Mesures et analyse des consommations énergétiques d'une architecture IoT très faible puissance

Przemyslaw Bakowski, Alexis Bitaillou, Benoît Parrein

Compas'2024 : Parallélisme/ Architecture / Système
IETR/LS2N/UN - Nantes - France, 2-5 juillet 2024

Mesures et analyse des consommations énergétiques d'une architecture IoT très faible puissance

Przemyslaw Bakowski, Alexis Bitaillou, Benoît Parrein

Nantes Université, Polytech Nantes
Laboratoire LS2N - rue Christian Pauc
44306 Nantes cedex 3 - France
benoit.parrein@univ-nantes.fr

Résumé
L'autonomie des objets communicants est un critère essentiel dans le choix des matériels et architectures. Dans ce travail, nous étudions une architecture originale construite autour du *System on Chip* ESP32-C3 dite très faible puissance (ou Very Low Power) qui concerne des consommations électriques en deçà de 100 µA. Les résultats obtenus indiquent d'une part qu'un même programme exécuté en C++ ou en MicroPython possède des consommations électriques équivalentes. D'autre part, nous présentons des valeurs objectives de consommation dans le cadre de transmissions Wi-Fi (IEEE 801.11g) et LoRa.

Mots-clés : Internet des objets, microcontrôleur, ESP32-C3, efficience énergétique

---

1. Introduction

Les contraintes énergétiques sont de plus en plus importantes pour l'Internet des Objets. En réponse, les microcontrôleurs deviennent plus économes en énergie. C'est ainsi que les microcontrôleurs (ou MCU) faible puissance (consommation <1 mA ou *Low Power* ou LP), puis très faible puissance (consommation <100 µA ou *Very Low Power* ou VLP) sont apparus.
Il existe déjà des études pré-existantes sur la mesure de la consommation énergétique de différents composants informatiques [4] ou systèmes des pour l'Internet des objets [1]. De même, il existe des comparatifs sur l'efficience des langages de programmation. Nous pouvons citer, par exemple, les études de Pereira *et al.* [3] et plus récemment, celle de Koedijk et Oprescu [2]. Il ressort de leurs expérimentations que les langages "compilés" permettent une plus grande efficience énergétique que les langages "interprétés". Ce résultat est notamment à la différence du niveau d'abstraction.
Dans cet article, nous proposons de mesurer la consommation électrique d'un kit de développement, assemblé par nos soins, construit autour du MCU ESP32-C3 très faible puissance dans différents scénarios de programmation et de transmission.
L'article est organisé comme suit. Dans la Section 2, nous décrivons l'ensemble du matériel utilisé, ainsi que tous les aspects logiciels (langage, version). Dans la Section 3, nous abordons les scénarios d'évaluation de la consommation de notre kit de développement et les résultats. Nous terminerons enfin par une conclusion dans la Section 4.

2. Matériel et méthode

Dans cette section, nous allons présenter l'ensemble du matériel utilisé et décrire les paramètres dans nos expérimentations. Pour la suite, nous allons déterminer la consommation à partir de l'intensité. En effet, la tension étant paramétrée à partir de notre carte de mesure, la puissance électrique consommée est donc proportionnelle à l'intensité. Par abus de langage, nous parlerons donc d'intensité pour parler de consommation électrique.

2.1. Matériel

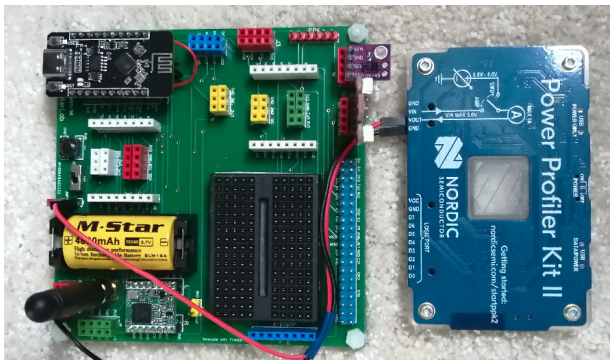

FIGURE 1 – Notre kit de développement (à gauche) et carte de mesure PPKII (à droite).

