



ÉLECTRONIQUE | EMBARQUÉ | RADIO | IOT

HACKABLE

L'EMBARQUÉ À SA SOURCE

N° 48
MAI / JUIN 2023

FRANCE MÉTRO. : 14,90 €
BELUX : 15,90 € - CH : 23,90 CHF ESP/IT/PORT-CONT : 14,90 €
DOM/S : 14,90 € - TUN : 35,60 TND - MAR : 165 MAD - CAN : 24,99 \$CAD

L 19338 - 48 - F: 14,90 € - RD



CPPAP : K92470

FREERTOS / AVR

Faites fonctionner un vrai **RTOS** sur un modeste **ATmega32U4** avec seulement 2560 octets de RAM ! p.96

DOMOTIQUE / ESPHOME

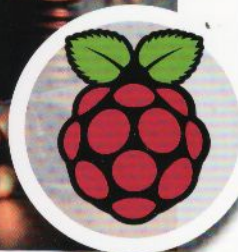
Pilotez et gérez votre installation Home Assistant grâce aux automatisations, scènes et scripts p.18

Agent conversationnel / LLM / IA

EXÉCUTER VOTRE PROPRE CHATBOT TYPE GPT SUR RASPBERRY PI !

p.52

- Comprendre les modèles de langage
- Optimiser les ressources
- Exécuter des modèles localement



GNURADIO / ÉVÉNEMENT

Retour sur les **European GNU Radio Days 2023** : la synchronisation sous toutes ses formes ! p.04

MULTIBOOT / DÉMONISTE

Partagez votre précieuse **Pi4** en installant **FreeBSD**, **OpenBSD** et **NetBSD** sur une même carte SD p.70

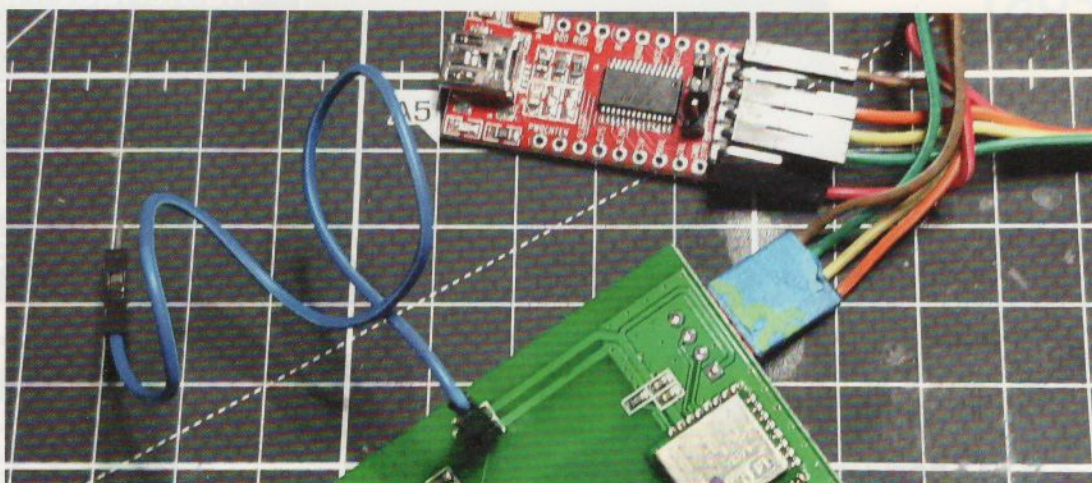
RADIO / RTKLIB / U-BLOX

Utilisation des stations de référence **Centipède/IGN** pour le positionnement centimétrique p.110

HOME ASSISTANT PHASE 3 : INTERRUPTEURS, CONTRÔLE ET AUTOMATISATION

Denis Bodor

Après avoir découvert Home Assistant puis enrichi l'installation avec quelques capteurs supplémentaires dans les deux précédents articles ([1] et [2]), il est temps d'entrer plus en profondeur dans ce qui fait qu'une solution domotique devient réellement intéressante au quotidien : remplacer des actions manuelles de contrôle par des automatisations avec dans l'idée que l'habitat se charge lui-même des tâches récurrentes ou réagisse automatiquement, seul, à l'avènement de situations données ou à des actions de la part de ses occupants.



L'installation telle que vous la connaissez à l'issue du second article était principalement constituée de capteurs de température et de systèmes de contrôle des lumières via des Sonoff Basic, en dehors de deux ou trois points particuliers se résumant à une occurrence unique (mesure du niveau de fioul dans une cuve, détection de mouvement avec capteur PIR ou encore comptage des *tics* détectés par un tube Geiger-Müller). Nous n'avons que très peu évoqué l'automatisation, si ce n'est par quelques exemples sporadiques nécessaires sur le coup. Il est grand temps de nous pencher sérieusement sur le sujet.

Mais pour pouvoir judicieusement et de façon ergonomique profiter des avantages

de l'automatisation, nous devons ajouter une **famille** d'entités à notre petit cheptel. Ce n'est, bien entendu, pas strictement nécessaire puisqu'il est possible d'automatiser tout et n'importe quoi, mais ceci donnera un aspect plus pragmatique au sujet (et puis j'ai reçu le matériel que j'attendais). Pour l'heure, nous captons des « choses » environnementales et agissons sur des entités (les lumières), mais nous n'avons pas d'entrée directe permettant à un humain de signifier ses intentions autrement que via l'interface web (ou mobile) de Home Assistant.

En version courte : nous n'avons ni bouton ni interrupteur permettant de faire réagir notre installation comme un habitat « normal ». Et c'est précisément ce que nous allons régler de ce pas, avant de nous attaquer aux automatisations. Et, en effet, les Sonoff Basic possèdent un bouton, mais on ne peut raisonnablement pas les voir (sauf cas particulier) comme un équivalent à un interrupteur mural ou une interface homme-machine digne de ce nom.

1. INTERRUPTEUR SONOFF TOUCH

Le premier produit à envisager pour ajouter un peu d'interactivité physique dans notre installation domotique est l'interrupteur « intelligent ». Encore une fois, c'est vers Sonoff que mon attention s'est portée et en particulier sur le Sonoff



Ces interrupteurs « intelligents » sont vendus comme compatible Google Assistant et Amazon Alexa. Bien entendu, ils ne le resteront pas longtemps, puisque soucieux de notre vie privée, nous allons les reflasher avec ESPHome (le marquage rouge à droite est de mon fait, l'inscription originale s'effaçant avec le temps).

La façade de l'interrupteur se retire assez facilement et révèle, apparemment, quelque chose de relativement simple. Notez l'emplacement pour connecteur 4 broches, c'est la liaison série. De bas en haut, VCC (carré), RX, TX et GND.



Le circuit de l'appareil est composé de deux PCB avec, en façade les LED, la détection capacitive et l'ESP8285, et dessous, la partie « puissance » avec le circuit d'alimentation et de contrôle du relais.



Touch EU. L'idée est de, tout simplement, remplacer les interrupteurs muraux traditionnels par ces dispositifs capables de fonctionner de façon autonome, comme un interrupteur « bête », mais également pilotable par Home Assistant ou un autre système domotique (comme Google Assistant ou Amazon Alexa si la notion de « vie privée » n'a pas trop d'importance pour vous).

Les Sonoff Touch EU se trouvent facilement sur des sites comme AliExpress ou autres, pour des prix allant de 10 à 15 euros. Initialement prévus pour fonctionner avec des solutions propriétaires ou *cloud* (Google, Amazon, etc.), ils pourront être facilement flashés avec un *firmware* ESPHome. Comme le Sonoff Basic, leur architecture est relativement simple et repose également sur un ESP8285, couplé à quelques éléments passifs, un contrôleur pour l'aspect tactile capacitif et un relais.

Comme le montrent les illustrations ci-contre, le démontage du produit est aisé. On retire simplement la face avant et on met à nu la circuiterie interne qui se compose de deux PCB, un pour la partie « intelligente » et l'autre pour le contrôle de la tension domestique (relais) et l'alimentation. Le circuit le plus en avant du boîtier, qui comprend l'ESP8285, dispose d'un emplacement permettant de souder un connecteur donnant accès à l'interface série du microcontrôleur. On retrouve les broches classiques, masse, alimentation, RX et TX, sur lesquelles on connectera un convertisseur USB/série (3,3 volts !). La seule difficulté tient dans le passage de l'ESP8285 en mode programmation, puisqu'il faut mettre GPIO0 à la masse lors de la mise sous tension (voir illustration en page 23). Ceci demande un peu de dextérité, mais

reste faisable sans trop d'effort. Notez bien qu'il est **parfaitement hors de question** de tenter une reprogrammation ou même ne serait-ce que d'ouvrir le produit s'il est alimenté par le courant domestique ! Alimentez-le via le VCC fourni par l'adaptateur USB/série, votre domotique ne vous servira à rien si vous finissez six pieds sous terre après avoir posé les doigts là où il ne fallait pas !

Côté ESPHome et script YAML, nous avons quelque chose d'assez similaire à ce que nous avons utilisé pour les Sonoff Basic :

```
substitutions:
  devicename: sonofftouch

esphome:
  name: $devicename

esp8266:
  board: sonoff_basic

logger:

<<: !include inc/wifi.yaml
<<: !include inc/auth.yaml

binary_sensor:
  - platform: gpio
    pin:
      number: GPIO0
      mode:
        input: true
        pullup: true
        inverted: true
    name: $devicename Bouton
    on_press:
      - switch.toggle: relay

switch:
  - platform: gpio
    name: $devicename Relais
    pin: GPIO12
    id: relay

status_led:
  pin:
    number: GPIO13
    inverted: yes
```

```
sensor:
  - platform: wifi_signal
    name: $devicename Wifi Signal
    update_interval: 60s
```

Pour rappel, les fichiers **wifi.yaml** et **auth.yaml** inclus par ce script contiennent respectivement les éléments de configuration Wi-Fi et les identifiants de connexion, à la fois pour le point d'accès, mais aussi pour l'API Home Assistant. Ainsi, **inc/wifi.yaml** contient les paramètres Wi-Fi :

```
wifi:
  ssid: !secret wifi_ssid
  password: !secret wifi_password

# fallback hotspot
ap:
  ssid: $devicename Fallback Hotspot
  password: !secret ap_password
```

Et **inc/auth.yaml**, ceux de l'authentification :

```
api:
  encryption:
    key: !secret api_key

ota:
  password: !secret ota_password
```

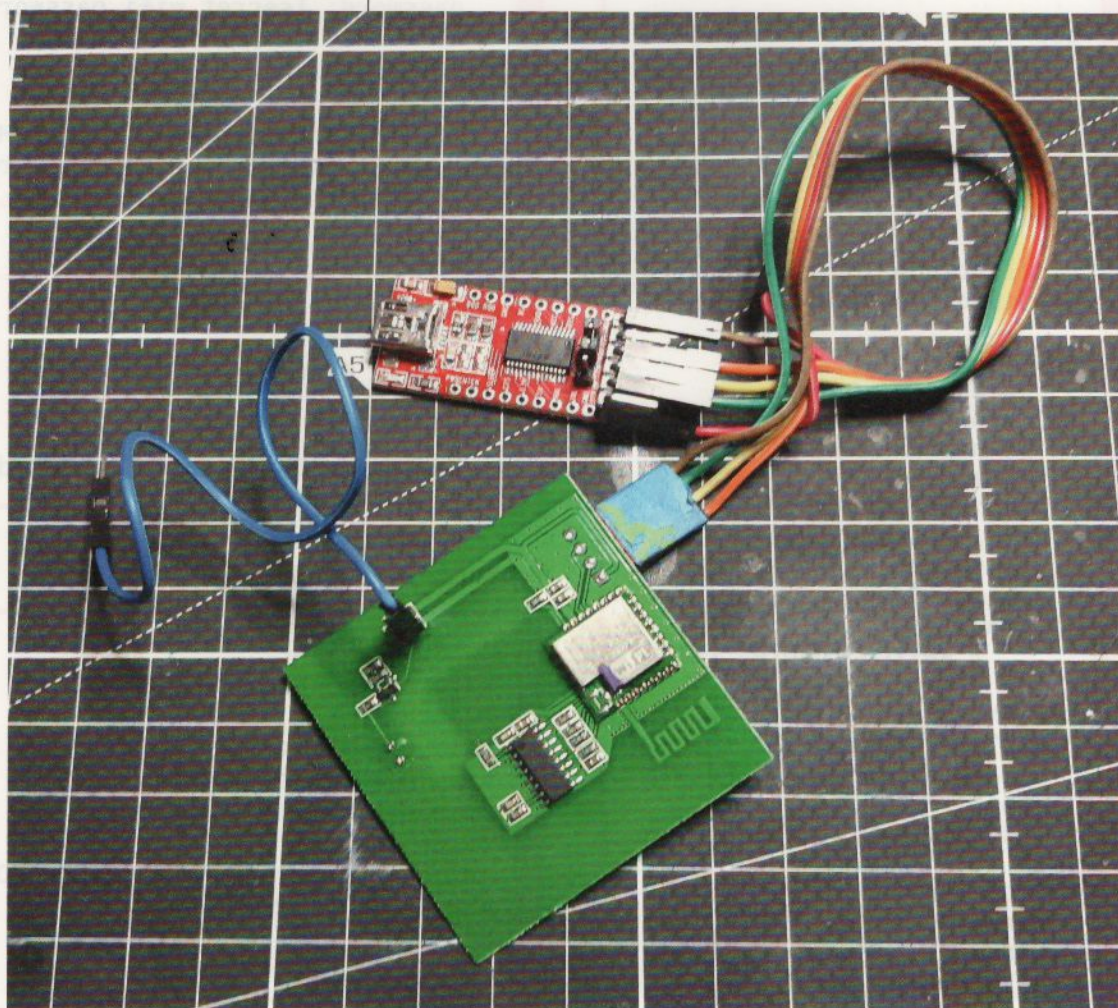
À quoi s'ajoute le fichier **inc/secrets.yaml**, regroupant les informations sensibles :

```
# SSID de l'AP
wifi_ssid: "monSSID"
# auth AP
wifi_password: "phrade2passe"
# auth AP de secours
ap_password: "mot2passeAP"
# clé API HA
api_key: "ODcyOWE0MWJjYmEzNDRjNDgxNzQ5NzMwMGIXNTEyOTI"
# mot de passe MàJ OTA
ota_password: "123456"
```


La programmation d'un Sonoff Touch EU nécessite de jongler un peu afin de mettre GPIO0 à la masse au moment de la mise sous tension via les broches VCC et GND. On récupérera une masse très facilement sur le connecteur inter-PCB dont les broches sont clairement identifiables grâce à la sérigraphie. GPIO0 est directement accessible sur le module ESP8285 et la connexion n'a pas besoin d'être maintenue après démarrage en mode programmation.

Dans le script lui-même, nous configurons simplement GPIO0 comme entrée (**binary_sensor**), puisque cela correspond à la zone capacitive en façade, et GPIO12 en sortie pour le contrôle du relais (**relay**). GPIO13 est connecté à la LED intégrée et est, très classiquement, configuré en témoin d'état (**status_led**). Une entité supplémentaire est également ajoutée (**wifi_signal**) permettant de garder une trace de la puissance du signal Wi-Fi.

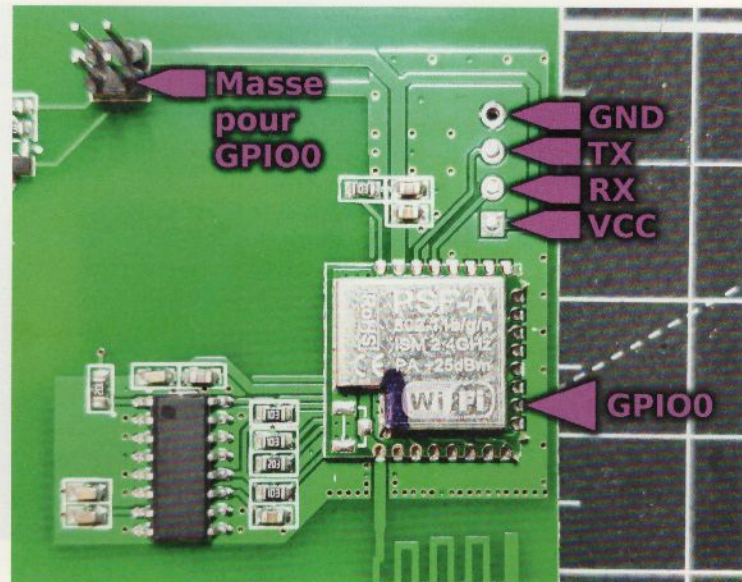
Comme pour les périphériques précédents, nous utiliserons le conteneur Docker (après un **docker pull esphome/esphome** si vous n'avez pas encore installé ESPHome) pour compiler et flasher le firmware avec : **docker run --network=host --rm -v "\${PWD}":/config --device=/dev/ttyUSB1 -it esphome/esphome run sonofftouch.yaml**. Avec **/dev/ttyUSB1** qui correspond au convertisseur USB/série tel que détecté par le système.



Pour déployer un grand nombre de ces interrupteurs, nous pourrions ajouter le paramètre `-s devicename` suivi d'un nom qui viendra remplacer `sonofftouch`, et donc flasher les uns après les autres tous les périphériques, avant de les installer.

Ce qui nous amène au principal problème de ce matériel : comme le montrent les illustrations/photos, la connectique à l'arrière des Sonoff Touch est constituée de trois points de connexion, un neutre, une phase en entrée et une phase en sortie. Cette dernière est celle contrôlée par le relais, alors que le périphérique lui-même est alimenté via la phase en entrée et le neutre. Or, dans nos habitations, il est excessivement rare d'avoir un neutre disponible au niveau de l'interrupteur mural. La configuration courante est une phase partant du tableau électrique, passant par l'interrupteur avant d'aller au luminaire et le neutre revient au tableau. Pour alimenter les Sonoff Touch, nous avons besoin d'un neutre supplémentaire et ceci implique de tirer un autre câble depuis le tableau ou, plus raisonnablement, depuis le boîtier de dérivation le plus proche.

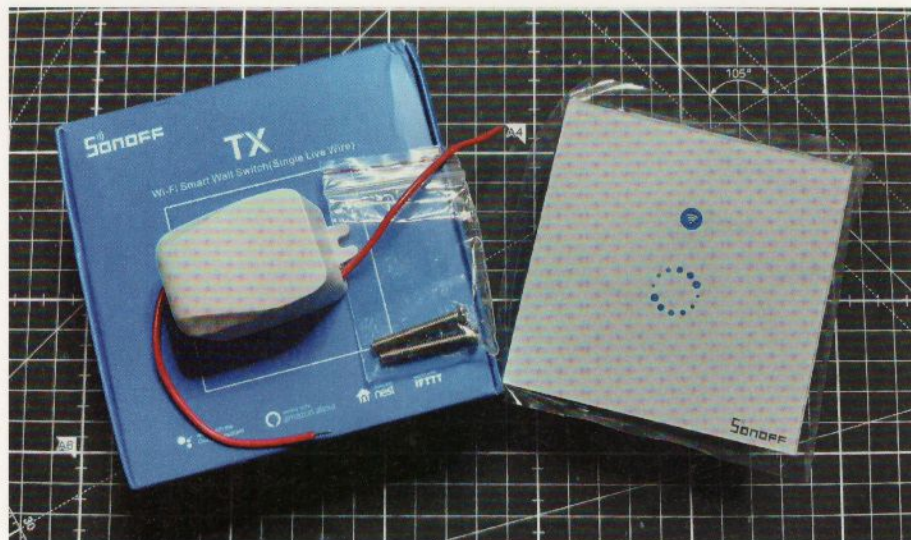
Ceci est non seulement pénible puisqu'en fonction de l'âge de l'installation, les boîtiers de dérivation ne sont pas toujours facilement accessibles, mais il est également



très probable que les couleurs des câbles déjà en place ne soient pas du tout standardisées. Ma maison date de 1947, et je peux vous dire que le classique et normalisé bleu=neutre, brun=phase et jaune/vert=terre, est un doux fantasme en ces lieux. J'ai du noir, du rouge, du blanc, de l'orange, du rose, du vert... et ce sans aucune logique, cohérence ou homogénéité d'une pièce à l'autre. Et je ne parle même pas de la surpopulation des boîtes de dérivation qui est un tout autre problème lorsqu'on souhaite tirer des câbles supplémentaires...

Fort heureusement, je ne suis pas le seul dans cette situation. En fait, la quasi-totalité de l'Europe est ainsi, sans neutre qui passe dans les interrupteurs muraux. Et, bien entendu, ceci constitue donc une magnifique opportunité commerciale pour un fabricant comme Sonoff qui profite de l'occasion pour adresser le problème avec un nouveau produit plus adapté. Et ce produit est un autre Sonoff Touch : le T4EU1C.

Voici un résumé en image de la façon de connecter l'interrupteur à un convertisseur USB/série pour la programmation de la flash de l'ESP8285. Le passage en mode programmation se fait en connectant GPIO0 à la masse, qu'on prendra sur le connecteur supérieur via un câble Dupont mâle/femelle, lors de la mise en alimentation.



2. T4EU1C : INTERRUPTEUR SONOFF SANS NEUTRE

Ce problème de présence d'un neutre peut être contourné assez facilement, mais nécessite de faire quelques concessions. Le principe de fonctionnement d'un interrupteur T4EU1C (disponible dans la même gamme de prix que les Sonoff Touch) consiste à

Le Sonoff T4EU1C est visuellement très similaire au Touch EU, mais est accompagné d'un condensateur dans un petit boîtier à installer en parallèle aux broches d'alimentation du luminaire.

laisse passer une partie du courant dans le circuit, de façon intermittente, afin d'alimenter le microcontrôleur. Pour compenser les effets de cette alimentation découpée, il est alors nécessaire de placer un condensateur aux bornes du luminaire, et celui-ci est bien entendu livré avec le produit (dans un petit boîtier en plastique). Le résultat est donc un interrupteur qui viendra se poser directement en lieu et place de celui, mécanique, existant dans l'installation, sans aucun autre changement.

