nous utilisons aujourd'hui, nous, consommateurs, sommes en quête d'informations sur les pratiques et les principes de sécurité appliqués au cours du développement et de la gestion, du produit connecté. La sécurité et la protection de la vie privée ont été au cœur de la spécification Matter. Un grand nombre d'organisations (avec une expérience de plusieurs années dans le secteur de la maison intelligente) ont collaboré à l'élaboration de ces spécifications et ont veillé à ce que rien ne soit négligé pour offrir aux utilisateurs ce qu'il y a de mieux. Les utilisateurs des produits Matter peuvent être assurés que chaque aspect du comportement de l'appareil, de la configuration initiale au fonctionnement et à la gestion ultérieurs, a été développé conformément aux meilleures normes et pratiques en matière de sécurité.

Réseau local

L'architecture de Matter diffère fondamentalement de celle des appareils connectés avant Matter. Auparavant, chaque appareil connecté était généralement contrôlé sur le réseau local par un appareil, tel qu'un téléphone, du même réseau. Dans tous les autres cas, les appareils étaient connectés à une plateforme cloud et une application sur un téléphone ne se trouvant pas sur le même réseau communiquait avec ces appareils via le cloud. L'intégration avec les assistants vocaux se faisait également par une communication de nuage à nuage, comme le montre la **figure 2**.

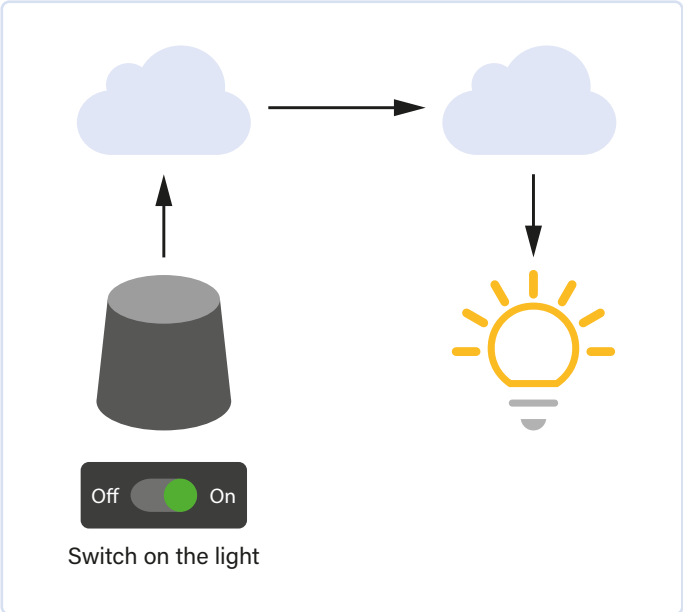


Figure 2. Exemple de communication (sans Matter).

Matter est un protocole qui fonctionne uniquement sur le réseau local. Il prend en charge la configuration initiale, la découverte et le fonctionnement des appareils connectés sur le même réseau local. Les appareils connectés eux-mêmes n'ont pas besoin de communiquer avec un service cloud, comme c'était le cas avec les appareils antérieurs à Matter. L'intégration de l'assistant vocal fonctionne donc directement en envoyant les commandes Matter appropriées sur le réseau local (**figure 3**). Il revient donc également aux contrôleurs Matter de décider comment contrôler ces appareils à distance.

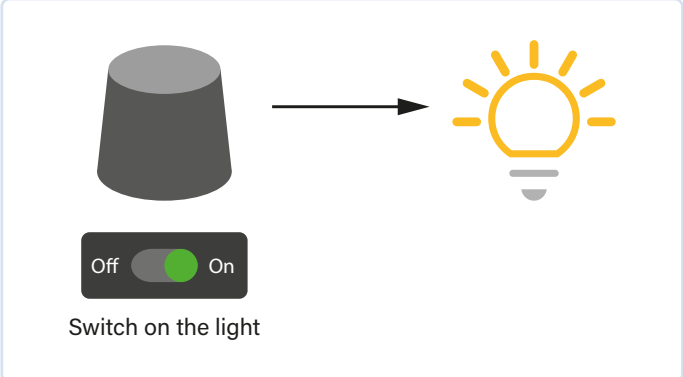


Figure 3. Exemple de communication (Matter).