Dans cette sous-section, nous allons présenter le matériel utilisé. Pour nos expérimentations, nous avons un kit de développement personnalisé et une carte de mesure (Power Profiler Kit II). La Figure 1 est une photographie du kit de développement terminal et de la carte de mesure connectée de type Power Profiler Kit II (PPKII) de Nordic. La Figure 2 schématise l'intégration des différents composants de ce kit de développement.

2.1.1. Kit de développement

Dans cette sous-section, nous allons décrire notre kit de développement. Notre kit de développement se compose principalement d'une carte de développement Arduino HELTEC Automation ESP32-C3 [1] et d'un module de communication externe LoRa Semtech SX1276.

La carte de développement HELTEC ESP32-C3 est équipée d'un SoC (*System-on-Chip*, système sur puce) ESP32-C3FN4. D'après les spécifications techniques du fabricant, la consommation du SoC est de 5 µA en sommeil profond et donc de catégorie VLP. D'après les spécifications de la carte de développement, la consommation en sommeil profond est de 5,5 µA. Donc, la carte ajoute une consommation en sommeil profond de 0,5 µA par rapport au SoC. Il s'agit du minimum de consommation de notre kit (hors extinction). Le SoC ESP32-C3FN4 comporte une connectivité Wi-Fi. Dans nos expérimentations, nous allons utiliser une connexion sans-fil Wi-Fi 802.11g avec sécurité WPA2. D'après les spécifications et dans ces conditions de transmission, le SoC peut consommer jusqu'à 285 mA toujours selon la spécification.

Le module de communication LoRa est de type SX1276 de Semtech. D'après les spécifications, il consomme entre 20 mA et 120 mA en fonction de la puissance d'émission. En veille et en sommeil, sa consommation est réduite, entre 0,2 µA et 1,8 mA.

1. Spécifications de la carte HELTEC disponible à ce lien :`https://resource.heltec.cn/download/ESP32-C3_DevBoard/ESP32-C3_DevBoard(Rev1.1).pdf`

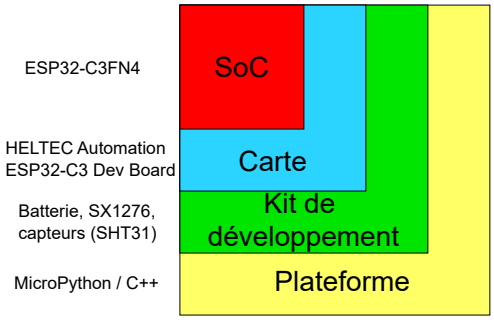

FIGURE 2 – Schéma récapitulatif du niveau d'intégration du matériel.

2.1.2. Carte de mesure

Notre carte de mesure est une carte Power Profiler Kit II (PPKII) de Nordic Semiconductor (nRF-PPK2) [2]. Elle a été conçue pour mesurer la consommation électrique d'une carte ou d'un composant électronique. Elle peut fonctionner en deux modes : source et ampèremètre. Dans le mode ampèremètre, la carte ou le composant mesuré est alimenté par une alimentation externe. En mode source, la carte PPKII alimente la carte ou le composant mesuré à la tension désirée. Dans notre cas, nous allons l'utiliser en mode source afin d'avoir une tension connue et stable. D'après les spécifications, la carte PPKII peut mesurer un courant compris entre 200 nA et 1 A. Ces spécifications sont suffisantes par rapport aux valeurs attendues. La carte PPKII est alimentée par une source d'énergie considérée comme stable (adaptateur secteur).

2.2. Logiciel : choix du langage

Dans cette sous-section, nous allons détailler la partie "logiciel" de nos expérimentations. Le langage de programmation de notre carte dépend du SoC, donc de l'ESP32-C3. Dans cette étude, nous comparons la consommation énergétique entre le langage C++ (avec le cadriciel Arduino) et le langage MicroPython.

2.2.1. C++

Le développement pour les SoC ESP32 s'articule sur le kit de développement logiciel (*Software Development Kit*, SDK) ESP-IDF. Ce SDK est disponible uniquement pour le langage C++ avec le support optionnel du cadriciel Arduino. Ceci permet, entre autre, la portabilité de la plupart des programmes Arduino sur ESP32 sans effort supplémentaire. Pour nos expérimentations, nous avons utilisé le SDK ESP-IDF avec le support du cadriciel Arduino, en version v5.2.1.