La principale et importante différence avec un Touch EU est le brochage du connecteur. Ici, on remplace simplement un interrupteur classique (sans neutre) par ce produit, et le tour est joué.

Le T4EU1C ne possède donc que deux connecteurs à l'arrière, *Lin* et *Lout* (phase en entrée et phase en sortie) et le reste de la construction du périphérique est assez similaire aux Sonoff Touch. Nous retrouvons une division en deux circuits imprimés, séparant les hautes tensions de la partie « logique », ainsi qu'un emplacement pour un connecteur à 4 broches permettant la reprogrammation de l'ESP8285. Dans ce produit, il ne s'agit plus d'ailleurs d'un module ESP, mais d'une intégration plus poussée avec le microcontrôleur directement soudé sur le PCB, avec toute la circuiterie annexe (antenne, alimentation, gestion du bouton tactile, etc.).

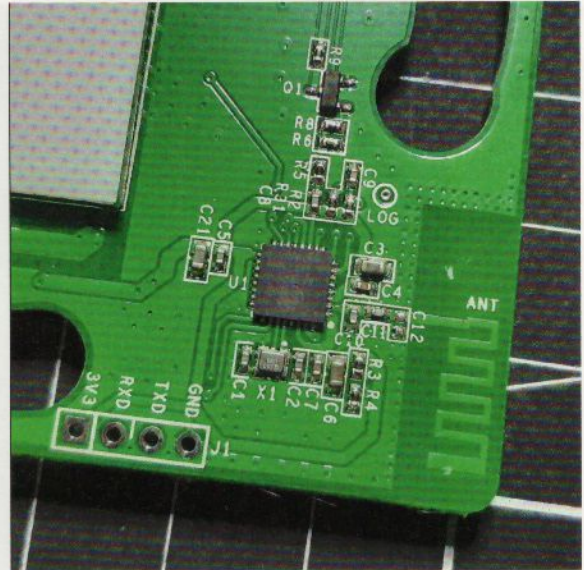
Ceci soulève un autre problème puisque les broches du microcontrôleur deviennent difficilement accessibles et que le passage à la masse de GPIO0, pour entrer en mode programmation, demande un peu de bricolage. Il sera en effet nécessaire de retirer une partie du vernis protégeant



une piste pour forcer le mode à la mise sous tension. Plusieurs endroits peuvent être accédés de la sorte, mais la façon la plus aisée est de poncer légèrement la piste partant du microcontrôleur, près de R31, en se dirigeant vers un via non loin de la zone tactile (voir photo). Une fois le cuivre exposé, inutile de souder quoi que ce soit. Il suffira de faire contact à l'aide d'un simple fil au moment de la mise sous tension via les GND masse et 3V3. La connexion n'a pas besoin d'être maintenue après le *boot*, mais on se retrouve dans une situation équivalente à celle rencontrée dans le premier article [1], avec les ESP01 : ESPHome n'est pas en mesure de procéder à un *reset* après la programmation de la flash. Ce n'est pas gravissime et vous pourrez parfaitement passer outre sans le moindre problème et simplement déconnecter l'adaptateur USB et le reconnecter. Après cette étape, et comme pour les autres montages/produits reflashés avec ESPHome, les programmations ultérieures pourront se faire via le Wi-Fi (OTA) et donc sans liaison série et mise à la masse de GPIO0.

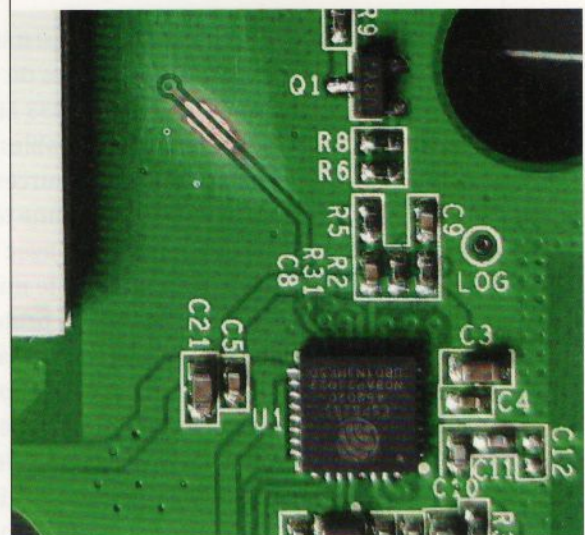
À propos de programmation justement, c'est exactement le même script YAML qu'avec les Sonoff Touch qui pourra être utilisé, et ce, malgré l'électronique légèrement différente entre les deux produits. Les GPIO sont les mêmes et utilisés de la même manière pour le même usage. À vous ensuite d'estimer pertinent ou non de différencier les deux dans leurs noms, lors de leur déploiement et de leur intégration dans Home Assistant.

Le choix de l'un ou l'autre modèle d'interrupteur se fera en fonction de la faisabilité de l'ajout d'un neutre à une installation. En effet, il existe une différence « d'utilisation » entre les deux : la zone capacitive du T4EUC1 est moins sensible que celle d'un Sonoff Touch. C'est un peu déroutant au premier abord, mais c'est quelque chose à quoi on s'habitue à l'usage (ou pas). Des mains un peu sèches ou une hygrométrie variable d'un jour à l'autre peuvent influencer sur la capacité de l'interrupteur à détecter un contact trop bref ou pas assez appuyé, alors que ce n'est absolument pas le cas du modèle « avec neutre », qui fonctionne

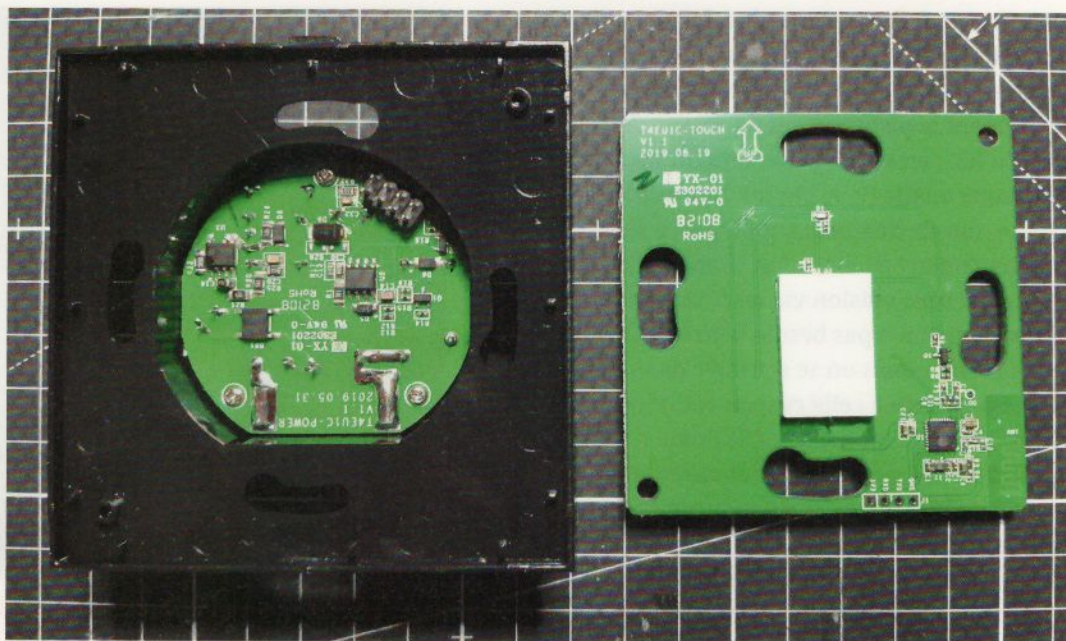


Sur ce modèle, le port série dispose d'une sérigraphie, mais le brochage est strictement identique au Sono Touch EU. Remarquez que l'ESP8285 n'est plus un module soudé, mais fait partie intégrante du circuit.

On pourra récupérer une connexion à GPIO0, pour le passage en mode programmation, en grattant ou ponçant le vernis d'une piste près du transistor Q1. Cet emplacement devra être mis à la masse au moment d'alimenter l'ESP8285.



Comme avec le Sonoff Touch EU, la conception est en deux parties avec le circuit de puissance (alimentation et contrôle du relais) placé au fond du boîtier.



même lorsqu'on porte des gants de travaux. Ainsi, j'ai donc préféré les Sonoff Touch aux Sonoff T4EUC1 pour les installations dans ma cave, qui sert également d'atelier de bricolage. L'ajout de neutre était aisé du fait de l'aspect « brut » du lieu et de la sensibilité importante.

3. TÉLÉCOMMANDES ET BOUTONS

Avant de mettre en place Home Assistant et une tripotée de Sonoff pour piloter mes luminaires du salon, j'utilisais de simples prises télécommandées sur 433 MHz. Ce genre de produits étant relativement configurables, je pouvais donc activer/désactiver un ensemble de sources d'éclairage avec une simple pression sur le bouton d'une télécommande (une solution poussée ensuite à l'extrême avec un pilotage de la télécommande par un Arduino dans le numéro 19 [3] puis un ESP8266 dans le Hackable 24 [4]). Le problème, cependant, était l'extinction de l'éclairage standard (au plafond) puisque les interrupteurs muraux, eux, n'étaient pas télécommandables. Le passage aux Sonoff T4EUC1 précédemment détaillés a corrigé ce problème. Je peux donc, depuis mon bureau et via l'interface de HA, basculer de l'éclairage standard à celui « d'ambiance », pour une petite séance de détente en

compagnie de Walter, Olivia et Peter (Fringe), ou tantôt Rick et Morty.

Mais les habitudes ont la vie dure et il m'arrive encore, plus d'une fois par semaine, de chercher la télécommande par réflexe, ou tout simplement d'éteindre le PC pour me rendre compte, ensuite, qu'il faut que je sorte mon smartphone pour éteindre l'éclairage d'ambiance avant de quitter la pièce direction mon lit douillet. Ça ne va pas, c'est au système de s'adapter et non à moi (du moins pour ça).

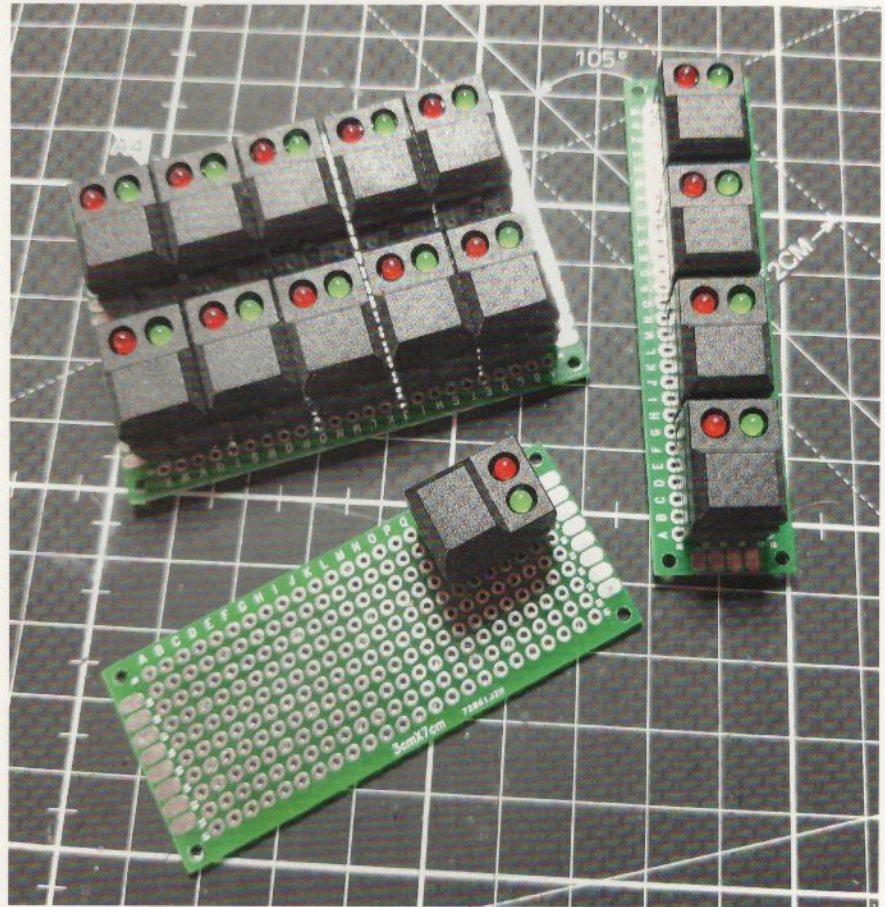
La solution à ce problème crucial est donc de disposer d'une « télécommande » me permettant, peu ou prou,

de faire la même chose que précédemment, mais avec Home Assistant et en Wi-Fi. Une interface web ou même une application sur smartphone est trop « lourde » à mon sens, en comparaison d'un petit dispositif qui traîne sur le bureau, n'attendant que quelques pressions sur ses boutons (je ne suis pas du genre à vivre en symbiose avec smartphone, loin de là). En y réfléchissant, ceci n'est pas très compliqué, à priori, puisqu'une carte ESP8266/ESP32 et un ou plusieurs boutons-poussoirs feront parfaitement l'affaire. Ceci étant, si montage il y a, autant pousser plus loin le concept et disposer, en prime, de quelques LED permettant de visualiser l'état des entités en temps réel. En rendant la réalisation mobile (accu Li-ion/LiPo), ceci pourrait même servir d'outil de *monitoring*, puisque les LED changeront d'état en fonction de celui des luminaires.

3.1 Réalisation sur base ESP32

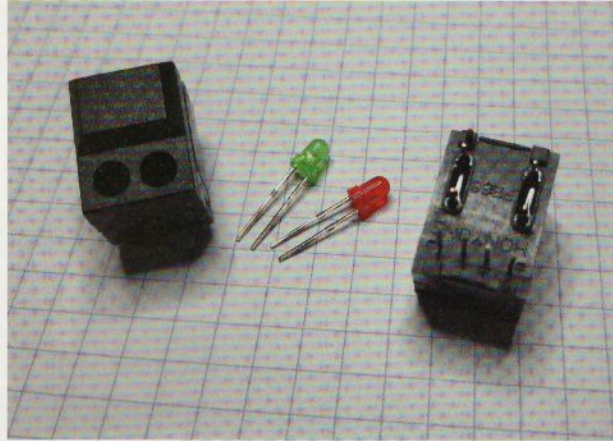
La carte utilisée ici sera une Wemos LOLIN32 v1.0.0 (que je suppose fortement être un clone). Elle repose sur un ESP-WROOM-32 et a été choisie pour plusieurs raisons :

- son format s'adapte parfaitement à la taille d'une des plaques pastillées que j'avais en stock, qui elle-même a la taille parfaite pour accueillir 5 boutons poussoirs ;
- elle dispose de suffisamment de GPIO accessibles et utilisables pour piloter les boutons et les LED (contrairement à un Wemos D1 mini), sans avoir recours à des composants ou des IC supplémentaires ;
- et enfin, elle possède un connecteur JST et un circuit de gestion de charge pour le branchement d'un accu LiPo monocellule.



Ces boutons-pression (tactile momentané), vendus en lot de 10 à environ 10 €, existent en plusieurs couleurs (noir, blanc et rouge) et leur gros avantage est un footprint parfaitement compatible avec une plaque pastillée au pas de 2,54 mm. De plus, ils s'agencent parfaitement les uns par rapport aux autres de façon très compacte.

Autre avantage de ces « composants » : ils utilisent des LED 3 mm parfaitement standard et sont très faciles à démonter entièrement. On peut donc non seulement remplacer les LED en cas de problème, mais aussi personnaliser totalement l'ensemble.



L'approche utilisée est entièrement centrée sur les boutons achetés il y a quelque temps déjà sur AliExpress [5]. Il s'agit de boutons-poussoirs relativement standard, mais étoffés de deux LED de 3 mm, une rouge et une verte. L'ensemble des connecteurs est parfaitement compatible avec un espacement au pas de 2,54 mm permettant d'insérer, sans le moindre problème, ces boutons sur une plaque perforée/pastillée double face. Mieux encore, les LED sont non seulement standard, mais simplement glissées dans le support, ce qui signifie qu'il est parfaitement possible, au besoin, de les remplacer sans bricolage horrible puisque rien n'est collé ou serti. L'ensemble tient pourtant ensuite solidement, après soudure.

La problématique ici est d'arriver à un montage relativement compact, tout en ne cherchant pas forcément à passer énormément de temps en conception. Les GPIO de l'ESP32 semblent nombreux, mais ne sont pas tous utilisables pour toutes les fonctions avec les caractéristiques adéquates. Je suis donc allé au plus simple en limitant les besoins de configuration et en reliant les LED directement à la tension d'alimentation (+3,3 V) via des résistances CMS de 120 ohms (vert) et 150 ohms (rouge). Les sorties seront donc inversées et les entrées auront des résistances de *pull-up* de 10 Kohms (physiques donc, et non configurées logiciellement). En termes de brochage, nous avons :

- bouton 1 = GPIO16, LED verte = GPIO22, LED rouge = GPIO21 ;

- bouton 2 = GPIO35, LED verte = GPIO26, LED rouge = GPIO25 ;
- bouton 3 = GPIO36, LED verte = GPIO14, LED rouge = GPIO27 ;
- bouton 4 = GPIO39, LED verte = GPIO15, LED rouge = GPIO2 ;
- bouton 5 = GPIO17, LED verte = GPIO32, LED rouge = GPIO33.

Je sais que l'ensemble, et en particulier les résistances de *pull-up* ne vont pas dans le sens d'une conception idéale pour un fonctionnement sur accu, mais l'idée était d'avoir une réalisation rapide et une utilisation ne nécessitant pas nécessairement une autonomie importante. La raison d'être de l'accu n'est pas de fournir une complète autonomie sur une longue durée, mais juste une certaine mobilité et souplesse d'utilisation. Il pourra être intéressant de réviser le montage pour obtenir quelque chose de plus adapté en passant, de plus, par la conception d'un circuit imprimé spécifique et les modes d'économie d'énergie fournis par le SDK Espressif et utilisable au travers d'ESPHome [6].

Quoi qu'il en soit, c'est suffisant pour une expérimentation et nous pouvons nous pencher sans plus attendre sur le fichier YAML destiné à ESPHome :


```
substitutions:
  devicename: switchboard01

esphome:
  name: $devicename

esp32:
  board: lolin32

logger:

<<: !include inc/wifi.yaml
<<: !include inc/auth.yaml

binary_sensor:
  - platform: gpio
    pin:
      number: GPIO16
      inverted: true
    name: swbutton1
    filters:
      - delayed_on: 75ms
  - platform: gpio
    pin:
      number: GPIO35
      inverted: true
    name: swbutton2
    filters:
      - delayed_on: 75ms
  - platform: gpio
    pin:
      number: GPIO36
      inverted: true
    name: swbutton3
    filters:
      - delayed_on: 75ms
  - platform: gpio
    pin:
      number: GPIO39
      inverted: true
    name: swbutton4
    filters:
      - delayed_on: 75ms
  - platform: gpio
    pin:
      number: GPIO17
      inverted: true
    name: swbutton5
    filters:
      - delayed_on: 75ms
```

```
light:
  - platform: binary
    name: "SW_greenled1"
    output: sw1_greenled
  - platform: binary
    name: "SW_redled1"
    output: sw1_redled
  - platform: binary
    name: "SW_greenled2"
    output: sw2_greenled
  - platform: binary
    name: "SW_redled2"
    output: sw2_redled
  - platform: binary
    name: "SW_greenled3"
    output: sw3_greenled
  - platform: binary
    name: "SW_redled3"
    output: sw3_redled
  - platform: binary
    name: "SW_greenled4"
    output: sw4_greenled
  - platform: binary
    name: "SW_redled4"
    output: sw4_redled
  - platform: binary
    name: "SW_greenled5"
    output: sw5_greenled
  - platform: binary
    name: "SW_redled5"
    output: sw5_redled

output:
  - id: sw1_greenled
    platform: gpio
    pin: GPIO22
    inverted: true
  - id: sw1_redled
    platform: gpio
    pin: GPIO21
    inverted: true
  - id: sw2_greenled
    platform: gpio
    pin: GPIO26
    inverted: true
  - id: sw2_redled
    platform: gpio
    pin: GPIO25
    inverted: true
```



```

- id: sw3_greenled
  platform: gpio
  pin: GPIO14
  inverted: true
- id: sw3_redled
  platform: gpio
  pin: GPIO27
  inverted: true
- id: sw4_greenled
  platform: gpio
  pin: GPIO15
  inverted: true
- id: sw4_redled
  platform: gpio
  pin: GPIO2
  inverted: true
- id: sw5_greenled
  platform: gpio
  pin: GPIO32
  inverted: true
- id: sw5_redled
  platform: gpio
  pin: GPIO33
  inverted: true

```

Je ne vais pas trop entrer dans le détail ici étant donné que nous avons vu tout cela point par point dans un précédent numéro et qu'il n'y a, dans ce fichier (`switchboard.yaml`), rien de bien nouveau. Comme précédemment, nous utilisons Docker pour provoquer la construction du *firmware* avec l'argument `run` et enchaîner sur la programmation de la flash de l'ESP32 :

```

$ docker run --network=host --rm \
-v "${PWD}":/config --device=/dev/ttyUSB1 \
-it esphome/esphome run switchboard.yaml

```

Suite à quoi un nouvel appareil, `switchboard01`, sera détecté par Home Assistant, nous donnant accès à quelques 15 entités (5 boutons physiques, plus 5 fois 2 LED). Il ne nous reste plus maintenant qu'à faire quelque chose avec ces entités et surtout, réagir lorsqu'un bouton est pressé. Il est vraiment temps de parler d'automatisation.