Cela permet à l'utilisateur final de choisir les appareils et les écosystèmes Matter auxquels il confie sa vie privée et sa sécurité.

Transfert de données

Matter fournit ses fonctions via les protocoles de transfert Wifi et Thread (802.15.4), qui sont omniprésents dans la maison intelligente. Chacun de ces protocoles a ses propres avantages.
Le réseau Wifi étant omniprésent, il est facile d'intégrer les appareils Wifi Matter dans les réseaux existants.
Les appareils basés sur Thread (802.15.4) sont mieux adaptés aux applications à faible consommation d'énergie, telles que les capteurs, où le cycle d'utilisation des données est beaucoup plus faible. Les appareils Matter basés sur Thread ont besoin d'un routeur frontalier Thread pour faire partie du réseau. La plupart des haut-parleurs/écosystèmes intelligents existants prennent en charge Thread et mettent à jour leur logiciel pour exploiter la fonction de routeur frontalier. Cela facilite grandement l'intégration des appareils Matter dans le réseau domestique.

Mise en service

Tous les appareils connectés doivent d'abord être connectés au réseau de l'utilisateur avec un mécanisme de provisionnement. Ce processus est souvent associé à des erreurs, à une mauvaise configuration et constitue une source de frustration pour l'utilisateur. Les appareils Matter sont connectés grâce à un processus appelé mise en service Matter. Compte tenu de la prise en charge standard de Matter dans la plupart des écosystèmes, ce processus est robuste et sophistiqué et offre une expérience fluide à tous les utilisateurs. Sous le capot, la plupart des appareils Matter rendent la fonctionnalité de mise en service disponible via Bluetooth Low Energy (BLE). Les contrôleurs Matter transfèrent les informations d'identification du réseau prévu (Wifi ou Thread) via une liaison BLE sécurisée.
Les appareils peuvent également offrir une mise en service Matter via Ethernet ou Wifi s'ils font déjà partie du réseau de l'utilisateur. Ce mécanisme est essentiel pour mettre à niveau les appareils existants afin qu'ils puissent offrir une prise en charge Matter à leurs utilisateurs finaux.

Attestation

L'attestation de l'appareil est une fonction de sécurité importante de Matter. Il s'agit d'un processus qui a lieu lors de la mise en service d'un appareil Matter, au cours duquel le contrôleur Matter établit l'authenticité de l'appareil. Chaque appareil Matter est programmé avec un ensemble unique de certificats de sécurité spécifiques au fabricant. Le processus d'attestation Devive garantit que l'appareil provient bien du fabricant dont il se réclame. En cas de divergence, l'utilisateur final est averti du problème d'authentification de l'appareil. Cela rend difficile la création et la mise en service de produits Matter contrefaits, protégeant ainsi les utilisateurs et leurs données.

Communication entre appareils

Avant Matter, la plupart des communications entre appareils étaient organisées via le cloud ou par des assistants vocaux. Tous les appareils devaient donc communiquer avec le même cloud ou posséder des intégrations de cloud à cloud pour créer des écosystèmes à valeur ajoutée qui faisaient automatiquement ce qu'il fallait.
L'un des principaux avantages des appareils Matter est que la communication entre les appareils s'effectue entièrement au sein du réseau local, c'est-à-dire qu'il n'y a pas de communication dans le cloud. Il est possible de configurer un appareil Matter d'un fabricant pour qu'il communique avec un appareil Matter d'un autre fabricant. Par exemple, vous pouvez créer des automatismes pour commuter les lumières (ou un climatiseur) en fonction de l'état d'un interrupteur ou d'un capteur. Cela peut même fonctionner si les appareils utilisent des protocoles différents (l'un utilise le Wifi tandis que l'autre utilise Thread). La possibilité de définir ces automatismes (appelés " bindings " dans le langage Matter) à partir de n'importe quel écosystème ou application supporté par Matter facilite encore plus la concrétisation de la valeur de la maison intelligente (**figure 4**).