2.2.2. MicroPython

MicroPython permet d'avoir un langage de programmation de très haut niveau sur un microcontrôleur. Plus précisément, MicroPython est une implémentation partielle en C de Python. Il permet donc d'exécuter un programme Python sur un microcontrôleur. Cependant, MicroPython ne contient qu'un sous-ensemble des fonctionnalités de Python, pour pouvoir satisfaire les contraintes matérielles. Pour nos expérimentations, nous avons utilisé un binaire pré-compilé officiel pour ESP32-C3 générique en version 1.18 [3].

2. Spécifications de la carte Nordic PPKII disponible à ce lien : `https://infocenter.nordicsemi.com/pdf/PPK2_User_Guide_v1.0.1.pdf`
3. Version de MicroPython disponible à cette adresse : `https://micropython.org/download/ESP32_GENERIC_C3/`

2.3. Réseaux et services testés

Dans notre étude, les échanges en Wi-Fi s'effectuent selon la norme IEEE 802.11g. Afin de les sécuriser, les échanges utilisent WPA 2 CCMP (*Counter-Mode/CBC-Mac protocol*) avec une clé pré-partagée (*Pre-Shared Key*, PSK). L'environnement radio n'est pas proprement normalisé. Il peut être sujet à des interférences radios non contrôlées. La maîtrise de l'environnement radio et la reproductibilité des mesures est l'objet d'une prochaine étude. Le kit de développement est à environ 10 m du point d'accès. Cette distance permet de minimiser les pertes de trames.
Les échanges en LoRa s'effectuent sur la bande ISM 868 MHz. LoRaWAN n'est pas utilisé lors des échanges. Il n'y a donc pas de classe de terminaux associée. Le *spreading factor* (SF) est de 7 avec la modulation native CSS (*Chirp Spread Spectrum*). SF 7 correspond à la vitesse de transmission la plus élevée, donc la durée d'émission la plus courte et la portée la plus faible. Au contraire, SF 12 correspond la transmission la plus lente, donc la plus énergivore. Comme la non-réception des envois n'a pas d'incidence d'un point de vue protocolaire, la distance avec le point d'accès n'a donc pas d'effet ici sur la consommation énergétique de LoRa contrairement à une communication en Wi-Fi.

3. Résultats

Dans cette section, nous allons détailler successivement les scénarios de test et leurs résultats. Nous commencerons par évaluer l'impact du langage de programmation sur la consommation d'énergie. Pour rappel, la consommation électrique est évaluée à partir de l'intensité mesurée, la tension étant supposée constante.

3.1. Consommation énergétique des 2 langages

Dans cette sous-section, nous allons exécuter l'Algorithme 1 implémenté en C++ et en MicroPython pour évaluer l'impact du langage de programmation sur les consommations énergétiques. Dans notre Algorithme 1, le mode sommeil profond est le mode de fonctionnement le plus économe. En contrepartie, certaines fonctionnalités sont désactivées. Par exemple, la mémoire vive n'est plus alimentée électriquement. Des algorithmes plus complexes sont envisagés pour une prochaine étude.

Algorithme 1 : Algorithme pour la comparaison des langages.

répéter
    Allumer la DEL;
    Attendre 1 seconde;
    Éteindre la DEL;
    Attendre 1 seconde;
    Préparer le sommeil profond;
    Attendre 5 secondes;
    Afficher une chaîne de caractères constante sur la sortie série;
    Entrer en sommeil profond pendant 10 secondes;
jusqu'à *fin*;

La Figure 3 illustre l'intensité en fonction du temps pour le programme en C++ sur une fenêtre de 1 minute. L'implémentation C++ de notre algorithme s'exécute en moyenne en 7 s. En de-

FIGURE 3 – Intensité instantanée mesurée lors de l'exécution en C++ de l'Algorithme 1 (la sélection grisée et les mesures correspondantes en bas à droite portent sur une période de sommeil profond).

hors de la phase de sommeil profond, le microcontrôleur consomme en moyenne 16,6 mA (non affichée sur la figure). En sommeil profond, la consommation moyenne est réduite à 13,79 µA (sélection grisée) plaçant ainsi l'architecture testée effectivement dans la catégorie des architectures très faible puissance (VLP).