3.2 D'abord mettre de l'ordre dans les fichiers d'automatisation

En utilisant l'interface web Lovelace, tous les scripts d'automatisation se voient affectés un identifiant (`id`) et se retrouvent tous stockés dans `~/config/automations.yaml`. Mais en gagnant en expérience, vous en viendrez certainement, comme moi, à tout simplement écrire toutes vos automatisations directement en YAML sans passer par l'aide de l'interface graphique.

Dans ce cas, il est sans doute plus intéressant de directement écrire votre YAML avec un éditeur de texte, que ce soit directement via une connexion SSH sur la Pi (Vim est présent, et Git aussi) ou localement sur une autre machine avant de copier cela sur la cible (avec `scp`). Dans un cas comme dans l'autre, tout avoir dans un seul fichier peut rapidement devenir peu plaisant à gérer, même si l'interface web divise le contenu en plusieurs éléments.

La solution consiste donc à répartir les automatisations en plusieurs fichiers YAML qu'on placera dans un répertoire spécifique, ici `~/config/myautomations` qu'on aura créé pour l'occasion. Bien entendu, la configuration par défaut ne prend pas cela en compte et nous devons donc, en premier lieu, éditer le fichier `~/config/configuration.yaml` qui initialement référence les automatisations avec la ligne :


```
automation: !include automations.yaml
```

que nous changeons en :

```
automation: !include_dir_list myautomations/
```

Cette déclinaison de YAML autorise, en effet, des inclusions (même récur-sives) de fichiers via **!include_dir_list** qui retourne une liste où chaque fichier est une entrée (voir [7]). Comme nous ignorons ici le fichier **automations.yaml**, l'emplacement par défaut pour ce qui est créé dans Lovelace, nous ne pourrons plus éditer via l'interface web. Mais les automatisations créées manuellement apparaîtront tout de même dans **Paramètres** et **Automatisations et scènes** et pourront être activées, désac-tivées, exécutées, etc. Notez qu'il est théoriquement possible de supporter une configuration par l'interface et des fichiers distincts, mais je n'en vois pas l'intérêt.

Vous pouvez profiter de cette édition pour créer un répertoire **myscripts** et donc procéder de même pour les futurs scripts, en changeant le :

```
script: !include scripts.yaml
```

en :

```
script: !include_dir_list myscripts/
```

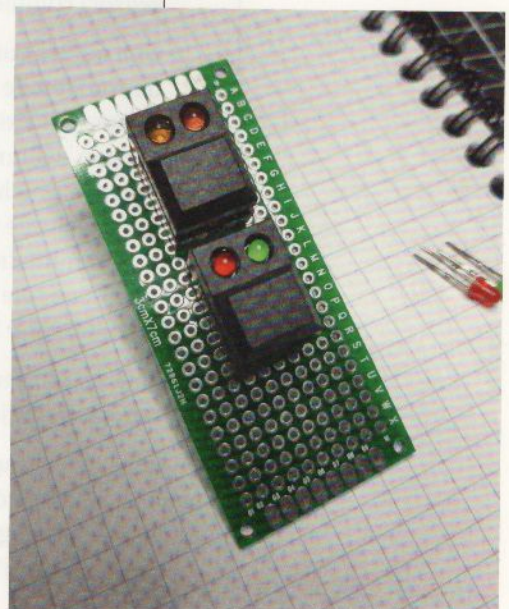
Comme vous venez de modifier la configuration de base, il sera nécessaire de relancer Home Assistant, mais des change-ments dans les automatisations, par la suite, ne rendront pas cela nécessaire. Une petite visite dans **Paramètres** et **Outils de développement**, puis dans la liste **Rechargement de la configu-ration**, un clic sur **Automatisations** et/ou **Scripts** fera l'affaire.

Nous n'avons plus maintenant qu'à créer **~/config/myautomations** si ce n'est déjà fait et à ajouter autant de **.yaml** que nécessaires pour tout automatiser.

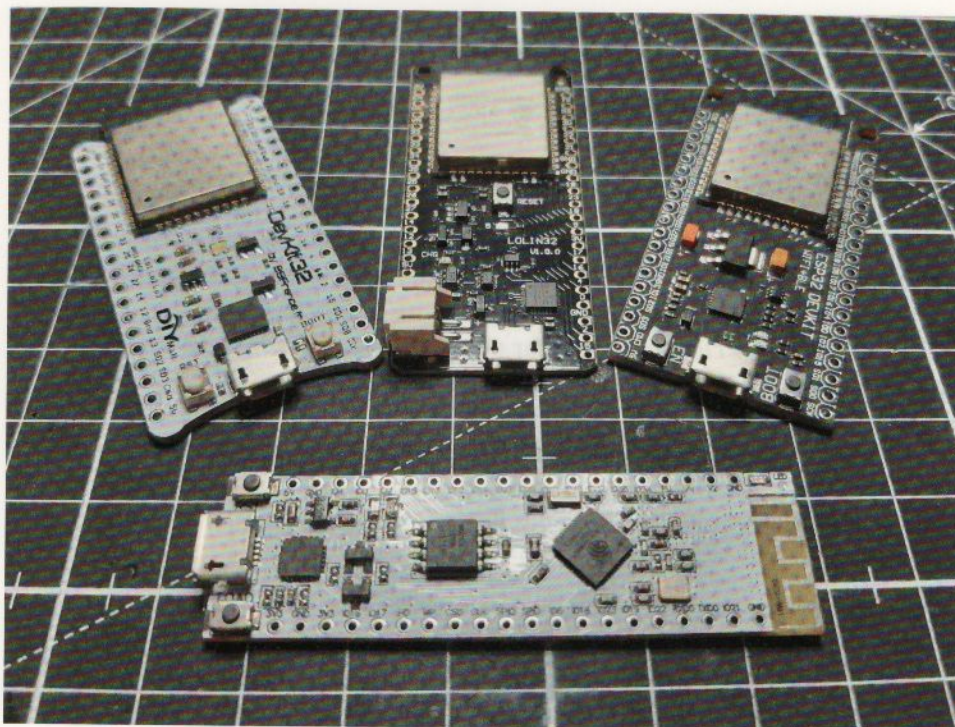
3.3 Automatisons notre télécommande

Grâce à notre télécommande, nous avons à disposition l'équi-valent de 5 interrupteurs Sonoff. L'intérêt de contrôler ainsi des luminaires dans une pièce où nous ne nous trouvons pas étant assez réduit, et le fait d'avoir plus de 5 luminaires d'ambiance dans une pièce assez peu probable, nous pouvons envisager toutes sortes d'usage pour certains de ces boutons. Mais avant cela, voyons déjà à quoi ressemble une utilisation « normale ».

Voici un exemple de remplacement des LED (en haut). Aucun bricolage lourd n'a été nécessaire, les LED existantes ont simplement été glissées hors de l'emplacement et échangées avec d'autres que j'avais en stock.



Les cartes de développement ESP32 sont généralement plus étoffées en GPIO que les ESP8266, en plus de disposer de plus de mémoire et de puissance CPU. Ce sont toutes de bonnes alternatives aux Wemos D1 mini et aux ESP8266 en général.



Une automatisation Home Assistant est un scénario composé de trois éléments, dont un optionnel :

- Un **déclencheur** : c'est événement à l'origine de l'appel de l'automatisation. Dans notre cas, ceci peut être un changement d'état du bouton (un *binary sensor*) comme une bascule, un passage à **on** ou un passage à **off**.
- Une ou des **conditions** qui, sur la base du déclencheur, permettront de décider ou non de mener une action. Ici, une condition n'a pas vraiment de sens à moins, par exemple, de décider que le bouton passant à **on** ne peut déclencher l'action d'allumer la lumière que dans certaines plages horaires ou encore, après le coucher du soleil. Il est généralement plus raisonnable d'envisager l'usage de conditions dans des contextes où le déclencheur n'est pas une action humaine (mais c'est juste mon point de vue). Notez qu'il est également possible de combiner plusieurs conditions avec des opérateurs logiques comme *OU* et *ET*.
- Une ou plusieurs **actions** exécutées lors de l'avènement d'un déclencheur qui seront conditionnées par une condition *VRAI* (si elle est présente).

Armé de ces merveilleuses connaissances, nous pouvons rédiger nos automatisations, chacune dans un fichier placé dans `~/config/myautomations`. Rappelons que notre montage `switchboard01` met à disposition les entités `swbutton1` à `swbutton5` (oui, j'aurais dû commencer à `0`, ça aurait été logique), les LED vertes `sw_greenled1` à `sw_greenled5` et les LED rouges de `sw_redled1` à `sw_redled5`.

Nous voulons deux choses :

- qu'une pression sur l'un des boutons provoque un basculement d'état (**on** à **off**, ou **off** à **on**) d'un des luminaires pilotés par les Sonoff Basic ;
- et que les LED vertes et rouges indiquent l'état dudit luminaire : rouge pour éteint et vert pour allumé.

Pour le premier point, nous avons comme déclencheur un changement d'état du bouton. Le fait que la logique soit inversée côté *firmware* ESPHome n'entre pas ici en ligne de compte, pour HA le bouton appuyé passe de '**off**' à '**on**'. C'est notre déclencheur. Voici donc ce que cela nous donne dans un premier script YAML que nous appelons **sw1button.yaml** :

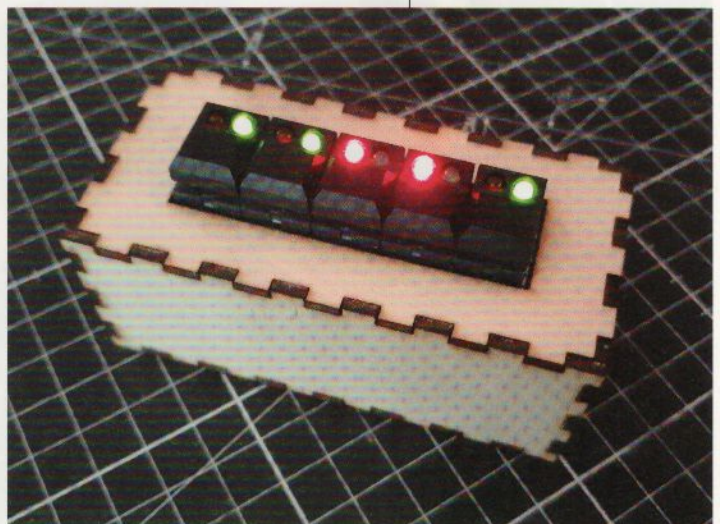
```
alias: SW1 Tardis toggle
description: ''
trigger:
  platform: state
  entity_id: binary_sensor.swbutton1
  to: 'on'
action:
  service: switch.toggle
  entity_id: switch.sonoff5_basic_relais
mode: single
```

Les mots-clés **alias** et **description** parlent d'eux-mêmes, et sont obligatoires. L'alias est le nom sous lequel apparaîtra l'automatisation dans l'interface web, sous **Paramètres, Automatisation et scènes**, et onglet **Automatisation**. Notez que l'interface en question vous signalera qu'il n'est pas possible de modifier de telles automatisations, car elles ne sont pas placées dans **automations.yaml** et n'ont pas d'ID. Ce n'est pas un problème, mais un simple éditeur de code serait peut-être le bienvenu à cet endroit, un peu comme pour des **dashboards**.

Quoi qu'il en soit, l'aspect important de ce script est sa syntaxe. Nous y trouvons un déclencheur (**trigger**) qui concerne un état (**state**) de **binary_sensor.swbutton1** qui passe (**to**) à **on**. Ceci se lit comme « le state de l'entité **binary_sensor.swbutton1** va vers **on** ». En d'autres termes, un appui sur le bouton.

Nous n'avons pas de condition et l'action à opérer est un appel à un service **switch.toggle** (basculer un interrupteur) désignant l'entité **switch.sonoff5_basic_relais**, notre cinquième Sonoff Basic

Voici la version finale de ma « télécommande domotique Wi-Fi », qui trône depuis quelques semaines sur mon bureau et qui est utilisée quotidiennement. Un petit coup de peinture sera nécessaire (un jour) pour finir la réalisation proprement.



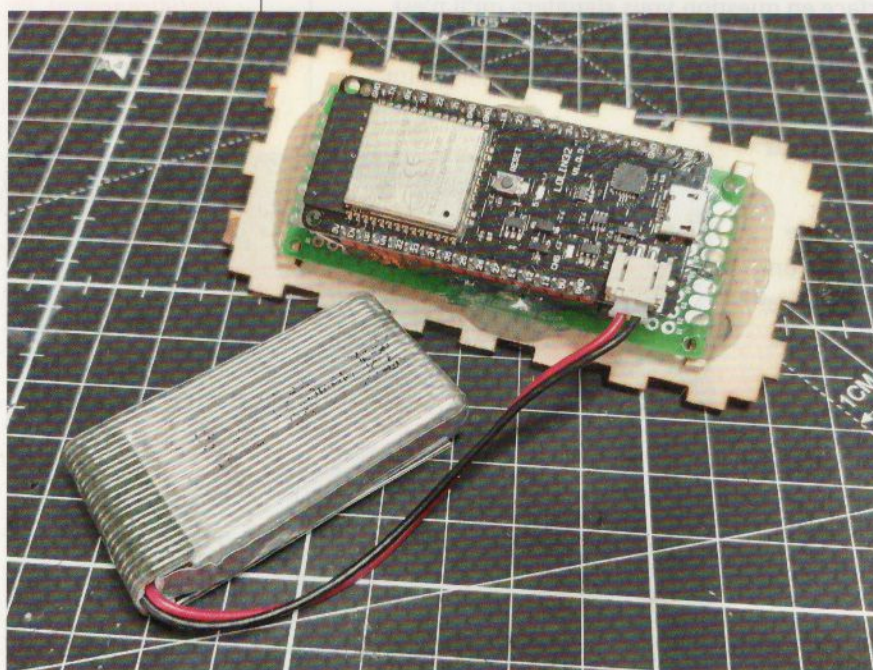
Le devkit Wemos LOLIN32 v1.0.0 possède un contrôleur de charge pour un accu LiPo. Ceci rend automatiquement le montage autonome pour plusieurs heures dans sa configuration actuelle sans la moindre optimisation, ce qui est déjà bien suffisant pour mon usage.

configuré dans le premier article du numéro 46. La méthode de fonctionnement peut être vue, si vous êtes programmeur, comme une IPC ou un appel de fonction avec un passage d'argument. Le service `switch.toggle` est une « chose » qui peut être invoquée envers n'importe quelle entité de type `switch`. Les autres services utilisables sont `switch.turn_on` et `switch.turn_off`, et vous les trouverez facilement dans l'interface web en vous rendant dans **Outils de développement**, onglet **Services** et en commençant à taper le nom qui vous intéresse (ici « `switch` »). Notez que vous pouvez également utiliser `homeassistant.toggle` ici, un service générique utilisable avec n'importe quelle entité supportant d'être « basculée » (*toggle*), comme un `switch`, une `light`, une autre automatisation, etc. Enfin, le `mode: single` de fin spécifie le comportement à adopter si un déclenchement survient alors que l'automatisation est déjà en cours d'exécution. Le mode `single` ne provoquera pas une nouvelle exécution dans ce cas, mais produira simplement un avertissement. On peut également choisir les modes `restart`, `queued` ou `parallel`, respectivement pour une nouvelle exécution, juste après celle

en cours, une exécution « empilée » après toutes celles déjà à exécuter, ou une exécution en parallèle à ce qui s'exécute déjà.

Notre `sw1button.yaml` dûment complété, enregistré, rechargé dans HA, testé et pleinement validé, nous pouvons dupliquer exactement la même configuration pour `sw2button.yaml`, `sw4button.yaml` et `sw5button.yaml`, activant et désactivant à loisir différents éléments de notre installation. Et `sw3button.yaml`, me demanderez-vous ? Nous avons quelque chose de spécial prévu pour lui, mais nous verrons cela plus tard. Avant, nous devons gérer les LED.

Les LED ne sont pas plus difficiles à contrôler que les boutons et le principe est exactement le même. Nous devons cependant prendre en considération deux cas de figure et non un seul (comme avec la pression sur le bouton). Ici, nous avons deux situations, l'une est un passage du Sonoff à **on** où nous basculons la LED verte à **on** et la LED rouge à **off** et l'autre dans le cas inverse, c'est-à-dire le Sonoff changeant son état à **off** où nous passons la LED verte à **off** et la rouge à **on**. Ce n'est pas le seul moyen d'arriver à ce résultat, mais mon approche consiste alors à prévoir deux scripts YAML.



Le premier pour un passage « à vert » (`sw1gogreen.yaml`) :

```
alias: SW1 go green
description: ''
trigger:
  platform: state
  entity_id: switch.sonoff5_basic_relais
  to: 'on'
action:
  - service: light.turn_on
    entity_id: light.sw_greenled1
  - service: light.turn_off
    entity_id: light.sw_redled1
mode: single
```

Et un second pour un passage « à rouge » (`sw1gored.yaml`) :

```
alias: SW1 go red
description: ''
trigger:
  platform: state
  entity_id: switch.sonoff5_basic_relais
  to: 'off'
action:
  - service: light.turn_off
    entity_id: light.sw_greenled1
  - service: light.turn_on
    entity_id: light.sw_redled1
mode: single
```

Nous retrouvons ici les mêmes éléments que précédemment et les mêmes mots-clés, nous ne reviendrons donc pas dessus. La seule petite nuance tient dans le fait que nous avons plusieurs actions et appelons donc deux services pour changer l'état des LED.

Notez que les LED de la télécommande ne reflètent pas réellement une situation à un instant « t », mais réagissent à un changement. Ainsi, si nous activons manuellement (physiquement ou via l'interface web) le Sonoff Basic en question, cet état de fait sera bien répercuté sur les LED, mais si cela intervient alors que la télécommande est absente (pas de Wi-Fi, rupture d'alimentation du montage, etc.), son retour ne signifiera pas une synchronisation des LED. Ce n'est pas bien important dans le cas présent, mais c'est quelque chose à garder en tête et qui peut être important dans d'autres situations.

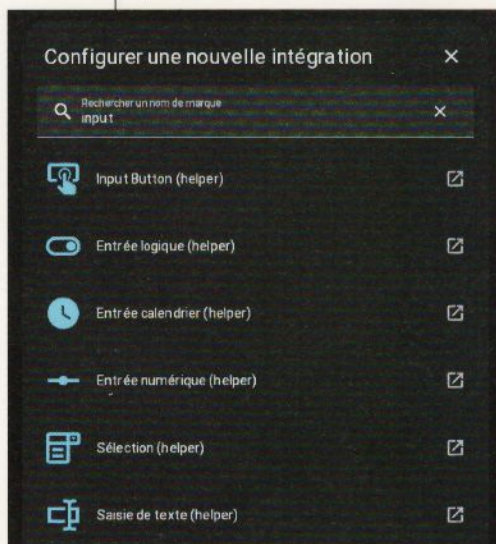
Au final, nous avons en tout 15 scripts YAML, 5 pour les boutons et 10 pour les LED. Il serait possible d'architecturer tout cela différemment, mais j'apprécie énormément la division en de multiples scripts, permettant d'adapter rapidement une configuration spécifique sans toucher au reste. Et justement, le bouton 3, qui aura commencé sa vie comme les autres est un peu spécial. Voyons donc cela...

3.4 Automation, input boolean et scripts

Si vous avez bien suivi ma petite histoire du début d'article, l'idée initiale de cette automatisation est de gérer un changement d'ambiance pour mon salon, d'une simple pression sur un bouton. L'arrivée des T4EUC1 a enfin permis d'atteindre cet objectif : lorsqu'il est enfin temps de se détendre un peu, je veux une automatisation qui éteindra les lumières au plafond (en agissant sur les T4EUC1) et activera des lumières plus douces, contrôlées par les Sonoff Basic. Ceci doit être déclenché par une pression sur le bouton 3, mais inversement, si le mode « ambiance » est déjà actif, je ne veux pas pour autant allumer les lumières du plafond. Généralement, la

sortie du mode « ambiance » du salon rime avec l'entrée en mode « dodo » du rédac-chef, après un passage par le mode « brosse à dents », bien entendu (il pourrait d'ailleurs être amusant d'allumer la lumière à la salle de bain et dans la chambre automatiquement), il n'y a donc pas de raison de rallumer la lumière et ceci sera éteint peu après.

Ce qui nous intéresse ici, c'est de gérer un mode susceptible de changer d'état, mais qui ne relève d'aucune entité physique présente. Il nous faut donc une entité « virtuelle » correspondant au mode « ambiance » que nous pouvons intégrer dans Home Assistant. Et ça tombe bien, ça existe ! Rendez-vous simplement dans **Paramètres, Appareils et services**, cliquez **Ajouter une intégration** et cherchez « input », et vous verrez apparaître ceci (version 2023.3.6) :



Choisissez **Entrée logique (helper)** et vous serez invité à donner un nom à cette nouvelle intégration de type « entrée logique » ou « *input boolean* » en anglais. Ici, j'ai choisi « Mode ambiance », qui se traduira, dans nos scripts, en **mode_ambiance** et plus exactement **input_boolean.mode_ambiance** (son ID).