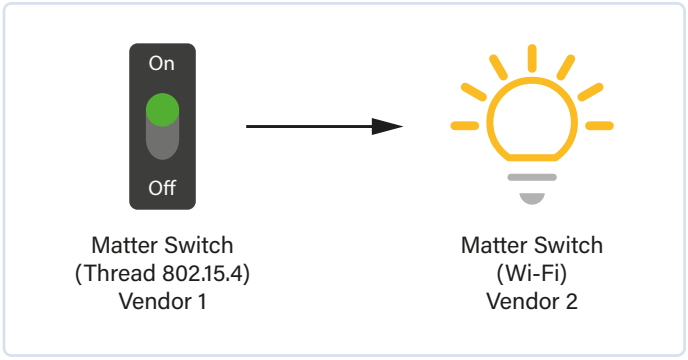


Figure 4: Liaison Matter : communication appareil à appareil.

Coexistence des écosystèmes

Un autre problème auquel sont confrontées la plupart des maisons est l'hétérogénéité des écosystèmes utilisés par les habitants. La plupart des utilisateurs ont leurs propres préférences en matière d'écosystèmes, tels qu'Android, iOS, Alexa, GVA, SmartThings ou Home Assistant. Cela conduit à une expérience utilisateur fragmentée, où certains appareils connectés ne peuvent faire partie que d'un seul écosystème ou n'offrent que des fonctionnalités de second ordre dans d'autres écosystèmes. En revanche, un appareil Matter peut faire partie de plusieurs écosystèmes à la fois. Vous pouvez ajouter un appareil Matter à un maximum de cinq écosystèmes distincts. Tous les utilisateurs de l'écosystème peuvent bénéficier simultanément des mêmes avantages que ceux offerts par les appareils Matter. Même une notification de changement d'écosystème est reçue par les autres.

Mise à jour à distance du micrologiciel (OTA)

La mise à jour OTA du micrologiciel permet à un produit non seulement d'intégrer les derniers correctifs de sécurité ou de fonctionnalité, mais aussi d'activer de nouvelles fonctions au fil du temps. La spécification Matter prévoit un livre distribué dans lequel les fabricants peuvent télécharger et conserver des images de mise à jour du micrologiciel OTA pour leurs produits. Le grand livre distribué est accessible à tous les contrôleurs Matter, qui peuvent télécharger et déployer les mises à jour de micrologiciel nécessaires aux appareils Matter, avec l'accord de leurs utilisateurs.

Certification

Matter a mis en place un processus de certification strict pour les appareils. La présence de le badge Matter sur un appareil connecté indique que l'appareil a subi et réussi tous les tests requis pour obtenir la certification Matter. La certification Matter permet de garantir que l'appareil est interopérable avec d'autres appareils

et contrôleurs Matter et qu'il répond à un niveau minimum de robustesse et de fonctionnalité souhaité par les consommateurs. Les consommateurs de produits Matter n'ont plus à se fier au fabricant pour ce qui est de ces exigences de base. Tous les fabricants de produits Matter sont tenus de respecter les mêmes normes de robustesse et d'interopérabilité.

Quelles sont les nouveautés de Matter 1.2 ?

La version 1.2 de Matter est sortie récemment. La première version de Matter commençait par prendre en charge les appareils les plus courants, tels que les lampes, les prises de courant, les fiches, les stores et les thermostats. Le dernier standard Matter 1.2 inclut la prise en charge des appareils électroménagers, y compris les machines à laver, les réfrigérateurs, les climatiseurs, les lave-vaisselle et les aspirateurs robotisés. La prise en charge des détecteurs de fumée et de monoxyde de carbone, des capteurs de qualité de l'air, des purificateurs d'air et des ventilateurs est également incluse. Cette prise en charge comprend des commandes spécifiques supplémentaires pour ces appareils, en plus de la commande marche/arrêt habituelle.
La spécification de base Matter comprend également des mises à jour permettant la création plus souple d'appareils à partir de ses sous-composants. Cela permet une modélisation plus précise d'une variété d'appareils. La nouvelle prise en charge des étiquettes sémantiques offre un moyen interopérable de décrire l'emplacement ou les fonctions sémantiques d'un appareil

Qu'en est-il des développeurs ?

