L'internet des objets libéré : RISC-V, AWS et FreeRTOS

Avra Saslow (États-Unis)

Au cours des décennies, la philosophie de l'open source s'est révélée d'une importance considérable dans les communautés logicielles et scientifiques. Le principe est que tout le monde peut contribuer à améliorer un projet/produit, ce qui finit par le rendre plus accessible et plus fiable. En encourageant les experts à travailler ensemble sur un même projet, la technologie open source garantit la qualité et la sécurité et permet in fine d'obtenir un excellent produit sans limitation de propriété. Cet article est un guide pour la mise en œuvre concrète de ces concepts dans une application en temps réel.

Retour sur RISC-V et FreeRTOS

Après des années de succès dans l'industrie du logiciel, l'industrie du matériel informatique dispose enfin d'une technologie *open source*, RISC-V, qui pourrait changer complètement l'avenir des microcontrôleurs. Traditionnellement, l'ISA (architecture du jeu d'instructions) d'une carte décrit la manière dont le logiciel et le matériel communiquent entre eux, et est généralement verrouillée par des licences, des redevances et des NDA (accords de non-divulgation).

Par opposition, l'ISA RISC-V est fournie sous licence open source, même si le SoC (système sur puce) n'améliore pas forcément les performances de calcul (bien qu'il soit encore parmi les plus rapides du marché). L'architecture RISC-V semble ainsi prête à bouleverser complètement le modèle commercial de la technologie. Plutôt que de choisir un ISA et donc de s'enfermer dans la bibliothèque du fournisseur, RISC-V permet aux entreprises de personnaliser, dimensionner et adapter le noyau à leurs besoins spécifiques.

Le projet décrit ici capitalise sur le matériel open source en mettant en œuvre FreeRTOS, qui est un noyau de système d'exploitation en temps réel pour les systèmes embarqués. FreeRTOS permet d'implémenter des solutions flexibles de multithreading ; au lieu d'utiliser des bibliothèques autonomes découplées, FreeRTOS permet à l'utilisateur de maintenir et de porter du code vers différentes applications au fil du temps.

Voyons les choses ainsi… lorsqu'on utilise un microcontrôleur courant comme un Arduino, la structure traditionnelle d'exécution d'un programme est une boucle sans fin qui contient toutes les tâches du système. Lorsque le programme démarre, il effectue une fonction de configuration puis exécute en boucle une série de tâches dans l'attente d'une interruption.

Ces tâches peuvent interroger des capteurs, écrire sur des écrans ou faire des calculs ; peu importe ce qu'elles font, elles le font de manière séquentielle. Cette structure de programme est excellente car elle ne nécessite pas de grosses dépenses d'énergie.

Imaginons que vous deviez effectuer plusieurs tâches simultanément. Même si la structure décrite ci-dessus peut aller vite, elle ne fonctionne pas en parallèle. C'est là qu'un système d'exploitation en temps réel comme FreeRTOS devient utile, car ses temps de latence pour les interruptions et la commutation de fils de traitement (*threads)* sont minimes.

Une latence d'interruption minimale signifie que le système d'exploitation (OS) a optimisé le temps minimum entre les exécutions de tâches ou de sous-programmes. La latence minimale de commutation de fils est le temps minimum nécessaire à l'OS pour commuter le CPU (un seul CPU) afin d'exécuter un autre sous-programme. Ce

Figure 1. Fil unique ou fils multiples, mono-thread vs. multithreading.

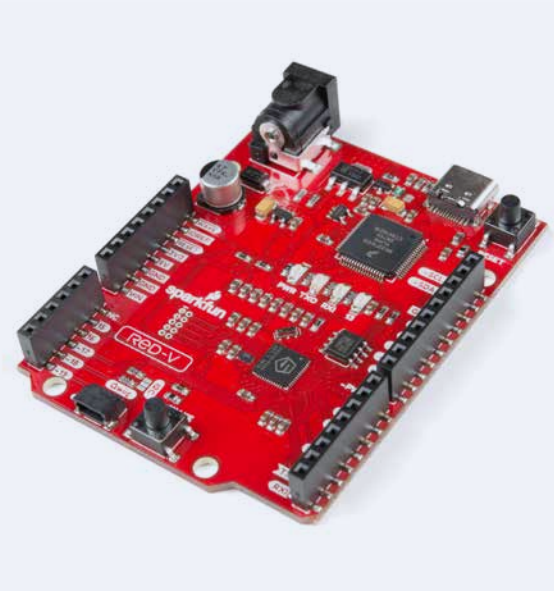

Figure 2. La carte SparkFun RED-V est basée sur une bête de calcul RISC-V.

sont essentiellement ces latences minimales qui font d'un RTOS un OS multitâche, capable d'exécuter des tâches simultanément plutôt que séquentiellement (**fig. 1**).