Cette nouvelle entité va nous permet d'avoir un « point central » représentant l'état du fameux mode et nous pouvons alors créer des automatisations qui l'utilise, à commencer par le fameux bouton 3 de la télécommande (**sw3button.yaml**) :

```
alias: SW3 Ambiance toggle
description: ''
trigger:
  platform: state
  entity_id: binary_sensor.swbutton3
  to: 'on'
action:
  service: input_boolean.toggle
  entity_id: input_boolean.mode_ambiance
mode: single
```

Ceci est très similaire aux scripts YAML précédents puisque la seule chose qui change est l'entité concernée par l'action. Il en va de même pour le contrôle de l'état des LED avec **sw3gogreen.yaml** :

```
alias: SW3 go green
description: ''
trigger:
  platform: state
  entity_id: input_boolean.mode_ambiance
  to: 'on'
action:
  - service: light.turn_on
    entity_id: light.sw_greenled3
  - service: light.turn_off
    entity_id: light.sw_redled3
mode: single
```

Et **sw3gored.yaml** :

```
alias: SW3 go red
description: ''
trigger:
  platform: state
```


Home assistant

- Home Assistant phase 3 : interrupteurs, contrôle et automatisation -

```
entity_id: input_boolean.mode_ambiance
to: 'off'
action:
- service: light.turn_off
  entity_id: light.sw_greenled3
- service: light.turn_on
  entity_id: light.sw_redled3
mode: single
```

À ce stade, nous avons donc un bouton qui agit sur l'*input boolean* et des LED qui reflètent ces changements, mais ça s'arrête là. Il nous reste à définir ce qu'implique effectivement ce changement et pour cela, nous avons besoin de deux autres automatisations. Celles-ci définissent ce qui se passe lorsqu'on entre (**off** à **on**) ou qu'on sort (**on** à **off**) du mode d'ambiance. Mais plutôt que d'avoir une série d'actions dans l'automatisation, nous allons réunir cela ailleurs, dans des scripts. Ceci prendra la forme de deux fichiers YAML placés dans `~/config/myscripts` et nous pouvons créer deux automatisations en conséquence. La première se déclenche lorsque `input_boolean.mode_ambiance` passe à **on** (`ambiancegoon.yaml`) :

```
alias: Ambiance ON
description: ''
trigger:
  platform: state
  entity_id: input_boolean.mode_ambiance
  to: 'on'
action:
- service: script.ambiance_on
mode: single
```

Et la seconde lorsqu'on passe à **off** (`ambiancegooff.yaml`) :

```
alias: Ambiance OFF
description: ''
trigger:
  platform: state
  entity_id: input_boolean.mode_ambiance
  to: 'off'
action:
- service: script.ambiance_off
mode: single
```

Nous retrouvons ici l'utilisation d'un service, mais qui cette fois est l'appel d'un script avec **script**, suivi de son nom. L'intérêt des scripts, dans ce cas, est d'avoir, à un seul endroit, l'ensemble des actions à opérer. Ainsi, ce changement d'ambiance n'est pas intimement lié à ces deux automatisations et pourra être déclenché par d'autres automatisations, voire être exécuté manuellement. De plus, si nous ajoutons des luminaires ou d'autres éléments à contrôler, seuls ces deux scripts devront être modifiés et non toutes les automatisations qui en provoquent l'exécution.

Les scripts ne sont pas plus difficiles à écrire que les automatisations. Il s'agit simplement d'une suite ou séquence d'actions devant être exécutées. Voici celui pour le passage en mode ambiance (`ambianceON.yaml`):

```
ambiance_on:
  alias: Ambiance ON
  sequence:
    # TARDIS ON
    - service: switch.turn_on
      entity_id: switch.sonoff5_basic_relais
    # sol ON
    - service: switch.turn_on
      entity_id: switch.sonoff2_basic_relais
    # Bahut ON
    - service: switch.turn_on
      entity_id: switch.sonoff3_basic_relais
    # Haut ON
    - service: switch.turn_on
      entity_id: switch.sonoff4_basic_relais
    # Salon Plafond Sud OFF
    - service: switch.turn_off
      entity_id: switch.sonofft4eulc01_relais
    # Salon Plafond Nord OFF
    - service: switch.turn_off
      entity_id: switch.sonofft4eulc02_relais
```

Et celui pour la sortie (`ambianceOFF.yaml`):

```
ambiance_off:
  alias: Ambiance OFF
  sequence:
    # TARDIS
    - service: switch.turn_off
      entity_id: switch.sonoff5_basic_relais
    # sol
    - service: switch.turn_off
      entity_id: switch.sonoff2_basic_relais
    # Bahut
    - service: switch.turn_off
      entity_id: switch.sonoff3_basic_relais
    # Haut
    - service: switch.turn_off
      entity_id: switch.sonoff4_basic_relais
```

On retrouve naturellement une série d'appels à des services pour chaque entité concernée et pas grand-chose d'autre. Les scripts peuvent être plus complexes, utiliser des variables, gérer des données à passer en argument des services, mais ceci n'a que peu d'importance ici.

Et voici donc comment notre télécommande peut être utilisée pour activer/désactiver des luminaires, et à l'aide d'une simple pression sur le bouton central, changer entièrement la configuration de l'éclairage, sans se lever et faire le tour des interrupteurs... Du pur bonheur et je peux vous dire qu'on en prend rapidement l'habitude, sans aucun désir de revenir en arrière. Mais le plus intéressant est la généricité de l'ensemble, car tout reste pilotable individuellement, physiquement, mais également via l'interface web ou mobile.

4. UNE AUTRE APPROCHE AVEC JUSTE UN SONOFF BASIC

Peut-être trouvez-vous le fait de créer une télécommande dédiée un peu trop poussé à votre goût et n'avez pas réellement envie de passer plusieurs heures à créer un montage de toutes pièces, avec tout ce que cela implique, dont le fait que le résultat devra être relativement présentable. Il existe une autre manière de déclencher un appel à un service, et ce, directement sur une entité déjà présente dans notre installation : l'un des Sonoff.

Certes, l'entrée physique (le bouton) possède déjà un usage dédié, mais en vérité, ce n'est pas tout à fait exact. C'est une pression (pour un Sonoff Basic) ou un contact (pour les interrupteurs) **ponctuel et bref** qui est pris en compte et non quelque chose de plus « subtil ». Considérez donc l'extrait suivant pour le script YAML ESPHome pour un de ces interrupteurs :

```
binary_sensor:
  - platform: gpio
    pin:
      number: GPIO00
      mode:
        input: true
        pullup: true
        inverted: true
    name: $devicename Basic Bouton
    on_click:
      - min_length: 20ms
        max_length: 900ms
        then:
          - switch.toggle: relay
      - min_length: 1000ms
        max_length: 5000ms
        then:
          - homeassistant.service:
              service: input_boolean.toggle
              data:
                entity_id: input_boolean.mode_ambiance

switch:
  - platform: gpio
    name: $devicename Basic Relais
    pin: GPIO12
    id: relay
```

Le simple **on_press** : **- switch.toggle: relay** a laissé place à quelque chose de plus étoffé. Nous utilisons maintenant **on_click**, nous offrant davantage de contrôle en nous permettant de spécifier quoi faire en fonction du temps durant lequel l'état de l'entrée, le **binary_sensor** sur GPIO00, change. Deux cas de figure se présentent :

- une activation entre 200 et 900 millisecondes correspond à ce que nous avons avec **on_press**, c'est un simple contact franc, tout en évitant l'éventuel « bruit » (<20 ms) ;
- et une activation de plus d'une seconde avec un maximum de 5 secondes, qu'on considérera comme un « appui long ».

Dans le premier cas, on bascule le relais normalement, mais dans le second, nous demandons à Home Assistant d'appeler le service changeant l'état de notre « entrée logique » qui contrôle le mode d'ambiance. Ainsi, dans mon cas, en ayant un Sonoff Basic à portée de main à proximité du bureau (fixé sous le retour et contrôlant la barre lumineuse placée au-dessus via un appui bref), je peux me servir du minuscule bouton qui s'y trouve comme d'un équivalent du bouton 3 de la télécommande ESP32 présenté ci-avant. Mieux encore, en reflashant l'ensemble

des Sonoff (Basic, Touch et T4EU1C), le mode peut être activé (basculé en fait) de n'importe où. Notez que la syntaxe est légèrement différente entre ESPHome et HA, concernant l'appel aux services, mais la logique est parfaitement identique. Il est important également de remarquer que ce n'est pas vraiment le *firmware* qui appelle le service, mais qu'il demande à HA de le faire.

Ce genre de fonctionnalité nous permet également de régler un autre problème : le *deep sleep* biologique, alias « c'est l'heure du dodo ». J'ai évoqué cela précédemment en parlant de la sortie du mode *ambiance*. Lorsqu'il est l'heure d'aller se coucher, il peut être intéressant de disposer d'une fonctionnalité permettant de tout éteindre, de préférence accessible depuis l'interrupteur se trouvant à l'entrée du salon (qui est fermé pour la nuit afin d'éviter que certains quadrupèdes à moustaches ne jouent avec toutes ces amusantes choses électroniques qu'on peut faire tomber des tables et bureaux). Et avec **on_click** et une pression longue, pour cet interrupteur précis, cela devient un jeu d'enfant. Tout ce qu'il nous faut, c'est un nouveau script (**salonOFF.yaml**) :

```
salon_off:
  alias: Salon Full OFF
  sequence:
    # Bahut
    - service: switch.turn_off
      entity_id: switch.sonoff3_basic_relais
    # Barre
    - service: switch.turn_off
      entity_id: switch.sonoff1_basic_relais
    # sol
    - service: switch.turn_off
      entity_id: switch.sonoff2_basic_relais
    # Bahut
    - service: switch.turn_off
      entity_id: switch.sonoff3_basic_relais
    # Haut
    - service: switch.turn_off
      entity_id: switch.sonoff4_basic_relais
    # TARDIS
    - service: switch.turn_off
      entity_id: switch.sonoff5_basic_relais
    # Plafond Sud OFF
    - service: switch.turn_off
      entity_id: switch.sonofft4eulc01_relais
    # Plafond Nord OFF
    - service: switch.turn_off
      entity_id: switch.sonofft4eulc02_relais
    # Ambiance OFF
    - service: input_boolean.turn_off
      entity_id: input_boolean.mode_ambiance
    # Bench OFF
    - service: switch.turn_off
      entity_id: switch.sonoff6_basic_relais
    # pause - touch control
    - delay:
        seconds: 5
    # Bench ON
    - service: switch.turn_on
      entity_id: switch.sonoff6_basic_relais
```

Le Sonoff 6 est spécial, car connecté à une lampe/loupe disposant d'un bouton réglant l'intensité lumineuse de fort à éteint, décrémentant à chaque pression avant de boucler pour revenir à la pleine intensité. Une double pression éteint totalement les LED et, bien entendu, en cas de rupture et reprise d'alimentation, la lampe reste éteinte. Ceci n'est pas gérable directement avec un Sonoff Basic, mais ce n'est pas bien grave, puisqu'activer

cet éclairage n'a de sens que si l'on est au *bench* en train de travailler, je n'ai pas besoin de l'allumer à distance. L'extinction en revanche est utile, mais il convient de réalimenter le dispositif quelques secondes après, pour qu'il soit prêt à servir à nouveau. C'est donc la raison d'être de ce **delay** de 5 secondes en fin de script, et du **switch.turn_on** qui s'en suit (une durée inférieure éteint bien le dispositif, mais le rallume étrangement à pleine puissance ensuite).

5. UN PETIT MOT À PROPOS DES SCÈNES

L'approche présentée ci-avant n'est pas la seule possible concernant la gestion de différents styles d'ambiance pour une pièce, ou d'ailleurs pour toute une habitation. Il est possible de voir ce genre de « composition » d'entités actives/inactives comme la description d'une situation à un instant « t » ou, autrement dit, une scène. Là, il ne s'agit plus d'actions, mais de l'énumération des états que doivent prendre certaines entités.

Comme avec les automatisations et les scripts, la configuration manuelle et la

division en multiples fichiers personnalisés passe par la modification du `~/config/configuration.yaml` de votre HA, en changeant :

```
scene: !include scenes.yaml
```

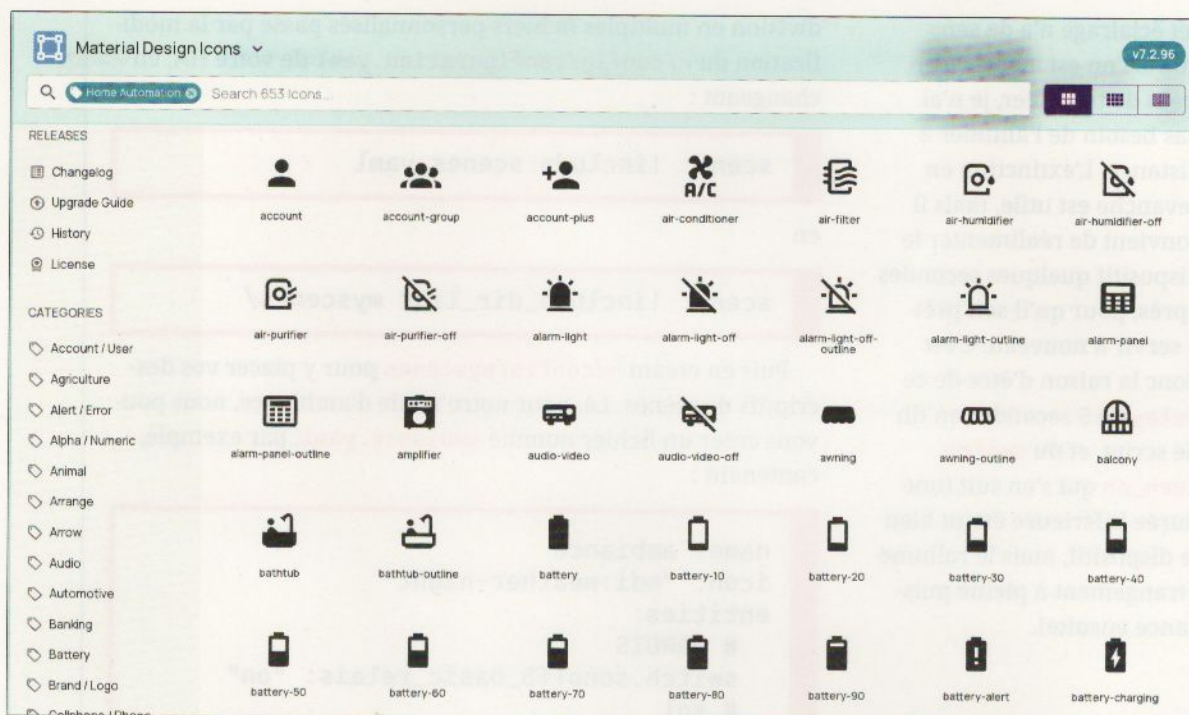
en

```
scene: !include_dir_list myscenes/
```

Puis en créant `~/config/myscenes` pour y placer vos descriptifs de scènes. Là, pour notre mode d'ambiance, nous pouvons créer un fichier nommé `ambiance.yaml`, par exemple, contenant :

```
name: ambiance
icon: "mdi:weather-night"
entities:
  # TARDIS
  switch.sonoff5_basic_relais: "on"
  # sol
  switch.sonoff2_basic_relais: "on"
  # Bahut
  switch.sonoff3_basic_relais: "on"
  # Haut
  switch.sonoff4_basic_relais: "on"
  # Plafond Sud
  switch.sonofft4eu1c01_relais: "off"
  # Plafond Nord
  switch.sonofft4eu1c02_relais: "off"
```

Rien de bien complexe ici puisque la description d'une situation n'est finalement qu'une simple liste d'états. Notez la présence de **icon** également disponible pour les scripts et permettant de décorer ces éléments dans l'interface web de façon amusante. Le mot-clé utilisé est tiré du projet « *Material Design Icons* » (MDI), une bibliothèque communautaire d'icônes reposant sur les *Material Design guidelines* de Google et proposant une belle collection de quelque 6500 icônes uniques (et assez plaisantes). Il est possible de choisir une icône qui vous plaît via l'interface web de HA, mais la méthode la plus simple est de pointer votre navigateur sur <https://pictogrammers.com/library/mdi/>, de procéder à une recherche et de flâner un peu pour trouver votre bonheur. Recopiez ensuite simplement le nom dans le YAML de votre script ou scène et le tour est joué.



Le projet « Material Design Icons » ou MDI regroupe des milliers d'icônes permettant de personnaliser, entre autres, son installation Home Assistant. Celles-ci sont déjà intégrées dans le système, tout ce que vous avez à faire est de choisir celle qui vous plaît (ce qui prend le plus de temps) et de simplement utiliser son nom préfixé de « mdi: ».

Pour utiliser cette scène, nous pouvons passer par l'interface web, via **Paramètres, Automatisation et scènes**, onglet **Scènes**, repérer votre scène dans la liste, cliquer sur les trois petits points à droite et choisir **Activé**. Mais le plus simple est, bien entendu, d'utiliser une automatisation pour cela. Ainsi, en lieu et place de notre script, nous pouvons utiliser la scène dans notre automatisation `ambiancegoon.yaml` de cette manière :

```
alias: Ambiance ON
description: ''
trigger:
  platform: state
  entity_id: input_boolean.mode_ambiance
  to: 'on'
action:
  service: scene.turn_on
  target:
    entity_id: scene.ambiance
  mode: single
```

Nous avons toujours un service qui est appelé mais cette fois, il s'agit de celui permettant d'activer une scène et nous spécifions la cible avec **target**.

Je ne suis pas un grand fan des scènes, sans doute parce que je suis davantage à l'aise avec les langages de programmation que ceux de description (comme HTML ou VHDL), plus « actifs », plus dynamiques. Mais également

parce que nous avons des limitations dans ce cas. Nous pourrions, par exemple, faire de même pour créer la scène `salonfulloff.yaml`, décrivant tous les luminaires du salon à `off` pour gérer l'extinction au moment d'aller se coucher et lancer le script depuis l'interrupteur Sonoff, mais il y a un problème : une scène décrit une situation et n'est absolument pas capable de gérer le Sonoff Basic de ma lampe-loupe devant s'éteindre et se rallumer presque immédiatement. Ce n'est pas une procédure à suivre, mais une simple description.

Vous me direz que rien ne m'empêche de garder le script d'extinction totale et de gérer l'ambiance avec une scène, et le reste avec des scripts. C'est vrai, mais la gestion tournera alors autour de trois types de configurations différentes (automatisation, script et scène), alors que deux peuvent suffire. Vous aurez sans doute un point de vue différent, ou une situation peut être plus simple. Quoi qu'il en soit, les scènes méritent d'être au minimum connues.

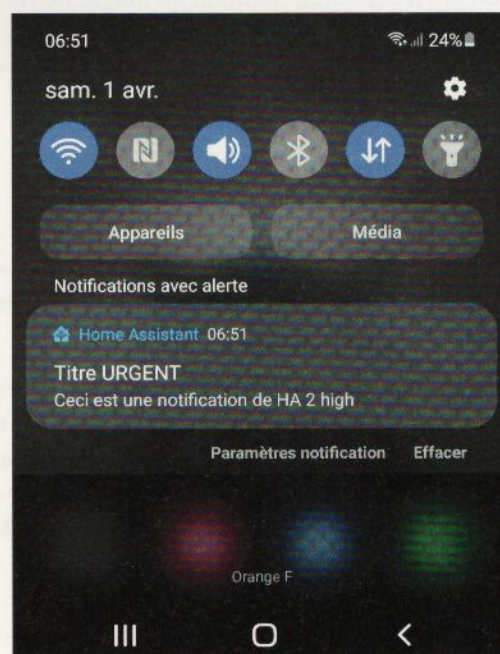
6. BONUS : NOTIFICATION SUR MOBILE

Lorsque vous installez l'application Home Assistant sur mobile, ceci vous permet d'accéder très facilement à l'interface utilisateur, appelée *Lovelace* et reposant, entre autres, sur le *framework* JS Polymer. Cette interface apparaît comme une application, mais est en réalité la même que celle que vous voyez dans le navigateur, c'est ce qu'on appelle une

Progressive Web App ou PWA. Ce qui est très intéressant, en revanche, est le fait que le smartphone devient également une intégration HA et vous donne accès à toute une liste d'entités. Mon Samsung Galaxy A40 (SM-A405FN), par exemple, fournit quelques 80 entités allant du niveau de charge de la batterie au capteur de lumière intégré en passant par l'IP utilisée, la luminosité de l'écran ou encore les différents volumes audio (ce qui peut être intéressant à contrôler dans le fameux mode d'ambiance).

C'est intéressant et on pourra envisager d'utiliser ces capteurs, peut-être avec d'anciens smartphones pour différents usages. Il est même possible de recycler ce genre d'appareil pour en faire des webcams Wi-Fi relativement simplement (avec *IP Webcam* sous Android, par exemple [8]). Ceci ouvre plein de possibilités...

Mais ici, je parle de votre smartphone que vous utilisez quotidiennement et qui permet, via l'application HA [9], de recevoir des notifications. Bien entendu,



Voici un exemple de notification que l'application Android Home Assistant est en mesure d'afficher. Ceci cependant ne fonctionnera que si l'application est capable de communiquer avec l'installation et donc uniquement si le smartphone est connecté au réseau local en Wi-Fi.

comme nous parlons d'une configuration locale, ceci ne fonctionnera que lorsque vous vous trouvez sur le même réseau et donc sur le point d'accès Wi-Fi de votre habitation. Mais ce n'est pas une limitation très importante, il est également possible d'envisager des configurations plus poussées (e-mail, SMS, etc.) en utilisant d'autres intégrations ou montages ESPHome, par exemple (via un module Sim800L).