La norme Matter est publiquement disponible au téléchargement (après avoir rempli un formulaire). Le SDK Matter C++ est hébergé sur GitHub [1], et tout le développement se fait sur GitHub. Le SDK implémente à la fois des fonctionnalités côté micrologiciel (appareil) et côté contrôleur. Des implémentations expérimentales pour JavaScript [2] et Rust [3] sont également en cours. Cela permet aux développeurs d'accéder facilement aux spécifications et d'apporter des améliorations au dépôt.
Avec des solutions comme ESP-Launchpad [4] et ESP-ZeroCode [5], les développeurs peuvent facilement essayer et expérimenter la convivialité de Matter en flashant le micrologiciel sur l'un des nombreux kits de développement matériel.
Cela permet aux développeurs de construire et d'expérimenter Matter pour l'appareil qu'ils souhaitent fabriquer. Une fois que le micrologiciel et le matériel sont disponibles, tous les écosystèmes Matter permettent aux développeurs d'évaluer ces appareils sans avoir besoin d'un micrologiciel de production ou de certifications de production. Cela facilite grandement les modifications dans le développement des appareils Matter.
Pour les développeurs d'applications mobiles, les dernières versions d'iOS [6] et d'Android [7] incluent des API qui permettent à ces applications d'accéder aux API Matter. Les développeurs d'applications peuvent utiliser cette fonction pour offrir des caractéristiques et des fonctions plus riches que celles offertes par les systèmes d'exploitation de base des téléphones.
J'espère que cela vous incitera, vous les développeurs, à découvrir les appareils Matter et à vous lancer dans le développement de vos propres appareils.

230617-04

Questions ou commentaires ?

Envoyez un courriel à l'auteur (kedar.sovani@espressif.com) ou contactez Elektor (redaction@elektor.fr).

À propos de l'auteur

Kedar Sovani a plus de 21 ans d'expérience dans les domaines du logiciel système, de la sécurité et de l'IdO. Il travaille depuis six ans chez Espressif. Il est directeur principal ingénierie, spécialisé dans les écosystèmes IdO, où son travail consiste à identifier et à construire des plateformes et des solutions qui aident les développeurs à créer des produits IdO plus rapidement, et de manière plus robuste et plus sûre. Développeur pratique, il travaille actuellement sur Matter et Rust.

Produits

- **ESP32-DevKitC-32E**
  www.elektor.fr/20518
- **ESP32-C3-DevKitM-1**
  www.elektor.fr/20324

LIENS

[1] SDK Matter C++ sur GitHub : https://github.com/project-chip/connectedhomeip
[2] Implémentation pour JavaScript : https://github.com/project-chip/matter.js
[3] Implémentation pour Rust : https://github.com/project-chip/rs-matter
[4] ESP-Launchpad : https://espressif.github.io/esp-launchpad
[5] ESP-ZeroCode : https://zerocode.espressif.com
[6] Home Mobile SDK pour iOS : https://developer.apple.com/documentation/matter
[7] Home Mobile SDK pour Android : https://developers.home.google.com/matter/apis/home

Mettez la main sur le nouveau matériel ESPRESSIF

Il n'y a rien qui nous excite plus que de mettre la main sur du nouveau matériel, et cette collaboration avec Espressif a été un vrai régal ! Vous voulez en faire l'expérience vous-même ? Elektor a approvisionné ses magasins pour offrir tous les produits présentés dans cette édition !

ESP32-S3-Box-3

L'ESP32-S3-BOX-3 est un kit de développement AIoT entièrement open-source basé sur le puissant SoC AI ESP32-S3, et conçu pour révolutionner le domaine des cartes de développement ordinaires. L'ESP32-S3-BOX-3 est livré avec un ensemble riche de modules complémentaires, permettant aux développeurs de personnaliser et d'étendre facilement les applications de ce kit.

www.elektor.fr/20627

ESP32-S3-Eye

L'ESP32-S3-EYE est une carte de développement IA de petit format. Elle est basée sur le SoC ESP32-S3 et sur l'ESP-WHO, le cadre de développement IA d'Espressif. Elle comporte un appareil photo de 2 mégapixels, un écran LCD et un microphone, utilisés pour la reconnaissance d'images et le traitement audio.