FIGURE 4 – Intensité instantanée mesurée lors de l'exécution en MicroPython de l'Algorithme 1 (la sélection grisée et les mesures correspondantes en bas à droite portent sur une période de sommeil profond).

La Figure 4 illustre l'intensité en fonction du temps pour le programme écrit cette fois en MicroPython. En dehors de la phase de sommeil profond, le microcontrôleur a une consommation tout à fait comparable à l'exécution en C++ précédente (légèrement plus élevée de 1 à 2 mA). De même, en sommeil profond, la consommation minimum est du même ordre que pour la plateforme précédente (de l'ordre de 12 à 13 µA).
En outre, nous observons un pic d'intensité (d'environ 30 mA) qui précède l'exécution du programme en C++ et MicroPython. En effet, le sommeil profond décharge la mémoire vive du microcontrôleur. Avant d'exécuter les instructions du programme de l'utilisateur, le microcon-

trôleur doit donc recharger l'ensemble de notre programme. Un second pic d'intensité est présent avant la mise en sommeil profond. Son intensité et sa durée sont moindres comparés au premier pic.
La différence de consommation entre le programme C++ et MicroPython est contre-intuitive. A cause de son niveau d'abstraction, Python nécessite plus d'opérations, donc plus d'énergie. Nous nous attendions à la même conclusion avec MicroPython. Ce résultat nécessite d'être confirmer avec d'autres algorithmes plus complexes.

3.2. Coût des transmissions

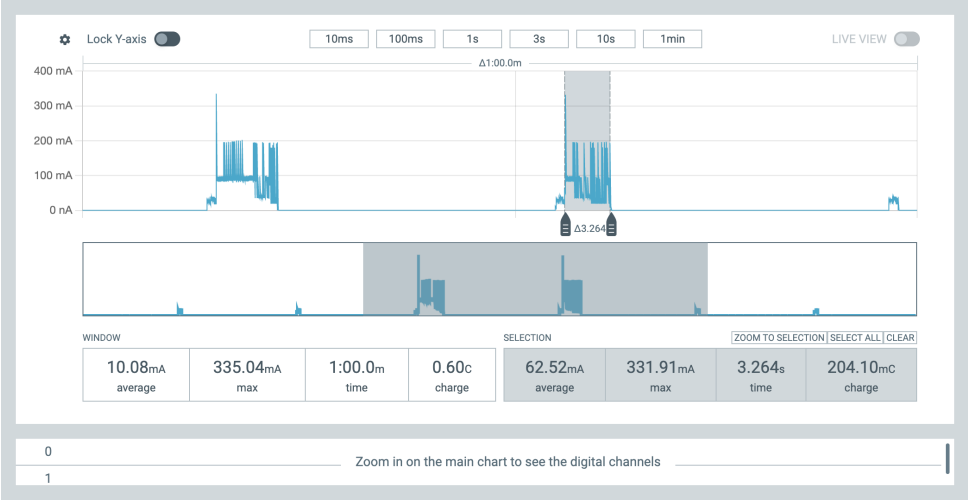

FIGURE 5 – Intensité instantanée mesurée du courant (en mA) en fonction du temps (en s). La sélection grisée et les mesures correspondantes en bas à droite portent sur la phase de transmission en Wi-Fi.

Dans ce second scénario, nous allons évaluer la consommation énergétique de deux technologies sans-fil : Wi-Fi et LoRa. La Figure 5 illustre une séquence de mesure et d'envoi de données via Wi-Fi. La partie transmission est mise en évidence. Dans notre exemple, l'échange dure 3,264 s. La consommation électrique est en moyenne de 62,52 mA, avec un pic à 331,91 mA. En dehors du pic, la consommation est conforme à la spécification (285 mA).
La Figure 6 illustre une séquence de mesure et d'envoi de données en LoRa. Dans notre exemple, l'échange dure 1,228 s. La consommation électrique est en moyenne de 45,28 mA, avec un pic 151,30 mA.
La différence de consommation entre LoRa et Wi-Fi peut s'expliquer par la différence des opérations réalisées. En effet, la connexion Wi-Fi est sécurisée avec WPA 2. Ce protocole nécessite d'effectuer 2 envois et 2 réceptions pour sécuriser la connexion au réseau. Ensuite, la bibliothèque ThingSpeak pour Arduino utilise des connexions HTTP pour l'envoi des données. Hors HTTP3, cela nécessite d'établir une connexion TCP, donc d'effectuer a minima 2 envois et 2 réceptions supplémentaires. Enfin, contrairement à Wi-Fi, LoRa ne fournit aucune garantie de livraison.