Une fois votre smartphone et son application Home Assistant intégrée dans votre configuration, vous pouvez très facilement tester le système de notification via l'interface web. Pour cela, rendez-vous dans **Outils de développement**, onglet **Service** et dans le formulaire, commencez à taper « notification ». Devrait alors apparaître le nom d'un service se terminant par le nom (ou modèle) de votre smartphone, dans mon cas `notify.mobile_app_sm_a405fn` (soit « `notify.mobile_app_` » suivi de l'ID du périphérique). Vous pouvez alors utiliser le mode UI ou passer en mode YAML (via le bouton correspondant), chose que je préfère. Là, invoquer le service de notification peut se faire ainsi :

```
service: notify.mobile_app_sm_a405fn
data:
  message: Ceci est une notification de HA
  title: Titre URGENT
  data:
    ttl: 0
    priority: high
```

Vous pouvez également utiliser le service `notify.notify` qui est une notification « globale » qui sera donc envoyée de manière non ciblée à l'ensemble des périphériques susceptibles de l'afficher, et non simplement **un** smartphone. Ceci est, à mon sens, peu souhaitable en particulier dans un environnement familial où tout le monde aura l'application installée. Mais si tel est votre souhait, voici le YAML correspondant :

```
service: notify.notify
data:
  message: coucou test 2
  data:
    priority: high
```

La page de la documentation officielle [10] [11] détaille les différents éléments de configuration utilisables (comme ici `priority`).

Notez que ce type de notification ne doit pas être confondu avec celles apparaissant dans l'interface web (quand un nouveau périphérique est détecté, par exemple). Là, il s'agit de « notifications persistantes », pouvant être testées également, avec quelque chose comme :

```
service: notify.persistent_notification
data:
  message: Coucou encore
  title: Et de 3 !
```


Bien entendu, les tests sont très amusants et très pratiques pour se faire la main sur le formatage et les différents paramètres, mais le réel intérêt est de provoquer cela via des automatisations. Si nous reprenons l'exemple évoqué dans le premier article (Hackable 46), nous avons un capteur PIR placé dans l'entrée et contrôlant l'éclairage correspondant. Ceci était hypothétique au moment de la rédaction du numéro 46, puisque je n'avais pas encore reçu les interrupteurs muraux Sonoff. Depuis, l'installation a été complétée et l'automatisation ressemble à ceci :

```
alias: Allumage lumière PIR
description: ''
trigger:
  platform: state
  entity_id: binary_sensor.entree_pir
  to: 'on'
action:
  - service: switch.turn_on
    entity_id: switch.sonofft4eulc03_relais
```

et :

```
alias: Extinction auto PIR
description: ''
trigger:
  platform: state
  entity_id: binary_sensor.entree_pir
  to: 'off'
  for:
    seconds: 30
action:
  service: switch.turn_off
  entity_id: switch.sonofft4eulc03_relais
```

Je ne reviendrai pas sur l'architecture générale du YAML (qui doit être claire à présent) ou la gestion des 30 secondes d'inactivité évoquée précédemment. Ce qui nous intéresse ici est d'ajouter une notification sur le smartphone, lorsque le capteur PIR est déclenché. Pour ce faire, il nous suffit de compléter le premier fichier YAML avec un appel supplémentaire au nouveau service :

```
- service: notify.mobile_app_sm_a405fn
  data:
    message: Mouvement détecté dans Entrée
    title: DETECTION URGENT
    data:
      ttl: 0
      priority: high
```


On recharge ensuite les automatisations depuis les **Outils de développement** de l'interface web et il n'y a plus qu'à gesticuler comme un demeuré devant le capteur (ne faites pas semblant, vous savez exactement de quoi je parle). Bien entendu, nous recevons rapidement (merci **priority: high**) une notification sur le mobile, comme attendu. Nous avons donc un moyen d'être informés d'un mouvement, où que nous soyons dans l'habitation. Ceci peut être vu comme les prémices d'un système de surveillance, de sécurité ou d'alarme entièrement basé sur HA. D'autres styles de capteurs existent et sont pris en charge par ESPHome, comme les capteurs à effet Hall (en module ou intégrés aux ESP32) permettant de détecter l'ouverture de portes ou fenêtres. De plus, un certain nombre de capteurs de mouvement sont bien plus fiables que les capteurs PIR, c'est le cas par exemple des capteurs micro-ondes comme le module HW-MS03 s'interfaçant de la même manière qu'un capteur PIR.

Et c'est justement là la raison pour laquelle je n'ai utilisé ce système de notification que très peu de temps : les faux positifs et déclenchements aléatoires sont très nombreux, y compris

dans un environnement qui semble parfaitement adapté. Avoir un déclenchement erroné de luminaire d'une trentaine de secondes est une chose, recevoir des notifications à foison pour rien en est une autre. Je testerai prochainement le module HW-MS03 et changerai peut-être d'avis, mais pour l'instant ce n'est pas à l'ordre du jour, les notifications sont trop intrusives pour moi.

7. BONUS 2 : LES MACHINES DU RÉSEAU

Si vous disposez de plusieurs machines sur votre réseau local et n'avez pas installé de solution de supervision, il peut être intéressant d'utiliser Home Assistant pour garder un œil, très sommairement, sur l'état de celles-ci. Attention, il ne s'agit pas ici d'une fonctionnalité très poussée, mais de simplement avoir un aperçu de ce qui est ou non en fonction et répondra via ICMP (*ping*) sur le réseau.

Pour obtenir cela et avoir, pour chaque hôte concerné, un **binary_sensor** indiquant sa présence, nous pouvons utiliser la plateforme **ping** [12]. Ceci se configure directement dans votre `~/config/configuration.yaml` en y ajoutant un peu de YAML :

```
binary_sensor:
  - platform: ping
    name: "Host Sun Blade"
    host: bladeobsd.local
    count: 5
    scan_interval: 30
  - platform: ping
    name: "Host Beastie X61"
    host: beastie61.local
    count: 5
    scan_interval: 30
  - platform: ping
    name: "Host Beast"
    host: beast.local
    count: 5
    scan_interval: 30
  - platform: ping
    name: "Host FBSD Ryzen"
```



```
host: fbsdryz.local
count: 5
scan_interval: 30
- platform: ping
  name: "Host NetX31"
  host: netx31.local
  count: 5
  scan_interval: 30
- platform: ping
  name: "Host ReadyNAS Media"
  host: 192.168.0.9
  count: 5
  scan_interval: 30
```

Pour chaque hôte à surveiller, nous spécifions un nom (**name**), une adresse ou un nom d'hôte DNS/mdNS (**host**), un nombre de requêtes ICMP à envoyer (**count**) et l'intervalle de temps séparant les vérifications (**scan_interval**). Ceci fait, il faudra recharger la configuration de HA via les outils de développement de l'interface web (qui avec la dernière version peut être un **Rechargement rapide** et non un vrai redémarrage) et vous pourrez ajouter ces nouvelles entités à vos tableaux. Ainsi, d'un coup d'œil, vous saurez quelle machine est allumée ou éteinte et éviterez peut-être de gâcher du courant pour rien par simple étourderie (comme moi).

8. POUR FINIR

Comme nous venons de le voir, les automatisations donnent une tout autre dimension à la notion de domotique « cloudless » et permettent de réellement améliorer votre confort. C'est quelque chose que je ne considérerais sincèrement pas il y a une paire d'années, en particulièrement lorsqu'on me parlait simplement de pouvoir monter la température des lieux ou d'ouvrir les volets à distance. C'est là, étrangement, un point souvent mis en avant, y compris par les fabricants de solutions domotiques, dont l'intérêt m'échappe totalement. Se simplifier réellement la vie en faisant en sorte que son environnement réagisse automatiquement, lorsqu'on est présent, est beaucoup plus démonstratif des bénéfices d'un tel investissement qui, dans le cas présent, est relativement modeste.

On est, bien entendu, toujours tenté de rendre le système accessible à distance, via Internet, mais c'est non seulement très dangereux (HA ne me semble pas pensé dans ce sens du tout), mais n'est également que peu intéressant, sauf peut-être pour montrer sa magnifique installation à qui veut bien la voir. Des solutions sécurisées existent, comme le fait d'utiliser un VPN (privé) rendant l'interface web et l'installation accessibles à distance, mais les quelques *add-ons* disponibles sont relativement anciens (>4 ans), non maintenus à jour et posent souvent problème. Une autre approche possible est de ne pas toucher à HA, mais de structurer sa configuration réseau et le routage de manière à arriver à un résultat équivalent. Ceci demande une certaine expertise, une architecture déjà complexe par rapport à une simple *box* internet domestique et devra être motivé par bien plus qu'un « *si je veux, je peux allumer une lumière au salon quand je suis au restaurant* ».

À titre d'exemple dans ce sens et en combinant un peu tout ce que nous venons de voir, imaginez ce qu'il est possible de faire en termes de simulation de présence

avec une telle installation. Nous sommes bien au-delà du bon vieux « je laisse la TV allumée pour faire croire qu'il y a quelqu'un » et ceci sera certainement plus efficace qu'une notification arrivant sur votre smartphone, alors que vous êtes à 300 km de là, vous signalant une possible intrusion (si ce n'est pas un faux positif). Quoi qu'il en soit, si vous n'êtes pas déjà convaincu qu'une solution domotique est quelque chose de typiquement et uniquement « local », je ne pense pas pouvoir vous en convaincre à ce stade.

Mon installation va continuer sa petite vie et nous reviendrons peut-être sur le sujet si quelque chose de notable et digne d'intérêt est ajouté. Les idées ne manquent pas, tant pratiques que motivées par la simple curiosité : gestion de l'arrosage extérieur, utilisation de *tags* NFC pour le contrôle, gestion de *streaming* audio, supervision de la consommation électrique (pas facile avec un compteur qui doit avoir au moins 50 ans), etc. Non, ce qui manque c'est le temps et sans doute votre patience si la domotique n'est pas l'un de vos centres d'intérêt premier. Home Assistant restera cependant en filigrane des pages de ce magazine, car j'avoue qu'à ce stade, il m'est relativement difficile de ne pas me dire que n'importe quel montage passé, présent et futur peut avoir un lien avec la domotique... **DB**

RÉFÉRENCES

- [1] <https://connect.ed-diamond.com/hackable/hk-046/home-assistant-domotique-vite-fait-bien-fait>
- [2] <https://connect.ed-diamond.com/hackable/hk-047/home-assistant-quelques-changements-ajouts-et-ajustements>
- [3] <https://connect.ed-diamond.com/Hackable/hk-024/la-telecommande-arduino-le-retour-version-esp8266>
- [4] <https://connect.ed-diamond.com/Hackable/hk-019/construisez-un-emetteur-433-mhz-pour-remplacer-vos-telecommandes>
- [5] <https://fr.aliexpress.com/item/32968428602.html>
- [6] https://esphome.io/components/deep_sleep.html
- [7] https://www.home-assistant.io/docs/configuration/splitting_configuration/
- [8] <https://play.google.com/store/apps/details?id=com.pas.webcam>
- [9] <https://play.google.com/store/apps/details?id=io.homeassistant.companion.android>
- [10] <https://www.home-assistant.io/integrations/notify>
- [11] <https://companion.home-assistant.io/docs/notifications/notifications-basic/>
- [12] <https://www.home-assistant.io/integrations/ping/>

UTILISEZ VOTRE PROPRE AGENT CONVERSATIONNEL SUR PI4

Denis Bodor

Au cas où vous venez de passer les quatre derniers mois dans une cabane isolée du monde, vous ne savez sans doute pas que le buzz actuel est chatGPT/GPT4, un prototype d'agent conversationnel, ou chatbot, aux performances impressionnantes, développé par OpenAI. Ceci a déclenché une frénésie incontrôlable et une accélération massive dans l'adoption de ce genre de systèmes pour un usage pratique (autre que lui faire dire des âneries). Si vous ne viviez pas dans une cabane la moitié de l'année, inutile de détailler davantage, si ce n'est en vous expliquant comment fonctionnent ces technologies et, surtout, comment avoir votre propre chatbot fonctionnant sur un modeste SBC.



– Utilisez votre propre agent conversationnel sur Pi4 –

Avant d'attaquer le sujet, il me paraît très important de préciser que nous parlons bien ici de faire fonctionner l'agent conversationnel sur la carte et, en aucune manière, d'utiliser un script ou un programme local reposant sur une API comme celle d'OpenAI. Non, nous envisageons ici une solution vous permettant de faire fonctionner un **chatbot localement** et **sans connexion internet** permanente (il faut bien télécharger des choses).

Autre point tout aussi important, la motivation concernant ce qui va suivre est purement éducative, pour comprendre les tenants et les aboutissants, la terminologie et le principe de fonctionnement de cette technologie. J'irai même plus loin en vous annonçant ouvertement que vouloir faire ce qui va suivre sur une carte comme une Raspberry Pi est, tout simplement, complètement stupide. Ceci pour des raisons qui apparaîtront de manière plus évidente une fois que nous testerons le résultat de nos manipulations. En d'autres termes, n'espérez pas obtenir quelque chose de réellement utilisable, sauf si vous avez une patience qui relève de la pathologie.

Enfin, si tant est qu'il soit utile de le préciser, nous n'allons pas **créer** une IA, un *chatbot*, un modèle de langage, etc. Nous n'allons pas même l'entraîner, mais simplement utiliser des éléments qui existent et se trouvent sur le Net, et nous efforcer de les faire fonctionner sur un système embarqué qui n'est absolument pas fait pour cela. À ce propos, ce qui va suivre est techniquement utilisable non seulement sur d'autres SBC « musclés », mais aussi, et surtout sur une machine de bureau ou un serveur (de préférence sous GNU/Linux, mais FreeBSD a également été testé), avec des résultats bien plus démonstratifs.

1. DE QUOI PARLE-T-ON EXACTEMENT ?

Cela fait maintenant des semaines que l'écosystème IT est en ébullition et que les annonces se succèdent à une cadence infernale, tous les gros acteurs de tous les domaines annonçant soit des produits, soit des initiatives de recherche pour emboîter le pas d'OpenAI et rattraper leur retard. Ceci au point que le dernier *buzz* fantasmagique à la mode, à savoir le métavers, est presque immédiatement devenu de l'histoire ancienne, y compris pour Facebook, qui avait été renommé Meta pour l'occasion.

Des termes comme « GPT », « modèle de langage », « *tokens* », « agent conversationnel », « inférence », « nombre de paramètres », « IA générative », « alignement », « *chatbot* », « Intelligence Artificielle », fusent de tous côtés, tout comme, et c'est bien malheureux, les « il » lorsqu'il s'agit de désigner des produits comme chatGPT ou Bing chat. Soyons clair et, Sam Altman (CEO d'OpenAI) ne cesse de le répéter : il n'y a pas de « il », mais uniquement un « ça ». Et même si l'on peut effectivement faire un raccourci linguistique (en particulier en français, où les objets ont un genre), il faut éviter à tout prix de sombrer dans l'anthropomorphisme. En échangeant du texte avec un tel système ou agent conversationnel, vous ne dialoguez pas avec une entité, mais fournissez des données textuelles et recevez en retour un texte composé par un modèle qui « devine » statistiquement quels mots (ou *tokens*) il doit placer après celui qu'il vient d'afficher. En d'autres termes, il s'agit d'une prédiction de mots et de phrases qui ont une « probable » relation avec le texte que vous avez fourni, selon les données qui ont servi à l'entraînement. Rien de plus, même si le résultat est stupéfiant (et/ou inquiétant).

Ce qui occupe l'esprit et l'actualité en ce moment tourne autour des LLM ou *Large Language Model*. Les LLM sont un sous-ensemble du domaine de l'intelligence artificielle. Il s'agit d'un groupe d'algorithmes utilisant, entre autres, des réseaux de neurones de grande taille et un processus d'apprentissage complexe pour produire un texte par prédiction statistique. Un LLM n'est donc pas, à proprement parler, un réseau de neurones même s'il s'agit d'une composante du système. C'est l'architecture complète (structure, algorithmes, données d'entraînement, etc.) qui forme le modèle final.

Dans le cas de GPT (*Generative Pre-trained Transformer*), une architecture spécifique, et maintenant très populaire, est utilisée, faisant usage de *transformers*. Ce concept, inventé dès 2017, a remplacé une autre forme de modèle appelé RNN (*Recurrent Neural Networks*) et est à la base des modèles que vous pouvez voir maintenant fleurir de toutes parts.

Un modèle est donc un *design* qui sera ensuite entraîné en lui faisant ingurgiter des données... des tonnes et des tonnes de données. Plus le modèle est grand, plus il sera en mesure de fournir une importante cohérence et précision, mais plus il sera consommateur de mémoire et de ressources processeur. La taille du modèle est généralement définie par le nombre de ses paramètres. Ce terme provient directement du monde des statistiques et désigne des valeurs, internes au modèle, qui peuvent changer à mesure que son entraînement progresse. Vous pouvez voir cela comme le nombre de points de réglages possibles sur un moteur (débit de carburant, d'air, allumage, température, pression, richesse du mélange, etc.), mais ici, la quantité se chiffre en milliards. Entre 2 et 7 pour les petits modèles et de l'ordre de 200 à 500 pour les gros. On parle même de plusieurs billions (10^{12}) de paramètres pour GPT-4...

Pour ajuster ces paramètres dans la phase d'apprentissage, les données textuelles sont découpées en *tokens*, tout comme les textes et contextes textuels que vous lui soumettrez une fois le modèle entraîné, le fameux *prompt*. Un *token* est l'unité la plus petite d'un texte du point de vue du LLM. Il peut s'agir de mots, de morceaux de mots ou de groupes de caractères/symboles. Un outil en ligne d'OpenAI [1] permet de visuellement comprendre ce qu'est un *token*. Ainsi, la chaîne de caractères « Ceci est une longue phrase. » (avec son point) est décomposée en : « C », « ec », « i », « est », « une », « long », « ue », « phrase », « . » (notez les espaces au début de certains *tokens*).

Comme nous allons le voir plus loin, un modèle entraîné prend la forme d'un fichier de plusieurs Gio dont la taille

est proportionnelle à son nombre de paramètres. Bien entendu, un modèle de quelque 7, 13, 30 ou 65 Gio ne peut raisonnablement pas tout « savoir » ou stocker, mais sera tout de même en mesure d'apporter des réponses sur plus ou moins tout. Cette capacité à faire des prédictions sur des données non connues est nommée « inférence ». Il y a donc une part d'inconnu dans le processus et s'ajoute même une part aléatoire faisant partie inhérente du système.

Ceci provoque alors parfois un phénomène appelé « hallucination », où le modèle donne une réponse qui n'est aucune-ment justifiée par les données d'entraînement. Ce sont des réponses, souvent catégoriques, qui « sortent de nulle part » et sont un réel problème, sinon le plus important, selon de nombreux spécialistes du domaine des LLM.

Enfin, un LLM qui n'est donc, dans les grandes lignes, qu'un modèle statistique n'a aucune notion de morale ou d'éthique et est, de base, parfaitement capable de répondre des choses étonnantes, sinon totalement choquantes. Je m'abstiendrai ici de vous donner des exemples puisque vous serez en

– Utilisez votre propre agent conversationnel sur Pi4 –

mesure de « torturer » vous-même des modèles une fois l'article conclu. Il est donc nécessaire d'orienter le modèle vers un ensemble de principes qui nous paraissent à tous (normalement) éthiques et moraux. On parle alors d'alignements du modèle qui comme d'autres ajustements (*fine-tuning*) interviennent après l'entraînement et demandent généralement une intervention humaine, comme un processus d'évaluation des réponses.

Nous venons de couvrir ici un certain nombre de termes et de notions qui forment la base de ce qu'il faut savoir sur le domaine, Gardez à l'esprit cependant que tout ceci évolue de façon extrêmement rapide et que d'autres définitions, techniques et appellations vont apparaître à mesure que les recherches progressent dans les mois futurs. L'avenir nous dira ce vers quoi tout ceci nous conduira, mais en attendant, il est temps de passer à la pratique...

2. EN AVANT !

Avant toute chose, précisons que les développements autour de ces technologies (modèle, outils, scripts, formats, etc.) se font à une cadence infernale.



Tout ceci progresse à une vitesse faramineuse, ce qui est très excitant, mais implique forcément que les manipulations faites à un instant « t » ne seront peut être plus applicables strictement à l'identique la semaine suivante. Ceci est vrai sur SBC comme avec une configuration PC conséquente et vous devez être prêt à devoir ajuster et adapter, de-ci de-là, les manipulations que nous décrivons.

2.1 Préparer le SBC

Lorsqu'on parle d'exécuter un modèle, quelle que soit la plateforme utilisée, les ressources disponibles sont, par définition, ce qui formera le goulot d'étranglement qui tuera tout aspect interactif. Trois d'entre elles sont importantes et une est cruciale. Nous avons :

- l'espace de stockage disponible. Les modèles occupent une place conséquente, en particulier à l'échelle d'un système embarqué comme une carte Raspberry Pi ;
- la puissance de calcul, dont le manque se traduira simplement en temps d'exécution supplémentaire. Ce n'est donc pas, en principe, un point éliminatoire, mais simplement très problématique ;

Le Raspberry Pi 400 est, dans les grandes lignes, une Raspberry Pi 4 avec 4 Go de RAM intégrée sous la forme d'un clavier/ordinateur compact (AZERTY svp). Son usage de prédilection semble être celui d'un client léger, plus qu'une carte de développement pour l'embarqué, mais son véritable intérêt ici est la taille de la mémoire physique disponible.