www.elektor.fr/20626

ESP32-S3-DevkitC-1

L'ESP32-S3-DevKitC-1 est une carte de développement pour débutants équipée de l'ESP32-S3-WROOM-1, l'ESP32-S3-WROOM-1U ou l'ESP32-S3-WROOM-2, un module microcontrôleur polyvalent Wifi + Bluetooth Low Energy qui intègre des fonctions Wifi et Bluetooth LE complètes.

www.elektor.fr/20697

ESP32-C3-DevKitM-1

ESP32-C3-DevKitM-1 est une carte de développement pour débutants basée sur l'ESP32-C3-MINI-1, un module nommé ainsi en raison de sa petite taille. Cette carte intègre des fonctions Wifi et Bluetooth LE complètes. La plupart des broches d'E/S du module ESP32-C3-MINI-1 sont disponibles sur les connecteurs des deux côtés de la carte pour faciliter l'interfaçage. Les développeurs peuvent connecter les périphériques avec des fils de connexion ou monter l'ESP32-C3-DevKitM-1 sur une plaque d'essai.

www.elektor.fr/20324

ESP32-Cam-CH340

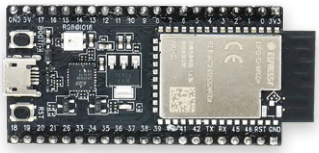
La carte de développement ESP32-Cam-CH340 est adaptée aux applications IdO, telles que les appareils intelligents, le contrôle industriel sans fil, la surveillance sans fil, la lecture de code QR sans fil, les signaux de systèmes de positionnement sans fil et d'autres applications de l'Internet des objets.

www.elektor.fr/19333

Elektor Cloc 2.0 Kit

Cloc est un réveil basée sur l'ESP32, facile à construire, qui se connecte à un serveur de temps et contrôle la radio et la TV. Il est doté d'un afficheur rétro double à 7 segments avec une luminosité variable. L'un des écrans affiche l'heure actuelle, l'autre l'heure de l'alarme.

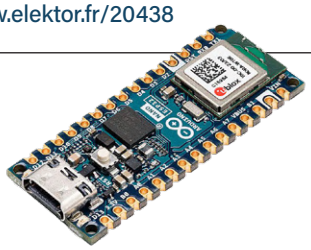
www.elektor.fr/20438



ESP32-S2-Saola-1M

L'ESP32-S2-Saola-1M est une carte de développement de petite taille basée sur l'ESP32-S2. La plupart des broches d'E/S sont disponibles sur les connecteurs des deux côtés pour faciliter l'interfaçage. Les développeurs peuvent connecter les périphériques avec des fils de connexion ou monter l'ESP32-S2-Saola-1M sur une plaque d'essai. L'ESP32-S2-Saola-1M est équipée du module ESP32-S2-WROOM, un puissant module microcontrôleur Wifi générique doté d'un riche ensemble de périphériques.

www.elektor.fr/19694



Arduino Nano ESP32

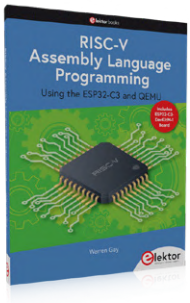
L'Arduino Nano ESP32 est une carte au format de l'Arduino Nano basée sur l'ESP32-S3 (intégré dans le NORA-W106-10B d'u-blox). Il s'agit de la première carte Arduino entièrement basée sur un ESP32. Elle offre aussi les fonctions Wifi, Bluetooth LE, le débogage via l'USB natif dans l'EDI Arduino ainsi qu'une faible consommation d'énergie.

www.elektor.fr/20562

MakePython ESP32 Development Kit

Le kit MakePython ESP32 est un kit de développement indispensable pour la programmation MicroPython de l'ESP32. En plus de la carte de développement MakePython ESP32, le kit contient les composants électroniques de base et les modules dont vous avez besoin pour commencer à programmer. Avec les 46 projets du livre accompagnant le kit, vous pouvez vous attaquer à des projets électroniques simples avec MicroPython sur ESP32 et réaliser vos propres projets IdO..