Bien que vous n'ayez pas besoin d'un RTOS pour les programmes qui interrogent des capteurs, effectuent des calculs sur les données et écrivent sur un écran LCD, le multithreading peut devenir très utile pour des cas comme l'interaction avec les piles logicielles d'un réseau sans fil qui exigent une réponse immédiate aux événements. Spécialement conçu pour les appareils embarqués, FreeRTOS est donc parfaitement adapté pour charger son noyau d'OS sur une carte RISC-V et communiquer dans les AWS (*Amazon Web Services*™) par le biais de bibliothèques sans fil.

Configuration du µC RED-V avec le noyau et les bibliothèques FreeRTOS et déploiement dans le nuage AWS

Pour créer une application utile dans le monde d'aujourd'hui, il faut nécessairement un modèle de base avec un dispositif IdO qui s'appuie sur une plate-forme de services sécurisée du nuage. Cette extension du noyau FreeRTOS tire parti des bibliothèques de protocoles d'application qui fournissent la connectivité nécessaire à la création d'applications IdO à partir de dispositifs à microcontrôleurs, comme la carte SparkFun RED-V (**fig. 2**) construite sur RISC-V. La structure à laquelle nous devons adhérer pour notre application est illustrée sur la **fig. 3**.

Heureusement, FreeRTOS peut être facilement mis en œuvre avec AWS ! Pour configurer une connexion à un serveur HTTP via AWS IoT, vous devez créer un utilisateur dans la section *Identity and Access Management* (IAM) d'AWS. Cet utilisateur aura des autorisations pour accéder à la fois à AmazonFreeRTOSFullAccess et à AWSIoTFullAccess - deux politiques d'accès à AWS. La carte RED-V doit également être enregistrée auprès d'AWS IoT, ce qui implique d'avoir :

- une politique AWS IoT qui accorde à votre appareil des autorisations d'accès aux ressources AWS IoT ;
- un objet AWS IoT qui vous permet de gérer vos appareils dans AWS IoT ;
- une clé privée et un certificat qui permettent à votre appareil de s'authentifier avec AWS IoT.

Tout cela peut sembler lourd, mais on peut se connecter rapidement en utilisant le processus Quick Connect (**fig. 4**) de la console FreeRTOS. Vous pouvez également passer chacune des étapes manuellement via la ligne de commande. La dernière étape pour configurer la carte RED-V en tant que dispositif IoT dans AWS consiste à télécharger FreeRTOS comme OS pour l'architecture RISC-V.

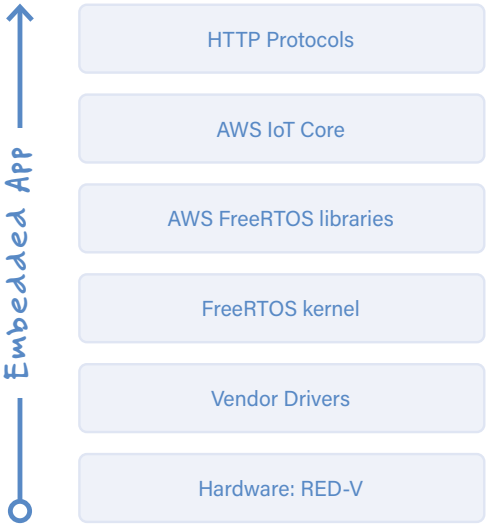

Figure 3. Couches d'application intégrées.