À l'arrière du Pi 400, on retrouve exactement la même connectique que celle d'une Pi4. Avec, de gauche à droite, les GPIO, le support MMC/SD, les deux ports micro-HDMI, l'USB-C pour l'alimentation (et l'OTG), deux ports USB 3, un USB 2.0 et, finalement, l'interface Ethernet.

- la mémoire qui, elle, est fondamentale et éliminera littéralement et sans état d'âme la plupart des plateformes populaires actuellement disponibles (Pi3, Orange Pi Zero, etc.), ne laissant que celles à un prix outrageusement élevé (Pi4) ou à l'avenir incertain (Orange Pi 5 RK3588S, par exemple).

90 % des SBC ne sont tout simplement pas capables de faire fonctionner ce qui va suivre, car ils n'embarquent pas suffisamment de RAM. Une Raspberry Pi 4 n'ayant que 2 Gio de mémoire ne sera pas utilisable. Même les plus petits modèles LLM utilisables et dignes de ce nom font quelque 4 Gio qui, une fois chargés en mémoire occuperont ~6 Gio plus, bien entendu, la mémoire nécessaire pour le reste du code et des données, et l'OS.

Plus de 6 Gio ! Comment tout cela peut-il donc fonctionner sur une Raspberry Pi 400 avec seulement 4 Gio de RAM ? La réponse tient en un mot : le « swap ». Le *swap* ou espace d'échange est un mécanisme permettant au système de stocker des données qui semblent être en mémoire et il devient alors possible d'étendre la mémoire virtuelle au volume RAM physique plus le *swap*. Pour les applications, la mémoire RAM semble donc plus importante, mais le contrecoup est important, car la RAM est rapide, mais le stockage lent. Ce que vous gagnez en mémoire, vous le perdez en performances (et en espace de stockage).

Ceci nous amène à un autre point important. Nous avons déjà déterminé qu'une Pi 4 avec 4 Gio, une Pi 400 ou tout autre SBC avec 4 Gio de RAM était un minimum, mais voici donc que les performances du support de stockage deviennent critiques, car le système va sans cesse « swapper » des pages de mémoire. Dans

le cas des Pi 4 et consorts, nous pouvons utiliser deux supports de stockage pour le système, la SD/MMC ou l'USB. N'imaginez pas une seconde pouvoir « swapper » sur la SD/MMC, les performances seront catastrophiques. Le plus simple est de « booter » le système sur un support USB, plus performant (ou éventuellement démarrer le système sur SD/MMC et « swapper » sur USB, où se trouvent également les données à utiliser).

Nous prenons ici l'exemple de l'ordinateur Raspberry Pi 400 (équivalent au SBC Pi4 4 Gio), mais des alternatives possibles existent : Banana pi M5 4 Gio, NVidia Jetson Nano (4 Gio), Odroid C4 (4 Gio), Orange Pi 3 LTS (4 Gio), Orange Pi 5 (4, 8 ou 16 Gio + NVME SSD) ou encore RockPi 4C+ (4, 8 ou 16 Gio). Chaque carte étant différente, les configurations de *boot* et les performances du stockage le seront également... tout comme le prix.

Votre Pi4/Pi400 devra être préalablement configurée pour utiliser l'USB et démarrer



sans SD/MMC. Pour cela, vous pouvez tout simplement utiliser l'outil officiel *Raspberry Pi Imager* (Windows, Ubuntu, macOS). Après lancement, cliquez sur **Choisir l'OS**, **Misc utility images**, **Bootloader** et **USB Boot**. Sélectionnez ensuite le support de stockage cible (typiquement la SD dans son lecteur) et cliquez simplement **Écrire**. La SD/MMC est maintenant prête pour configurer l'EEPROM du SBC.

Si on n'aime pas utiliser un outil précompilé sur ses machines pour cause de paranoïa (totalement légitime), il est possible d'obtenir ces images autrement avec un peu d'effort et d'ingéniosité. Vous remarquerez sans doute que la page de téléchargement officielle [2] ne propose pas ce type d'images (pourquoi ?), mais nous avons accès aux sources du *Raspberry Pi Imaging Utility* [3]. Nous pouvons donc le reconstruire de toutes pièces, ce qui est un gros travail (dépendances, C++, Qt5, etc.), ou tout simplement aller voir ce qu'il fait.

Ainsi, dans le fichier `config.h`, nous découvrons l'URL `http://downloads.raspberrypi.org/os_list_imagingutility_v3.json` qui, quand consultée, nous remonte un JSON listant toutes les entrées du menu, dont :

```
{
  "name": "Bootloader",
  "description": "Restore the factory default settings and change boot priority",
  "icon": "https://downloads.raspberrypi.org/raspios_armhf/Raspberry_Pi_OS_(32-bit).png",
  "subitems_url": "https://downloads.raspberrypi.org/eeeprom_recovery.json"
},
```

Cette nouvelle URL, `https://downloads.raspberrypi.org/eeeprom_recovery.json`, nous donne quelque chose de similaire et nous redirige vers `https://github.com/raspberrypi/rpi-eeeprom/releases`, où il nous suffit donc de piocher :

- `rpi-boot-eeeprom-recovery-*-network.zip` : configurer l'EEPROM pour démarrer sur le réseau ;
- `rpi-boot-eeeprom-recovery-*-sd.zip` : sur le support SD/MMC ;
- `rpi-boot-eeeprom-recovery-*-usb.zip` : sur un support de stockage USB.

On pourra alors simplement décompresser le fichier ZIP et écrire l'image obtenue sur le support avec `dd`, `dcfldd` ou Etcher [4].

Quelle que soit la méthode choisie, on placera ensuite le support dans l'emplacement de la Pi et on alimentera le tout. La console série et/ou l'affichage via HDMI devrait vous afficher quelque chose comme :

```
SIG pieeprom.sig
41d638a8a67ac14600056aa0761a0e997eb582e890bfc7299db99097085c1e5c
1674059736
Reading EEPROM: 524288
Writing EEPROM
+++++*****
Verify BOOT EEPROM
Reading EEPROM: 524288
BOOT-EEPROM: UPDATED
```




Nous allons faire un usage intensif du support de stockage et avons donc besoin de performances supérieures à celles que l'on peut obtenir d'une carte MMC/SD. La solution consiste donc à utiliser un support USB comme une clé ou un SSD USB. L'effort pour coordonner la couleur du périphérique avec celle du Pi400 ne semble malheureusement pas avoir d'impact sur les vitesses de transfert.

Un écran ou une console série n'est cependant pas requis puisqu'un flashage réussi provoquera également un furieux clignotement de la LED intégrée. On pourra alors couper l'alimentation et retirer la SD/MMC qui ne sera plus nécessaire. Vous serez dès lors en mesure de préparer le support USB exactement de la même manière qu'une SD/MMC, avec *Raspberry Pi Imager*, un téléchargement et **dd**, ou via la méthode que nous avons présentée dans le numéro 45 [5] automatisant énormément de choses. Dans tous les cas, utilisez le système de plus léger possible (typiquement *Raspberry Pi OS Lite* en version 64 bits). La ligne de commande est suffisante pour ce nous avons à faire et ajouter une interface graphique ne ferait que consommer des ressources CPU et notre précieuse RAM.

À ce propos d'ailleurs, une fois le système démarré (et redémarré) et la taille de la partition racine adaptée automatiquement à celle du support, nous pouvons nous pencher sur le problème de mémoire. L'espace d'échange ou *swap* est ici configuré sous la forme d'un fichier (généralement, sur PC, c'est une partition). Ce fichier, **/var/swap**, est créé et configuré par l'intermédiaire de l'outil **dphys-swapfile**. Pour changer ses paramètres, nous commençons par désactiver le *swap* avec :

```
$ sudo dphys-swapfile swapoff
```

On peut ensuite éditer le fichier de configuration **/etc/dphys-swapfile** pour ajuster les tailles. Et je dis bien **les** tailles, car deux lignes sont concernées :

```
CONF_SWAPSIZE=4096
CONF_MAXSWAP=4096
```

La première configure la taille du *swap* (et donc celle du fichier) et la seconde la taille maximum utilisable. Ceci peut paraître redondant, mais **CONF_SWAPSIZE** pourrait être calculé automatiquement comme un multiple de la RAM présente, et ce avec un maximum de **CONF_MAXSWAP** qui est, par défaut de 2 Gio. Il est donc nécessaire de spécifier deux fois nos 4 Gio pour dépasser la limite par défaut.

Ceci fait, Il ne reste plus qu'à régénérer le fichier de *swap* et réactiver le *swapping* :

```
$ sudo dphys-swapfile setup
want /var/swap=4096MByte,
checking existing:
deleting wrong size file (104857600),
generating swapfile ... of 4096MBytes

$ sudo dphys-swapfile swapon
```


– Utilisez votre propre agent conversationnel sur Pi4 –

```
$ free -h
              total        utilisé        libre      partagé  tamp/cache  disponible
Mem:        3,7Gi      78Mi      3,1Gi      1,0Mi      580Mi      3,6Gi
Partition d'échange:    4,0Gi                      0B      4,0Gi
```

La commande **free** nous confirme la bonne marche de l'opération et nous avons donc maintenant, virtuellement, un peu moins de 8 Gio de mémoire à disposition. Notez que cette configuration persistera entre les redémarrages du système. Vous n'avez rien d'autre à faire.

Remarquez qu'il peut être tentant de réduire la mémoire allouée au processeur graphique de la Pi et c'est effectivement, en principe, un gain. Par défaut, ce sont 64 Mio qui sont alloués au GPU et donc autant indisponibles pour le système. Via **raspi-config** (**Performance Options** et **GPU Memory**), ou en éditant **/boot/config.txt** (**gpu_mem=**), nous pouvons descendre cette valeur au minimum, soit 16 Mio, et redémarrer. Étant donné les valeurs concernées et les futurs besoins, cependant, ce ne sont pas 48 Mio qui feront une grande différence...

2.2 Récupérer le modèle LLaMA 7B et le convertir

On pourrait penser que ce qui a déclenché le raz de marée « IA » de ces derniers mois est chatGPT et lui seul. Mais en réalité, c'est tout autant la réaction des concurrents d'OpenAI qui est à l'origine du phénomène. Parmi eux, nous avons Meta et en particulier LLaMA (*Large Language Model Meta AI*) dont la diffusion en février 2023, sous licence GNU GPL v3, a littéralement ensemencé des dizaines de projets dans la même veine. Un site comme *Hugging Face*, qui est récemment devenu une plateforme où plus ou moins toute la communauté LLM se rassemble, réunit de nombreux modèles en lien avec LLaMA et tout le monde semble prendre un malin plaisir à les nommer relativement à cet acronyme : Alpaca, Vigogne, Cabrita, Vicuna, etc. (il existe même une interface de dialogue web appelé « Serge »).

Nous allons partir ici sur le modèle initial LLaMA 7B, mais l'outil qui permettra son utilisation sur RPi un peu plus loin, *LLaMA.cpp*, est également capable d'en utiliser d'autres : Alpaca, GLT4All, Vigogne, Vicuna... Nous allons donc devoir récupérer le modèle puis le convertir en un format qui est utilisable avec *LLaMA.cpp*, ce qui nous permettra de mettre en œuvre les outils à utiliser et le sens des différentes manipulations qui pourraient s'avérer nécessaires dans le futur (qui à ce stade désigne plus ou moins

L'intérieur du Pi400 est un simple PCB sensiblement différent d'une Pi standard. On retrouve tous les composants habituels, étalés tout du long du bord du boîtier. Notez la présence du connecteur pour le clavier à droite ainsi que le contrôleur dédié. Retiré pour la photo, un dissipateur thermique relativement épais couvre normalement l'ensemble du circuit et permet une régulation de température très efficace et surtout fanless.



le mois prochain, étant donné les cadences de développement). Bien entendu, ce qui suit peut, mais ne doit pas, être fait sur le SBC, mais plutôt sur un PC GNU/Linux, une machine virtuelle, un conteneur ou dans WSL, de préférence avec 16 Go de RAM.

Pour récupérer LLaMA dans sa version « 7 milliards de paramètres » (7B), nous allons tricher et utiliser un projet initialement développé pour le faire fonctionner à l'aide d'un unique GPU (une bonne carte graphique) : **pyllama**. Avec la Pi, nous n'allons utiliser que le CPU, mais ce qui nous intéresse ici, ce sont les outils facilitant le téléchargement.

Nous commençons par télécharger le projet depuis GitHub et installons immédiatement les dépendances nécessaires au fonctionnement du code Python :

```
$ git clone https://github.com/juncongmo/pyllama.git
$ cd pyllama
$ pip install -r requirements.txt
```

Nous pouvons alors demander le téléchargement sans autre forme de procès avec :

```
$ python3 -m llama.download --model_size 7B
♥ Resume download is supported.
You can ctrl-c and rerun the program to resume the downloading
Downloading tokenizer...
✓ pyllama_data/tokenizer.model
✓ pyllama_data/tokenizer_checklist.chk
tokenizer.model: Réussi
Downloading 7B
downloading file to pyllama_data/7B/consolidated.00.pth
...please wait for a few minutes ...
Download watchdog: total file size increased
normally at speed 41.70MB/s
Download watchdog: total file size increased
normally at speed 48.73MB/s
[...]
✓ pyllama_data/7B/params.json
✓ pyllama_data/7B/checklist.chk
Checking checksums
Download watchdog: total file size increased normally at speed 3.35MB/s
consolidated.00.pth: Réussi
params.json: Réussi
```

--model_size nous permet de spécifier la taille du modèle en nombre de paramètres. LLaMA existe en quatre déclinaisons, 7B, 13B, 30B et 65B, mais 7B est le seul que nous pourrions faire fonctionner sur un SBC aussi modeste. Après un temps dépendant surtout de votre connexion internet, vous devez obtenir les données du modèle dans :

```
$ ls -F pyllama_data/
7B/ tokenizer_checklist.chk tokenizer.model
```

Ceci est inutilisable dans l'état. Nous devons tout d'abord le convertir au format GGML FP16 puis « quantifier » (*quantize*) le modèle. Sans trop entrer dans le détail, la quantification, ou *quantization* en anglais, nous permet de réduire la précision afin d'obtenir un modèle plus

– Utilisez votre propre agent conversationnel sur Pi4 –

compact et plus facile à utiliser avec certains matériels (sans GPU). Ici, nous avons des valeurs internes sous forme de nombres en virgule flottante de 16 bits (FP16) et voulons obtenir quelque chose de représenté par 4 bits pour une utilisation avec un CPU (et non un GPU bien plus l'aise avec la virgule flottante).

Heureusement pour nous, *LLaMA.cpp* dispose du nécessaire pour cette double conversion. Nous allons donc devoir compiler une première fois ce programme, sur PC, pour préparer les données qui seront ensuite utilisées par le *LLaMA.cpp* que nous utiliserons avec le SBC. Commençons donc par récupérer les sources et compiler les binaires localement :

```
$ cd ..
$ git clone https://github.com/gggerganov/llama.cpp.git
$ cd llama.cpp
$ make
I llama.cpp build info:
I UNAME_S: Linux
I UNAME_P: unknown
[...]
```

Grâce à cela, nous obtenons les programmes **main**, **quantize**, **perplexity** et **embedding** et nous enchaînons sur l'installation des modules Python nécessaires (partiellement déjà couverts par les dépendances de **pyllama**) et la copie du modèle LLaMA 7B :

```
$ pip3 install torch numpy sentencepiece

$ cd models/
$ cp -r ../../pyllama/pyllama_data/* .
$ cd ..
```

On peut ensuite commencer par convertir le modèle au format GGML FP16 :

```
$ python3 convert-ptb-to-ggml.py models/7B/ 1
{'dim': 4096, 'multiple_of': 256, 'n_heads': 32,
 'n_layers': 32, 'norm_eps': 1e-06, 'vocab_size': -1}
Namespace(dir_model='models/7B/', ftype=1, vocab_only=0)
n_parts = 1
Processing part 1 of 1
[...]
```



Cette FriendlyARM NanoPi M3 pourrait sembler être un excellent choix pour nos expérimentations en raison de son SoC Samsung S5P6818, fournissant huit cœurs ARM Cortex-A53 à 1,4 GHz. Malheureusement, cette carte ne dispose que de 1 Gio de RAM DDR3 et, malgré la puissance du CPU, cette carence en mémoire est éliminatoire pour jouer avec des LLM.


```
Processing variable: layers.31.feed_forward.w3.weight
with shape: (11008, 4096) and type: torch.float16
Processing variable: layers.31.attention_norm.weight
with shape: (4096,) and type: torch.float16
Converting to float32
Processing variable: layers.31.ffn_norm.weight
with shape: (4096,) and type: torch.float16
Converting to float32
Done. Output file: models/7B//ggml-model-f16.bin

$ ls -lh models/7B/
total 26G
-rw-r--r-- 1 denis denis 100  6 avril 12:12 checklist.chk
-rw-r--r-- 1 denis denis 13G  6 avril 12:12 consolidated.00.pth
-rw-r--r-- 1 denis denis 13G  6 avril 12:16 ggml-model-f16.bin
-rw-r--r-- 1 denis denis 101  6 avril 12:12 params.json
```

Puis procéder à la *quantization* grâce au binaire que nous venons d'obtenir :

```
$ ./quantize ./models/7B/ggml-model-f16.bin ./models/7B/ggml-model-q4_0.bin 2
llama_model_quantize_internal: loading model from './models/7B/ggml-model-f16.bin'
llama_model_quantize_internal: n_vocab = 32000
llama_model_quantize_internal: n_ctx  = 512
llama_model_quantize_internal: n_embd = 4096
llama_model_quantize_internal: n_mult = 256
llama_model_quantize_internal: n_head = 32
llama_model_quantize_internal: n_layer = 32
llama_model_quantize_internal: f16    = 1
[...]
llama_model_quantize_internal: model size = 25705.02 MB
llama_model_quantize_internal: quant size = 4017.27 MB
llama_model_quantize_internal: hist: 0.000 0.022 0.019 0.033 0.053
0.078 0.104 0.125 0.134 0.125 0.104 0.078 0.053 0.033 0.019 0.022

main: quantize time = 171772.61 ms
main:   total time = 171772.61 ms
```

Nous pouvons éventuellement tester le résultat, le modèle maintenant disponible sous la forme du fichier **ggml-model-q4_0.bin** (4 Gio), avec :

```
./main -m ./models/7B/ggml-model-q4_0.bin -n 128
```

Mais nous pouvons aussi et surtout transférer **ggml-model-q4_0.bin** sur la Pi400, récupérer les sources de *LLaMA.cpp* et les compiler, pour enfin exécuter cette commande directement sur le SBC :

```
$ ./main -m ./models/ggml-model-q4_0.bin \
-n 128 -p "What are the 5 biggest cities in france?"
```


– Utilisez votre propre agent conversationnel sur Pi4 –

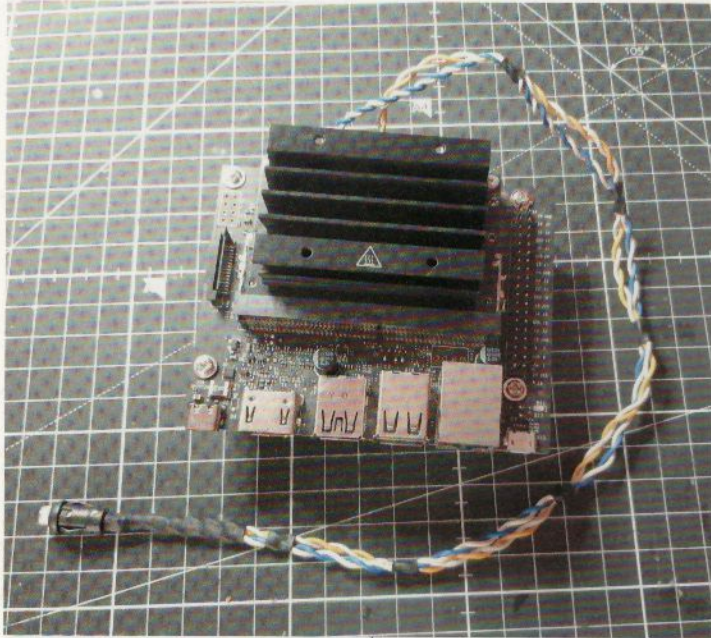
```
main: seed = 1680788739
llama_model_load: loading model from 'models/ggml-model-q4_0.bin' - please wait ...
llama_model_load: n_vocab = 32000
llama_model_load: n_ctx = 512
llama_model_load: n_embd = 4096
llama_model_load: n_mult = 256
llama_model_load: n_head = 32
llama_model_load: n_layer = 32
llama_model_load: n_rot = 128
llama_model_load: f16 = 2
llama_model_load: n_ff = 11008
llama_model_load: n_parts = 1
llama_model_load: type = 1
llama_model_load: ggml map size = 4017.70 MB
llama_model_load: ggml ctx size = 81.25 KB
llama_model_load: mem required = 5809.78 MB (+ 1026.00 MB per state)
llama_model_load: loading tensors from 'models/ggml-model-q4_0.bin'
llama_model_load: model size = 4017.27 MB / num tensors = 291
llama_init_from_file: kv self size = 256.00 MB

system_info: n_threads = 4 / 4 | AVX = 0 | AVX2 = 0 | AVX512 = 0
| FMA = 0 | NEON = 1 | ARM_FMA = 1 | F16C = 0 | FP16_VA = 0 |
WASM_SIMD = 0 | BLAS = 0 | SSE3 = 0 | VSX = 0 |
sampling: temp = 0.800000, top_k = 40, top_p = 0.950000,
repeat_last_n = 64, repeat_penalty = 1.100000
generate: n_ctx = 512, n_batch = 8, n_predict = 128, n_keep = 0

What are the 5 biggest cities in france?
Best Answer: Paris (7,6 million) Lyon (4,2 million), Marseille (600.000),
Lille (234.000) and Strasbourg (219.000). You can find the population of
the biggest cities here : http://fr.wikipedia.org/wiki/%C3%89tat\_f...
Because it is the capital city, Paris would be the largest French urban
area, but Lyon has a higher density than Paris so it's really the second
most populated urban area in France

llama_print_timings:          load time =
53153.42 ms
llama_print_timings:          sample time =
423.90 ms / 128 runs    ( 3.31 ms per run)
llama_print_timings: prompt eval time =
119529.07 ms / 12 tokens ( 9960.76 ms per token)
llama_print_timings:          eval time =
5859784.01 ms / 127 runs  (46140.03 ms per run)
llama_print_timings:        total time =
5983047.54 ms
```