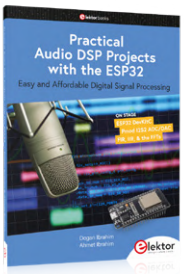
www.elektor.fr/20137



RISC-V Assembly Language Programming using ESP32-C3 and QEMU ((+ carte ESP32 RISC-V GRATUITE)

La puce ESP32-C3 d'Espressif permet d'acquérir une expérience pratique de RISC-V. L'émulateur QEMU à source ouverte permet d'acquérir une expérience de RISC-V 64 bits sous Linux. Les étudiants et les passionnés trouveront dans ce livre deux façons d'explorer RISC-V. Les projets de ce livre sont réduits à l'essentiel pour que les concepts du langage assembleur restent clairs et simples.

www.elektor.fr/20296



Practical Audio DSP Projects with the ESP32

L'objectif de ce livre est de présenter les principes de base du traitement numérique du signal (DSP) et de l'introduire d'un point de vue pratique en utilisant un minimum de calcul. Seul le niveau de base de la théorie des systèmes à temps discret est abordé, ce qui est suffisant pour réaliser des applications DSP en temps réel. Les applications pratiques utilisent la fameuse carte de développement à microcontrôleur ESP32 DevKitC et sont décrites en temps réel.

www.elektor.fr/20558

MicroPython for Microcontrollers

Les microcontrôleurs puissants tels que l'ESP32 offrent d'excellentes performances ainsi que des fonctions Wifi et Bluetooth à un prix abordable. Grâce à ces caractéristiques, la scène des maker a été prise d'assaut. Par rapport à d'autres contrôleurs, les capacités des mémoires flash et SRAM de l'ESP32 beaucoup plus grandes et la vitesse de l'unité centrale est plus élevée. Grâce à ces caractéristiques, la puce convient non seulement aux applications en C classiques, mais aussi à la programmation en MicroPython. Ce livre présente les applications des systèmes monopuces modernes.

www.elektor.fr/19736

l'essor de la maison intelligente connectée

Amey Inamdar, Espressif

Bien que les techniques de connectivité aient évolué et soient devenues plus accessibles, plus sûres et plus efficaces, la maison intelligente n'en est qu'à ses débuts par rapport à la vision qui s'est formée au cours de la dernière décennie. Nous constatons le développement de nombreux appareils, notamment des thermostats intelligents, des systèmes de sécurité, des appareils électroménagers intelligents, des éclairages intelligents et des assistants vocaux. Pourtant, en termes d'adoption, la technologie a encore un long chemin à parcourir. Récemment, deux avancées majeures ont permis d'accélérer l'adoption, et nous allons examiner ces avancées et leurs effets dans cet article.

La consommation d'énergie, le coût, la facilité de développement et les options de connectivité sont des éléments importants à prendre en compte lors de la conception d'un appareil domotique. Aujourd'hui, bon nombre de ces problèmes ont été résolus dans une large mesure. La question qui se pose alors est la suivante : qu'est-ce qui empêche l'adoption à grande échelle de ces appareils ? Les réponses à cette question sont multiples, mais la plus importante est probablement la valeur perçue des appareils intelligents, c'est-à-dire les cas d'utilisation auxquels ils donnent accès. Ces appareils sont-ils intelligents ou offrent-ils simplement la possibilité d'un contrôle à distance ? Une autre raison clé pourrait être les préoccupations en matière de protection de la vie privée et de sécurité. La voix constitue désormais une interface naturelle pour interagir avec la maison intelligente, mais elle a un prix, la perte de confidentialité.

C'est là que les deux avancées majeures survenues récemment offrent une lueur d'espoir. Nous constatons la grande transformation que l'IA générative et les grands modèles de langage (LLM) apportent à différents domaines. Les grands modèles de langage ont le potentiel d'améliorer l'efficacité et l'autonomie des appareils intelligents, en décentralisant la prise de décision et en l'amenant à la périphérie. Ces modèles ont également le potentiel de contribuer à l'amélioration de la protection de la vie privée. À titre d'exemple, si l'on considère les interfaces vocales actuelles, elles sont soit pilotées par des inférences basées sur le cloud, soit par une interface vocale basée sur des commandes prédéfinies exigeant de l'utilisateur qu'il se souvienne des commandes exactes nécessaires au fonctionnement de l'interface. Mais les LLM fournissant des modèles hors ligne tels que Whisper.ai capables d'interpréter différentes langues et de faire des inférences localement, ont le potentiel de fournir des interfaces vocales naturelles pour les maisons intelligentes qui peuvent fonctionner complètement hors ligne. Nous voyons déjà quelques projets open-source s'engager dans cette direction.