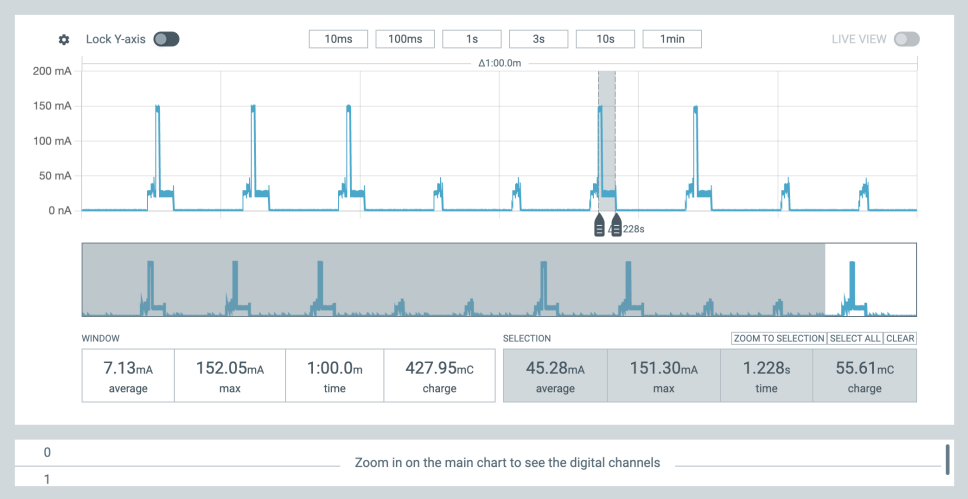

FIGURE 6 – Intensité instantanée mesurée du courant (en mA) en fonction du temps (en s), la période surlignée correspond une mesure de capteur et une phase de transmission en LoRa.

4. Conclusion

Dans cet article, nous avons mesuré la consommation énergétique d'un kit de développement construit autour du SoC ESP32-C3. Nous l'avons évalué sur deux scénarios de programmation et de transmission. Dans le premier, nous avons comparé la consommation électrique d'un même algorithme implémenté en C++ et en MicroPython. Contre-intuitivement, le programme en MicroPython a une consommation énergétique comparable à celui en C++. Ces premiers résultats nécessitent d'être confirmés par des tests extensifs, notamment en réalisant des tâches plus compliquées comme le relevé de capteurs ou l'envoi de données. Dans un second scénario, nous avons comparé le coût de la transmission de données entre les protocoles Wi-Fi et LoRa. Le scénario proposé désavantage Wi-Fi, car d'autres critères comme la qualité de service (taux de transmission, p. ex) ne sont pas évalués. Spécifiquement aux terminaux très faibles puissances, notre kit consomme moins de 15 µA en sommeil profond. En dehors des transmissions sans-fil, l'ensemble consomme en moyenne moins de 50 mA.

Bibliographie

1. Codeluppi (G.), Cilfone (A.), Davoli (L.) et Ferrari (G.). – LoRaFarM : A LoRaWAN-Based Smart Farming Modular IoT Architecture. *Sensors*, vol. 20, n7, janvier 2020, p. 2028. – Number : 7 Publisher : Multidisciplinary Digital Publishing Institute.
2. Koedijk (L.) et Oprescu (A.). – Finding Significant Differences in the Energy Consumption when Comparing Programming Languages and Programs. – In *2022 International Conference on ICT for Sustainability (ICT4S)*, pp. 1–12, juin 2022.
3. Pereira (R.), Couto (M.), Ribeiro (F.), Rua (R.), Cunha (J.), Fernandes (J. P.) et Saraiva (J.). – Ranking programming languages by energy efficiency. *Science of Computer Programming*, vol. 205, mai 2021, p. 102609.
4. Wang (Y.), Nörtershäuser (D.), Le Masson (S.) et Menaud (J.-M.). – Experimental Characterization of Variation in Power Consumption for Processors of Different Generations. – In *2019 International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, pp. 702–710, juillet 2019.