5983047 millisecondes, soit 5983 secondes ou un peu moins de 100 minutes. C'est très long, mais ceci vient principalement, dans le présent essai, d'une lenteur excessive de la clé USB SanDisk Ultra de 32 Gio servant de système de fichiers racine. Un petit test rapide (avec **sysbench fileio prepare** puis **sysbench fileio --file-test-mode=rndrw run**) nous montre, effectivement, des résultats pitoyables :



La carte NVIDIA Jetson Nano 4 GB est une option possible pour les solutions que nous utilisons ici, mais pas davantage qu'un Pi400 ou une Pi4 avec 4 Gio de RAM. En effet, LLaMA.cpp est dédié à une utilisation purement « processeur » et ne sera pas capable d'exploiter le GPU NVIDIA Maxwell à 128 cœurs pour exécuter le modèle LLaMA ou autre.

```
File operations:
  reads/s:      313.49
  writes/s:     208.99
  fsyncs/s:     677.32
Throughput:
  read, MiB/s:  4.90
  written, MiB/s: 3.27
```

À titre de comparaison, voici ce qu'on obtient sur un Ryzen 5 5500U avec un disque Crucial M.2 PCIe NVMe 1 To :

```
File operations:
  reads/s:      16701.88
  writes/s:     11134.59
  fsyncs/s:     35632.09
Throughput:
  read, MiB/s:  260.97
  written, MiB/s: 173.98
```

Ou encore, la même chose sur un modeste *netbook* Samsung NB30 (Atom N450) avec un SSD A400 d'entrée de gamme de chez Kingston :

```
File operations:
  reads/s:      1171.02
  writes/s:     780.68
  fsyncs/s:     2499.37
Throughput:
  read, MiB/s:  18.30
  written, MiB/s: 12.20
```

Environ 3 ou 4 Mio/s en lecture et/ou en écriture passe encore pour des copies ou des chargements de fichiers, mais lorsqu'il s'agit de *swap*, c'est une catastrophe. Le réel problème ici est donc le *swap*. D'autres utilisateurs ayant la chance de posséder une Pi4 8 Gio affichent des performances bien supérieures, de l'ordre d'une dizaine de secondes par *token* affiché. Mais pour l'heure, cet essai est plus ou moins un échec.

Peut-être avons-nous eu les yeux plus grands que le ventre. Essayons donc quelque chose de plus léger, comme un modèle BLOOM réputé beaucoup plus performant. Pour cela, nous n'allons pas utiliser LLaMA.cpp mais *bloomz.cpp* [6], une déclinaison adaptée aux modèles BLOOM-like :

```
$ cd ..
$ git clone https://github.com/NouamaneTazi/bloomz.cpp
$ cd bloomz.cpp
$ make
```


– Utilisez votre propre agent conversationnel sur Pi4 –

Normalement, nous devrions procéder comme avec *LLaMA.cpp*, récupérer un modèle, convertir le format et « quantizer » les valeurs. Mais heureusement pour nous, *Hugging Face* propose des modèles déjà parfaitement adaptés [7], comme celui de Lucain Pouget (Wauplin). Nous récupérons donc directement le fichier **ggml-model-bloomz-3b-f16-q4_0.bin** [8] et le plaçons dans un sous-répertoire **models** des sources de *bloomz.cpp* sur la Pi. Et là :

```
$ ./main -m models/ggml-model-bloomz-3b-f16-q4_0.bin \
-t 4 -n 256 -p 'Translate "I am going to bed." in French:'
main: seed = 1680968865
bloom_model_load: loading model from
'models/ggml-model-bloomz-3b-f16-q4_0.bin' - please wait ...
bloom_model_load: n_vocab = 250880
bloom_model_load: n_ctx = 512
bloom_model_load: n_embd = 2560
bloom_model_load: n_mult = 1
bloom_model_load: n_head = 32
bloom_model_load: n_layer = 30
bloom_model_load: f16 = 2
bloom_model_load: n_ff = 10240
bloom_model_load: n_parts = 1
bloom_model_load: ggml ctx size = 2476.68 MB
bloom_model_load: memory_size = 300.00 MB, n_mem = 15360
bloom_model_load: loading model part 1/1
from 'models/ggml-model-bloomz-3b-f16-q4_0.bin'
bloom_model_load: ..... done
bloom_model_load: model size = 2175.72 MB / num tensors = 366

main: prompt: 'Translate "I am going to bed." in French:'
main: number of tokens in prompt = 10
153772 -> 'Translate'
29000 -> ' "I'
912 -> ' am'
7963 -> ' going'
427 -> ' to'
32262 -> ' bed'
503 -> ' .'
5 -> ' "'
361 -> ' in'
196427 -> ' French:'

sampling parameters: temp = 0.800000, top_k = 40,
top_p = 0.950000, repeat_last_n = 64, repeat_penalty = 1.300000

Translate "I am going to bed." in French: Je vais dormir.</s> [end of text]

main: mem per token = 15381084 bytes
main: load time = 35868.06 ms
main: sample time = 327.90 ms
main: predict time = 6936.07 ms / 495.43 ms per token
main: total time = 45306.77 ms
```


45 secondes sont beaucoup mieux et la traduction est juste ! C'est, bien entendu, parce que le modèle fait seulement un peu plus de 2 Gio, auxquels s'ajoutent 14 Mio par *token* (les nôtres et ceux de la réponse). Le système n'a donc pas à *swapper*, ce qui explique les performances bien supérieures. Mais attention, la taille du modèle détermine sa précision et les réponses peuvent parfois être assez amusantes. Exemple : « *what is the capital of Slovakia?* » -> « *Vienna* ». La réponse est « *Bratislava* » et le modèle `ggml-model-bloomz-7b1-f16-q4_0.bin` répond correctement, mais il fait 5 Gio et se comportera peu ou prou comme le modèle LLaMA.

Contrairement aux modèles LLaMA ou basés sur LLaMA, il existe une plus grande diversité en termes de nombre de paramètres (et donc de taille). Explorez *Hugging Face* et flirtez avec les limites matérielles et vous trouverez certainement le juste milieu entre vitesse et précision. *LLaMA.cpp* et *bloomz.cpp* (ainsi que *Aipaca.cpp* [9]), vous permettront de tester, explorer et comprendre les mécanismes, les

formats et les contraintes. Il est fort possible qu'entre le moment de la rédaction du présent article et celui où vous le lirez, d'autres solutions, projets et modèles aient vu le jour. Mais vous devez avoir maintenant les bases pour les comprendre et les tester...

3. QUÊTE DE PERFORMANCES ET CONCLUSION

Il est possible de légèrement gagner en performance via différentes approches. Utiliser un système faisant fonctionner moins de services est un point de départ intéressant,

Voici une très bonne option pour l'exécution locale d'un LLM. Avec ses 16 Gio de RAM, nous serions en mesure d'utiliser des modèles plus précis (comme LLaMA 13B) sans avoir recourt au swap. Ce à quoi s'ajoute une interface PCIe permettant d'utiliser un SSD NVMe, tout en gardant un prix presque raisonnable (par rapport à ceux actuellement constatés sur les Raspberry Pi). Par contre, même pour un budget de quelque 130 €, il faut tout de même se poser la question : que faire de la carte une fois qu'on aura fini de « jouer » ?

AliExpress Shenzhen Xunlong Software CO., Limit... 96.8% Evaluations positives 21940 Abonnés

Page d'accueil Catégories Articles en Promos Meilleures ventes Nouveautés Avis sur le vendeur

Orange Pi 5 16GB RAM

Orange Pi 5 16 GO RK3588S PCIe Module WiFi Externe + BT, SSD Gigabit Ethernet Ordinateur Monocarte, Exécutez Android OS Debian

★★★★★ 4.7 103 Avis + 900 Commandes

€ 135.83

Prix TTC

Payez en 4 fois sans frais

Couleur: Available in Stock

Available in Stock

Quantité: 1 391 unités disponibles

Livré vers Colmar, Haut-Rhin, France

Livraison: € 11.01

De China à Colmar via AliExpress Standard Shipping

Date de livraison estimée: 04 mai.

Acheter maintenant Ajouter au panier 1691

tout comme le fait de structurer le système de façon à *swapper* sur une partition et non dans un fichier. On peut également envisager d'opter pour un autre système, dont l'installation par défaut sera plus économe, comme FreeBSD (voir l'article en rapport dans ce numéro ou celui dans le précédent numéro [10] ou dans le 41 avec NetBSD [11]).

Tout ceci ne rendra cependant pas pour autant l'ensemble réellement utilisable, juste très sensiblement moins « poussif ». Mais le réel objectif n'est pas d'avoir une longue et joyeuse conversation avec un modèle, mais simplement de pouvoir ramener le concept à quelque chose de tangible et de bien comprendre, en pratique, qu'il n'y a non seulement pas de magie, mais également en quoi ce genre de système est intimement lié aux performances matérielles.

Ce qui est également absolument stupéfiant c'est la quantité d'informations qui **semblent** stockées dans un simple fichier de 4 Gio. Ce n'est, bien entendu pas réel, mais le modèle **apparaît** pouvoir répondre de façon cohérente à tout et n'importe quoi. De la même manière qu'un agent

conversationnel **semble** avoir une réflexion, les informations **semblent** présentes dans les 4 Gio, mais ce n'est pas le cas, ce sont juste des probabilités.

Quoi qu'il en soit et même si les résultats sur un SBC sont assez décevants (mais sans surprise), il est important de retirer la mystique de ce qui est en train de se passer globalement et, pour cela, rien de mieux que d'expérimenter soi-même. Si vous souhaitez poursuivre votre exploration, dans un environnement plus riche en ressources et moins spartiate que de l'embarqué, je vous recommande fortement les récents *streams* Twitch d'Yves Rougy (Yorzian) [12], réunis dans une *playlist* YouTube dédiée [13]. **DB**

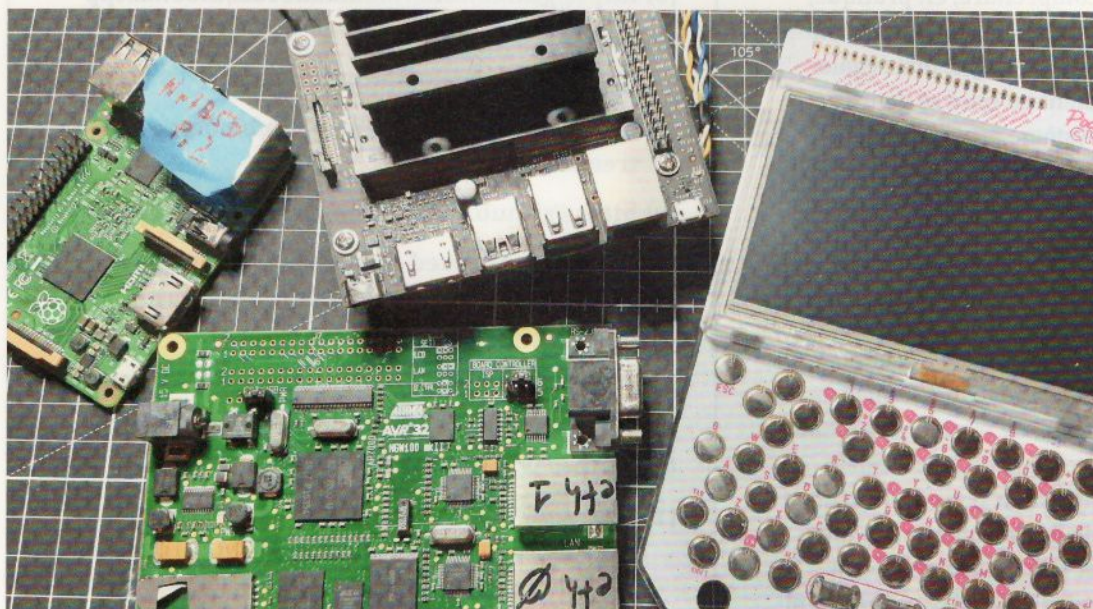
RÉFÉRENCES

- [1] <https://platform.openai.com/tokenizer>
- [2] <https://www.raspberrypi.com/software/>
- [3] <https://github.com/raspberrypi/rpi-imager>
- [4] <https://www.balena.io/etcher>
- [5] <https://connect.ed-diamond.com/hackable/hk-045>
- [6] <https://github.com/NouamaneTazi/bloomz.cpp>
- [7] <https://huggingface.co/models?other=bloom&other=ggml>
- [8] <https://huggingface.co/Wauplin/bloomz-3b.cpp/tree/main>
- [9] <https://github.com/antimatter15/alpaca.cpp>
- [10] <https://connect.ed-diamond.com/hackable/hk-047/freebsd-pour-l-embarque-le-cas-orange-pi-zero>
- [11] <https://connect.ed-diamond.com/hackable/hk-041/raspberry-pi-et-pourquoi-pas-netsbsd>
- [12] <https://www.twitch.tv/yorzian>
- [13] <https://www.youtube.com/playlist?list=PLPKSp0MgWyBwij1suqHmQQ79LnH1tAeAr>

INSTALLER TROIS DÉMONS SUR UNE RASPBERRY PI 4

Denis Bodor

Nous sommes début 2023 et les RPi4 sont toujours des choses rares et précieuses. De ce fait, les utiliser avec parcimonie, et uniquement pour des projets qui nécessitent réellement les ressources proposées, est devenu un peu le nerf de la guerre. Mais si vous êtes comme moi, ceci est une réelle problématique, car à force de se dire à répétition « non, je réserve cette carte pour quelque chose de plus sérieux », elle finit par ne jamais être utilisée. Et ça, c'est encore pire ! Pourquoi ne pas alors décider de partager ces ressources et d'installer plusieurs OS, très différents, sur un même SBC ?



Avant toute chose, je pense qu'il est important de préciser que ce qui va suivre est :

- 1 - loin d'être l'approche optimale pour ce type de choses et nous verrons, peut-être, dans un futur article comment faire cela en utilisant un *boot* en réseau ;
- 2 - surtout une belle opportunité de mettre son nez dans le mécanisme de démarrage d'une Pi et des systèmes que nous allons utiliser ;
- 3 - une piètre excuse pour justifier du fait de ne pas vouloir jongler avec les cartes MMC (microSD), car oui, rien ne vous empêche d'installer autant de systèmes que vous le souhaitez, chacun ayant son propre support de stockage amovible (moins l'usure de l'emplacement).

Ce qui est intéressant ici, c'est de partager l'espace d'une carte MMC conséquente (quelque 128 Gio) entre plusieurs systèmes et de pouvoir démarrer l'un ou l'autre en fonction des besoins. Besoins qui se résument, dans le cas présent, à pouvoir tester des codes portables d'un OS à l'autre, le tout sur une architecture ARM 64 bits (alias aarch64). Là encore, une autre

approche, celle de l'émulation avec QEMU, pourrait être une option, mais lorsqu'il s'agit de codes visant à utiliser des périphériques USB spécifiques, rien ne remplace le matériel bien physique et bien tangible.

Nous avons déjà couvert, par le passé, la structure d'un système GNU/Linux et la façon dont le système, sur une Raspberry Pi, démarre. Nous avons même d'ailleurs personnalisé ce mécanisme dans le numéro 45 [1] afin d'avoir directement un environnement prêt à servir. Nous allons donc laisser ce système de côté pour nous intéresser à quelque chose de (malheureusement) moins courant en se posant la question : combien d'anges peuvent danser sur la pointe d'une framboise ? Alors, des anges je n'en sais rien, mais au minimum trois démons (*daemon*), ça, c'est certain ! En effet, nous allons nous pencher sur les dignes et véritables héritiers du vénérable 4.BSD-Lite. Nous laisserons tout de même une petite porte pour le manchot, sous la forme d'une partition que nous garderons au chaud pour plus tard, si le cœur nous en dit.

1. PRÉPARATION ET PRÉREQUIS

L'objectif de l'opération est de disposer de FreeBSD, OpenBSD et NetBSD sur une architecture ARM 64 bits et de pouvoir, à souhait, redémarrer le SBC sur l'un ou l'autre système. Sur PC, ceci serait presque un jeu d'enfant, puisque chaque processus d'installation de chaque OS est parfaitement capable de gérer le matériel et de se débrouiller tout seul (ou presque). Dans l'embarqué, cependant, les choses sont très différentes. Différentes du PC bien sûr, mais également différentes d'un système à l'autre, tout en ayant quelques points communs.

Sur les trois systèmes, deux se présentent exactement comme « Raspberry Pi OS » (qui n'est pas un OS puisqu'il s'agit d'une distribution Debian modifiée) sous la forme d'une image qu'on utilisera pour initialiser une carte SD et « booter » directement le SBC : FreeBSD et NetBSD. OpenBSD, en revanche utilise une autre approche consistant à démarrer un système minimal (*miniroot* composé d'un noyau et d'un *RAMDISK*) et à lancer un installateur, exactement comme cela se passe sur les nombreuses autres architectures supportées.



Voici la fameuse Raspberry Pi 4. Denrée rare s'il en est en ce moment et justement, c'est précisément pour cette raison qu'elle n'a presque jamais été utilisée pour un projet... Ah, la psychologie humaine !

Étant donné les systèmes de fichiers, ou FS (pour *File System*) utilisés et le manque de compatibilité côté GNU/Linux, le système gérant le partitionnement et les manipulations sera un FreeBSD 13.1 installé sur une machine amd64 (AMD Ryzen 5 5500U) relativement fournie en ressources (CPU et mémoire), bien que ceci n'a pas grande importance ici, puisque nous n'allons (presque) rien recompiler ou reconstruire. Il s'agira principalement de manipulations de partitions et de modifications de fichiers, de-ci de-là. Parlant de partitions, le support utilisé sera une microSD de 128 Go de marque Sandisk (microSDXC SanDisk Extreme UHS-I), largement suffisante pour nous permettre de donner à chaque système au moins 32 Gio d'espace de stockage pour de futurs travaux, tout en ayant des performances acceptables. Notez que c'est précisément avec ce genre de manipulations, dont des copies de données assez volumineuses, qu'on peut se rendre compte que toutes les cartes microSD ne se valent pas, et même que certaines sont purement et simplement d'horribles contrefaçons aux capacités fantaisistes. Un conseil, achetez vos supports de stockage auprès de revendeurs réputés et toujours des produits de marques, sur lesquels vous n'avez aucun doute quant à l'origine.

Nous allons partitionner le support avec un outil appelé **gdisk**, installable très facilement sous FreeBSD avec `sudo pkg install gdisk`. Il s'agit

d'un utilitaire interactif permettant de manipuler les tables de partitions GUID, alias GPT (*GUID Partition Table*). Ceci par opposition à l'ancien type de système de partition, dit MBR (pour *Master Boot Record*), aux nombreuses limitations. Nous ne rentrerons ici pas en détail sur ce système puisqu'il n'existe que sur les anciennes architectures de type PC/BIOS et ne nous concerne absolument pas dans le cas présent. Notez cependant une différence d'appellation importante entre les systèmes héritiers de BSD et le reste du monde (PC, Windows, GNU/Linux, etc.). Ce que l'on nomme généralement une « partition » est, pour un système *BSD, une tranche ou *slice*, un morceau de disque. Ce *slice* peut ensuite être divisé en un ou plusieurs *labels* permettant de créer des systèmes de fichiers et rendre chaque partie accessible au système (dont, raisonnablement, au minimum un système de fichiers racine *rootfs* et une zone de *swap*). Nous avons donc une sous-division des partitions qui n'existe pas, la plupart du temps, avec des systèmes comme Windows ou GNU/Linux.

gdisk est également disponible sous GNU/Linux et Debian/Devuan en particulier,

Démonologie appliquée :) :) :)

– Installer trois démons sur une Raspberry Pi 4 –

vous pouvez normalement transposer ce qui va suivre sur une telle plateforme, aussi bien pour la création des partitions/*slices* GPT et la copie **brute** des images. Je ne couvrirai cependant pas le montage des images de disque sur tous les systèmes (**mdconfig** avec FreeBSD, **vnconfig** avec OpenBSD, **vndconfig** avec NetBSD et **dmsetup** avec GNU/Linux) en m'en tenant simplement à FreeBSD. Libre à vous, là encore, de transposer tout cela, ça ne devrait pas être bien compliqué. La manipulation (montage, redimensionnement, etc.) de systèmes de fichiers UFS/FFS est en revanche un vrai problème sous GNU/Linux, d'où... FreeBSD.