Ces progrès dans le domaine de l'IA générative sont renforcés par la capacité des petits microcontrôleurs à exécuter des modèles de ML en périphérie. Il n'est pas rare aujourd'hui de voir des microcontrôleurs dotés de moteurs d'accélération d'IA économes en énergie, capables d'accélérer l'exécution des modèles de ML. Cela permet aux capteurs à faible consommation d'énergie non seulement de détecter les données, mais aussi de les traiter pour en extraire le sens.

Cependant, le simple fait de disposer de cette intelligence pour les appareils IdO de la maison intelligente sera inutile si tous les appareils ne communiquent pas dans la même langue. C'est là que la normalisation est indispensable. Le manque de normalisation est actuellement l'un des principaux obstacles à l'adoption massive des appareils IdO domestiques intelligents. Le fait que les appareils de différents fabricants ne puissent pas communiquer entre eux limite les cas d'utilisation pour le consommateur, ce qui rend leur configuration et leur utilisation difficiles. C'est là que les récents efforts de normalisation jouent un rôle important. Si un protocole tel que Matter est adopté, les appareils pourront parler le même langage, ce qui permettra de créer une maison intelligente plus centrée sur le consommateur, avec de meilleurs cas d'utilisation. Un autre impact indirect de la normalisation est qu'elle permet aux développeurs de créer de la valeur à un niveau plus élevé que celui offert par les fabricants d'appareils.

Ainsi, l'IA et la normalisation sont toutes deux importantes pour résoudre les problèmes clés de la valeur et de la confidentialité des appareils intelligents. Elles doivent être accompagnées d'innovations dans les domaines de la connectivité, des capteurs et de l'informatique, afin de permettre la prolifération d'appareils intelligents sécurisés et peu coûteux. Nous avons été témoins de nombreuses avancées de ce type. Par exemple, la norme Wi-Fi 6 permet de créer des appareils connectés fonctionnant sur batterie sans compromettre la bande passante. Avec le développement des capteurs mmWave et UWB, la détection de l'occupation des lieux est améliorée sans qu'il soit nécessaire d'installer des caméras partout. Les systèmes d'étiquetage en matière de cybersécurité gagnent également du terrain et plusieurs régions et pays proposent un moyen clair d'étiqueter les appareils intelligents en leur attribuant une cote de sécurité permettant aux consommateurs de faire un choix éclairé.

Il n'existe pas de solution miracle pour faire de la maison intelligente omniprésente une réalité. Mais il est fort probable que nous prenions aujourd'hui une meilleure direction qu'auparavant. Il s'agit certainement d'une période et d'un domaine intéressants pour observer l'évolution et jouer nos rôles respectifs pour améliorer le monde.

230655-04

MACNICA
ATD EUROPE

Your official authorized distributor
in Europe for Espressif Systems

ESPRESSIF

Macnica ATD Europe
+49 (0)89 899 143-11
sales.mae@macnica.com
www.macnica-atd-europe.com

The Next Era of **Microcontrollers**

ESPRESSIF

High Performance MCU

With RISC-V Dual-Core Upto 400MHz

AI Acceleation

High-Speed **MEMORY**

Powerful **IMAGE & VOICE** Processing Capabilities

HMI Capabilities

- MIPI-CSI with ISP
- MIPI-DSI - 1080P
- Capacitive Touch
- H.264 Encoding - 1080P@30fps
- Pixel Processing Accellerator

Best-in-Class Security

- Cryptographic Accelerators
- Secure Boot, Flash Encryption
- Private Key protection
- Access Controls

ESP32-P4

Connectivity

- USB2.0 High Speed
- Ethernet
- SPI
- SDIO3.0
- UART
- I2C, I2S

.....


IP Camera


Touch Panel


Video Door bell


Robotic Control


Industrial Robot

www.espressif.com

Learn More About **ESP SoCs**