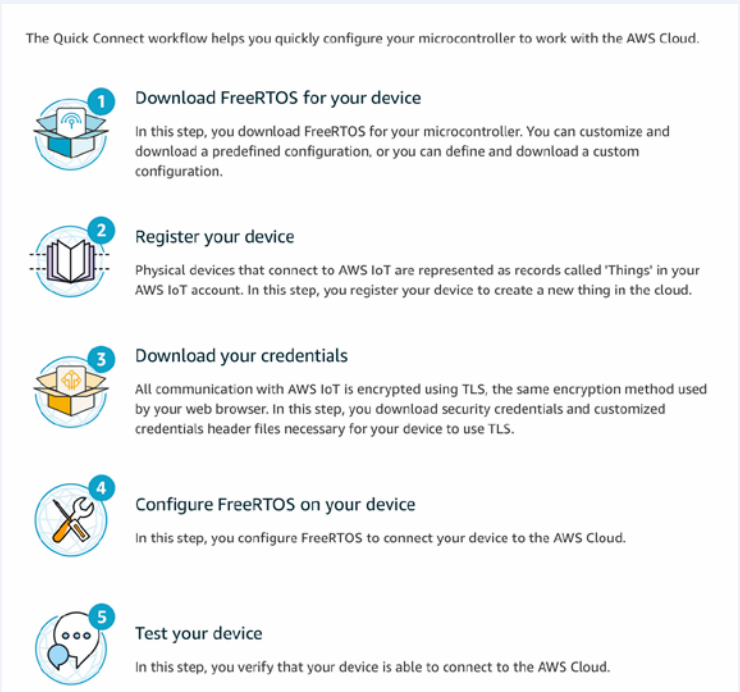

Figure 4. Le processus Quick Connect. (source : AWS)

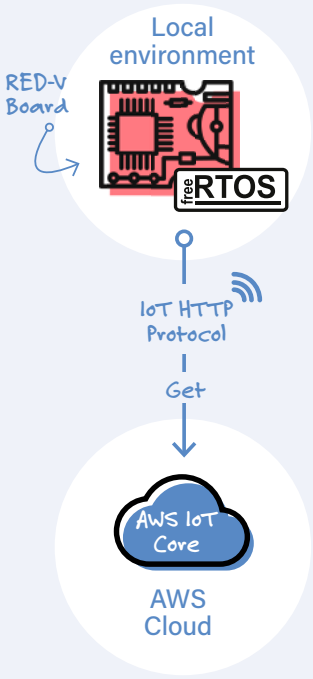

Figure 5. Représentation schématique du processus suivi par notre application IdO-sur-AWS.

Connexion de RED-V comme dispositif IoT au serveur HTTP IoT d'AWS

La configuration est faite, on va s'amuser… et faire une démo HTTP sur la carte RED-V via AWS IoT qui crée une seule tâche d'application. HTTP signifie *HyperText Transfer Protocol*, et HTTPS est crypté avec *Transport Layer Security*, où TLS. HTTP et HTTPS sont des protocoles utilisés pour transférer des données sur le web. Ils utilisent des méthodes de requête spécifiques pour effectuer diverses tâches. Les deux méthodes les plus connues sont GET, qui demande des données à une ressource spécifique, et POST, qui envoie des données à un serveur pour créer ou mettre à jour une ressource. Ce sont les éléments de base de toute application IdO, et cette démo montre qu'on peut tous les déployer via AWS en utilisant du matériel et des logiciels open source.

La démo (**fig. 5**) se connectera au serveur HTTP d'AWS IoT, créera et enverra une requête HTTP, recevra une réponse HTTP, puis se déconnectera du serveur. Grâce à la documentation de FreeRTOS, nous pouvons examiner la structure de l'application à travers le code assez complexe d'AWS [1].

En utilisant la bibliothèque FreeRTOS HTTP Client, le RED-V est maintenant un dispositif IdO capable d'envoyer des requêtes HTTP et de recevoir des réponses HTTP avec la plupart des serveurs HTTP - et pas seulement les serveurs AWS. Par conséquent, il est facilement transposable à d'autres applications dynamiques, car :

- il a des fonctions API à la fois entièrement asynchrones et synchrones ;
- la mémoire est gérée par l'application pour le contexte interne et les en-têtes HTTP ;
- il est *thread-aware* et les connexions sont parallélisées.

Des possibilités infinies

Il est facile de réaliser une application IoT sur l'architecture RISC-V, couplée au noyau et aux bibliothèques FreeRTOS. Elles permettent de personnaliser et dimensionner les dispositifs embarqués pour n'importe quelle application. La configuration de requêtes HTTP via la bibliothèque client est la partie visible de l'iceberg dans la création d'applications IdO. Les possibilités sont infinies pour ces technologies open source !

(200699-02 – VF : Denis Lafourcade)

Espace Elektor en ligne pour cet article
www.elektormagazine.fr/esfe-en-redv

Accessoires

Vous trouverez les accessoires mentionnés dans cet article chez SparkFun et chez Elektor !

- **SparkFun RED-V RedBoard - SiFive RISC-V FE310 SoC**
  www.elektormagazine.fr/esfe-en-redv1

- **SparkFun RED-V Thing Plus - SiFive RISC-V FE310 SoC**
  www.elektormagazine.fr/esfe-en-redv2

LIENS

[1] AWS code: https://www.freertos.org/http/http-demo-with-tls-mutual-authentication.html