Il est temps maintenant de nous attaquer au sujet, et à la microSD en particulier, en effaçant brutalement son contenu et en débutant son partitionnement. La première opération consistera ici à sauvagement écrire une tripotée de 0 sur les 3 premiers Mio du support (notez au passage que les « mébiotets », « kibiots » et autres « tébiots » sont des désignations qui m'insupportent totalement, je m'en tiendrai donc à « Mio », « Kio » et « Tio », m'évitant d'avoir l'impression d'écrire un article sur des Pokémon aux noms les plus stupides) :

```
# dd if=/dev/zero of=/dev/da0 bs=1m count=3
3+0 records in
3+0 records out
3145728 bytes transferred in 0.376988 secs (8344368 bytes/sec)
```

Le symbole # en guise de *prompt* signifie ici que cette commande est exécutée en *root* et toutes celles qui vont suivre également. **/dev/da0** désigne le pseudo-périphérique représentant le lecteur de microSD USB dans lequel le support de 128 Gio est inséré. Sous GNU/Linux, vous auriez quelque chose comme **/dev/sdd**, par exemple, et je souligne ici que vous devez être prudent et systématiquement jeter un œil à la sortie d'un **dmesg** après chaque connexion du périphérique pour être **absolument certain** de viser le bon disque.

Une fois cette étape de violence gratuite passée, nous pouvons sortir **gdisk /dev/da0** et partitionner tout cela. Comme nous avons saccagé la structure du disque, l'outil va nous signaler le problème et nous proposer de partir sur des bases saines :

```
Found valid MBR and corrupt GPT. Which do you want to use? (Using the
GPT MAY permit recovery of GPT data.)
1 - MBR
2 - GPT
3 - Create blank GPT

Your answer: 1
```

Nous avons à présent tout loisir de découper à souhait. Commençons donc par la base puisqu'il s'agit d'une Raspberry Pi. À savoir une zone FAT destinée à contenir les éléments, propriétaires et *open source*, qui permettent de démarrer le SBC. Comme nous allons le voir plus loin, les choses vont sensiblement se compliquer, car chaque système est construit sur une base distincte et surtout un ensemble de paramètres (**config.txt** et *Device Tree Blob*) non homogène. Autrement dit, nous allons devoir jongler avec le contenu de cette partition FAT pour pouvoir démarrer l'un ou l'autre OS. Ce qui implique de disposer de suffisamment de place pour stocker toutes les versions. Prévoyons large et partons sur quelques 300 Mio :


```

Command (? for help): n
Partition number (1-128, default 1):
First sector (34-249737182, default = 2048) or {+}size{KMGTP}:
Last sector (2048-249737182, default = 249735167) or {+}size{KMGTP}: +300M
Current type is A503 (FreeBSD UFS)
Hex code or GUID (L to show codes, Enter = A503): EF00
Changed type of partition to 'EFI system partition'

```

Le support de stockage est excessivement important, non seulement en termes de performance, mais également de simple fonctionnement, de base. Les soi-disant cartes microSD d'une capacité impressionnantes (de 64 Gio à 1 Tio) à prix défiant toute concurrence sont, dans 95 % des cas, des arnaques.

La commande utilisée est **n** pour créer une nouvelle partition. Les informations entre parenthèses sont les valeurs par défaut proposées, qui seront automatiquement utilisées en validant simplement la demande. Ainsi le numéro de partition sera naturellement **1** et le premier secteur utilisé est suggéré en fonction du contenu déjà présent (ici, rien). *Last sector* nous permet de spécifier le secteur de fin de la partition, mais comme personne n'aime réfléchir en termes de secteurs, une syntaxe plus aisée est disponible. Ici, nous spécifions une taille de 300 Mio à partir du début de la partition (c'est le sens du signe **+** qui précède la valeur). Les unités de mesure utilisables sont **K** pour Kio, **M** pour Mio, **G** pour Gio, **T** pour Tio et enfin **P** pour des Pio, ou bien sûr, une position en nombre de secteurs, sans unité.

Vient ensuite le choix du type de partition, ou plus exactement son identifiant sous la forme d'une valeur hexadécimale sur 4 positions. Là, vous pouvez entrer la valeur directement ou utiliser « **L** » pour lister toutes celles

connues (et standardisées), mais aussi procéder à une recherche en spécifiant une chaîne de caractères et en validant. Le résultat de la recherche s'affiche et vous n'avez alors qu'à recopier la valeur. Ici, cette partition est identifiée par **EF00**, c'est celle destinée à stocker les données EFI (*Extensible Firmware Interface*). Nous verrons cela un peu plus loin, mais retenez bien que l'identifiant d'une partition GPT (ou le type en MBR) est quelque chose de distinct d'un système de fichiers. Notre partition EFI sera initialisée avec un système de fichiers FAT, mais pourrait l'être avec de l'UFS ou



Démonologie appliquée :) :) :)

– Installer trois démons sur une Raspberry Pi 4 –

de l'ext4. Et inversement, on peut également vouloir avoir un système de fichiers FAT sur une partition de type **0C00** « *Microsoft basic data* ». N'importe quelle combinaison fonctionne, du moment qu'un système est capable de la gérer. Ceci signifie également que des tutoriels en ligne titrés « *Comment convertir une partition GTP en NTFS* » sont aussi une parfaite démonstration que leurs auteurs ne maîtrisent absolument pas leur sujet...

Nous enchaînons dans la même veine et avec la même série de commandes et paramètres pour créer :

- une partition de 30 Gio de type **A503** *FreeBSD UFS* (oui, le nom est trompeur, mais souvent le type est également un indice du FS potentiellement présent) ;
- une partition de 30 Gio de type **A902** *NetBSD FFS* ;
- une partition de 30 Gio de type **A600** *OpenBSD disklabel* ;
- une partition du reste (environ 29 Gio en validant simplement la valeur par défaut) de type **8300** *Linux filesystem*.

Vous pouvez ensuite lister les partitions en présence avec **p** :

```
Command (? for help): p
Disk /dev/da0: 249737216 sectors, 119.1 GiB
Sector size (logical): 512 bytes
Disk identifier (GUID): 4E0C0C4C-C913-4996-A8E3-862703B97C9B
Partition table holds up to 128 entries
Main partition table begins at sector 2 and ends at sector 33
First usable sector is 34, last usable sector is 249737182
Partitions will be aligned on 2048-sector boundaries
Total free space is 4029 sectors (2.0 MiB)
```

Number	Start (sector)	End (sector)	Size	Code	Name
1	2048	616447	300.0 MiB	EF00	EFI system partition
2	616448	63531007	30.0 GiB	A503	FreeBSD UFS
3	63531008	126445567	30.0 GiB	A902	NetBSD FFS
4	126445568	189360127	30.0 GiB	A600	OpenBSD disklabel
5	189360128	249735167	28.8 GiB	8300	Linux filesystem

Notez, dans la dernière colonne, que le nom de la partition (son « label ») a été automatiquement défini en fonction du type précédemment choisi. En général, ceci n'a que peu d'importance, mais tantôt un système ou outil se base sur cette information pour agir. Nous allons donc changer deux de ces noms pour nous éviter des problèmes par la suite (avec NetBSD). La partition 1 va alors s'appeler « EFI » et la 3 « netbsd-root » et, pour cela, nous utilisons la commande **c** :

```
Command (? for help): c
Partition number (1-5): 1
Enter name: EFI

Command (? for help): c
Partition number (1-5): 3
Enter name: netbsd-root
```


Voici notre partitionnement terminé, il ne nous reste plus qu'à l'écrire sur le support avec **w** :

```
Command (? for help): w
```

```
Final checks complete. About to write GPT data. THIS WILL OVERWRITE EXISTING
PARTITIONS!!
```

```
Do you want to proceed? (Y/N): y
```

```
OK; writing new GUID partition table (GPT) to /dev/da0.
```

```
Warning: The kernel may continue to use old or deleted partitions.
You should reboot or remove the drive.
```

```
The operation has completed successfully.
```

Comme le précise le message, il est de bon ton de déconnecter le lecteur et le reconnecter, pour s'assurer que le noyau « re-scan » le support. Et effectivement :

```
# ls /dev/da0*
/dev/da0      /dev/da0p1  /dev/da0p2
/dev/da0p3    /dev/da0p4  /dev/da0p5
```

Dès lors, nous enchaînons directement sur le formatage de la partition FAT ou, plus justement, sur la création d'un système de fichiers FAT sur la première partition. Pour cela, rien de plus simple puisque FreeBSD nous met l'outil **newfs_msdos** directement à disposition :

```
# newfs_msdos -L MSDOSBOOT -F16 /dev/da0p1
/dev/da0p1: 614208 sectors in 19194 FAT16 clusters (16384 bytes/cluster)
BytesPerSec=512 SecPerClust=32 ResSectors=1 FATs=2
RootDirEnts=512 Media=0xf0 FATsecs=75 SecPerTrack=63
Heads=255 HiddenSecs=0 HugeSectors=614400
```

Remarquez la désignation du périphérique utilisé, avec **da0** pour désigner l'unité de stockage et **p1** pour préciser la partition. Le système de fichiers utilisé ici est FAT16 et nous spécifions un label **MSDOSBOOT**. Attention, il ne s'agit pas d'un label de partition (qui est « EFI »), mais bien de quelque chose de propre au système de fichiers. Pour nous faciliter la vie, immédiatement comme par la suite, nous créons des répertoires **/mnt/SDcible** et **/mnt/IMGsource** qui nous serviront de point de montage pour le reste des manipulations, respectivement pour accéder à la microSD et pour les partitions des images sources. Ce qui nous amène précisément à un des éléments clés de l'article, que nous allons voir en nous occupant de l'installation de FreeBSD.

2. FREEBSD

Les images fournies pour la quasi-totalité des systèmes destinés à fonctionner sur Raspberry Pi prennent généralement la forme d'images à transférer sur le support microSD avec **dd** (ou un équivalent comme **dcfldd**). Ces images, faisant généralement entre 1 et 3 Gio, regroupent toute la structure présente sur la SD, avec la partition FAT et celle destinée au système, utilisée

Démonologie appliquée :) :) :)

– Installer trois démons sur une Raspberry Pi 4 –

comme système de fichiers racine ou *rootfs*. Ces images de taille réduite (par rapport à la taille d'un support courant de 16, 32, 64, 128 ou encore 512 Gio) sont génériques et n'utilisent, par défaut, pas tout le support. C'est le premier démarrage du système sur la Pi qui déclenchera une procédure unique changeant la taille de la partition cible et étendant le système de fichiers en conséquence, à tout le reste du support.

Or, ici, nous ne voulons surtout pas procéder de la sorte en laissant faire le système. C'est nous qui structurons le support et déterminons les tailles utilisées. Ceci signifie donc que le travail normalement réalisé automatiquement devra être fait à la main, et ce, depuis un unique système qui n'est pas forcément celui du SBC lui-même. C'est là précisément la raison pour laquelle FreeBSD est parfait pour cela.

Pour accéder à ce qui nous intéresse dans l'image, nous devons être en mesure de spécifier chaque partie individuellement, chose que le système devine pour nous lorsque nous connectons un disque USB, par exemple. Nous pouvons faire de même avec une image en utilisant un disque mémoire (*memory disks*) et en créant un « faux » périphérique bloc à partir d'un simple fichier. Sous FreeBSD, ceci passe par l'utilisation d'un périphérique `md` (voir `man md`). Mais avant cela, commençons par récupérer une image via le site officiel : <https://download.freebsd.org/releases/arm64/aarch64/ISO-IMAGES/13.1/>. Le fichier qui nous intéresse est `FreeBSD-13.1-RELEASE-arm64-aarch64-RPI.img`.

Ceci fait, nous pouvons demander au système d'inventer un périphérique à partir de l'image :

```
# mdconfig FreeBSD-13.1-RELEASE-arm64-aarch64-RPI.img
md0
```

`md0` est notre pseudo-disque et il pourra être utilisé exactement comme `da0`. Le système en reconnaît d'ailleurs la composition sans le moindre problème :

```
# ls /dev/md0*
/dev/md0      /dev/md0s1    /dev/md0s2    /dev/md0s2a
```

Où sont passés les `p1`, `p2`, etc. ? L'image est partitionnée en MBR et non en GPT, comme le montre la sortie de `fdisk` :

```
# fdisk md0
***** Working on device /dev/md0 *****
[...]
The data for partition 1 is:
sysid 12 (0x0c), (DOS or Windows 95 with 32 bit FAT (LBA))
start 2079, size 102312 (49 Meg), flag 80 (active)
beg: cyl 0/ head 33/ sector 1;
end: cyl 6/ head 126/ sector 63
The data for partition 2 is:
sysid 165 (0xa5), (FreeBSD/NetBSD/386BSD)
start 104391, size 6187041 (3021 Meg), flag 0
beg: cyl 6/ head 127/ sector 1;
end: cyl 391/ head 158/ sector 63
```



```
The data for partition 3 is:
<UNUSED>
The data for partition 4 is:
<UNUSED>
```

FreeBSD interprète donc ces deux partitions (type 0C et A5) comme des *slices*, et le second possède un *disklabel* :

```
# disklabel /dev/md0s2
# /dev/md0s2:
8 partitions:
#      size  offset  fstype  [fsize  bsize  bps/cpg]
a:  6186880    57  4.2BSD      0      0      0
c:  6187041     0  unused      0      0      0 # "raw" part, don't edit
```

Ceci n'est pas bien important ici, puisque nous avons l'intention de tout simplement prendre ce qui existe au niveau des *slices* et de transférer cela dans les partitions GPT de la SD. Mais commençons par nous occuper du contenu FAT en montant les deux systèmes de fichiers :

```
# mount -t msdosfs /dev/md0s1 /mnt/IMGsource
# mount -t msdosfs /dev/da0p1 /mnt/SDcible
```

Comme vous pouvez le voir, monter un FS d'un périphérique mémoire (*md0*) ou physique (*da0*) est strictement identique. Nous pouvons alors copier le contenu de l'un vers l'autre, puis démonter les deux :

```
# cp -rv /mnt/IMGsource/* /mnt/SDcible
/mnt/IMGsource/armstub8-gic.bin -> /mnt/SDcible/armstub8-gic.bin
/mnt/IMGsource/armstub8.bin -> /mnt/SDcible/armstub8.bin
[...]
/mnt/IMGsource/start4x.elf -> /mnt/SDcible/start4x.elf
/mnt/IMGsource/u-boot.bin -> /mnt/SDcible/u-boot.bin

# umount /mnt/SDcible
# umount /mnt/IMGsource
```

Transférons maintenant le reste, mais cette fois avec *dd*, puisqu'il s'agit de données brutes. Oui, dans l'absolu, nous pourrions faire cela exactement de la même manière en prenant soin de conserver les méta-informations comme les permissions, etc. Mais il est bien plus simple de procéder comme suit. Copions donc les données du *disklabel* « a » vers la seconde partition GPT :

```
# dd if=/dev/md0s2a of=/dev/da0p2
6186880+0 records in
6186880 records out
3167682560 bytes transferred in 2294.873438 secs (1380330 bytes/sec)
```


Démonologie appliquée :) :) :)

- Installer trois démons sur une Raspberry Pi 4 -

Nous avons maintenant l'équivalent de l'image initiale, placée sur notre support, mais avec un partitionnement différent et un espace FAT bien plus important. Nous n'en avons pas fini, puisque nous devons empêcher le système de procéder aux modifications du support lors du premier démarrage. Pour cela, commençons par étendre le système de fichiers UFS à l'ensemble de la partition avec :

```
# growfs /dev/da0p2
It's strongly recommended to make a backup before growing the file system.
OK to grow filesystem on /dev/da0p2 from 2.9GB to 30GB? [yes/no] yes
super-block backups (for fsck_ffs -b #) at:
6402432, 7682880, 8963328, 10243776, 11524224, 12804672, 14085120,
15365568, 16646016, 17926464, 19206912, 20487360, 21767808, 23048256,
24328704, 25609152, 26889600, 28170048, 29450496, 30730944, 32011392,
33291840, 34572288, 35852736, 37133184, 38413632, 39694080, 40974528,
42254976, 43535424, 44815872, 46096320, 47376768, 48657216, 49937664,
51218112, 52498560, 53779008, 55059456, 56339904, 57620352, 58900800,
60181248, 61461696, 62742144
```

Nous pouvons ensuite monter le système de fichiers pour en modifier le contenu :

```
# mount /dev/da0p2 /mnt/SDcible

# df -h
Filesystem      Size  Used Avail Capacity  Mounted on
[...]
/dev/da0p2      29G   2,8G   24G     10%   /mnt/SDcible
```

29 Gio en tout et 24 Gio disponibles, parfait. Édisons maintenant `/mnt/SDcible/etc/rc.conf`, le fichier de configuration système où nous trouvons, entre autres, une ligne :

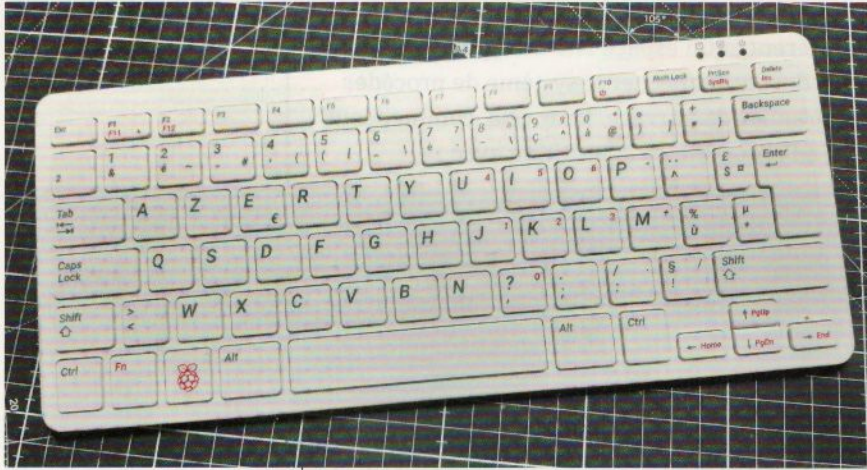
```
growfs_enable="YES"
```

qu'il convient de soit supprimer soit commenter en plaçant un `#` en début de ligne. On enregistre les modifications et on démonte avec `umount /mnt/SDcible`.

Une dernière modification, plus complexe, peut être intéressante et nous pouvons la faire dans la foulée. En effet, le système de démarrage utilisé repose sur U-Boot. Contrairement au démarrage classique Raspberry Pi OS où le *firmware* est chargé et lui-même charge directement le noyau Linux, ici nous avons le *firmware* Raspberry Pi qui charge le binaire d'un *stub* `armstub8-gic.bin`, qui va charger `u-boot.bin` qui s'occupera de trouver les éléments de démarrage (dont le binaire EFI) et d'initialiser le système. U-Boot est bien plus étoffé que

Pour éviter les surprises, optez pour des marques, modèles et sources fiables avec, comme ici, à gauche une microSD officielle Raspberry Pi et à droite, le support utilisé pour nos présentes expérimentations, provenant d'un distributeur de confiance (LDLC, pour ne pas le nommer). Si vous aimez le risque en revanche, Amazon, eBay ou AliExpress sont généralement de très bons choix...





L'ordinateur Raspberry Pi 400, ici en version française s'il vous plaît, est également une option pour ce genre de boot multiple. Ses performances sont peu ou prou identiques à celles d'une Raspberry Pi 4 B, mais il faudra faire attention au contenu du device tree, potentiellement sensiblement différent (non testé).

démarrage d'U-Boot (ce qui est possible ici) et de changer cet environnement pour l'adapter à ses besoins, le tout via une liaison série. Mais voilà, le binaire U-Boot ici présent (du moins à cette date) n'est pas en mesure d'enregistrer directement l'environnement sur le support SD (dans un fichier `uboot.env`).

Pour corriger le problème, nous devons changer la configuration par défaut dans les sources d'U-Boot et produire un nouveau binaire. Fort heureusement, le système de *ports* de FreeBSD nous permet de faire cela très rapidement et simplement. Ceci part du principe que l'arborescence de *ports* (sortes de paquets « sources » pour le commun des Linuxiens) est installée dans votre FreeBSD (sinon, voir [2]). On installe alors le compilateur croisé via `pkg install aarch64-none-elf-gcc python38`, puis on se rend dans `/usr/ports/sysutils/u-boot-rpi4`. Là, la modification consiste à éditer le fichier `files/rpi4_fragment` et à changer le contenu de :

```
CONFIG_ENV_FAT_DEVICE_AND_PART="1:1"
CONFIG_RPI_EFI_NR_SPIN_PAGES=2
```

en :

```
CONFIG_ENV_FAT_DEVICE_AND_PART="0:1"
CONFIG_RPI_EFI_NR_SPIN_PAGES=2
```

Les caractères magiques sont `0:1` en lieu et place de `1:1`, l'environnement doit être sauvegardé dans un fichier se trouvant sur le « disque » `0` et la première partition (`1`) de ce dernier, et non le disque `1`. On enregistre les modifications et on lance la construction d'un simple `make`. Il n'est pas utile de créer un paquet ou d'installer le résultat, ce qui nous intéresse se trouve dans le sous-répertoire `work/stage/usr/local/share/u-boot/u-boot-rpi4/u-boot.bin`, c'est le nouveau binaire à utiliser. On remonte alors le système de fichier FAT et on le copie, avant de tout démonter/défaire :

le processus de démarrage classique, il utilise un environnement, composé de variables et de séries de commandes et tests pour déterminer comment et sur quoi démarrer.

Il n'y a aucun problème avec cette configuration initiale qui fonctionne très bien. Mais lorsqu'on œuvre dans l'embarqué depuis un certain temps, on développe des habitudes, dont le fait de pouvoir interrompre le

Démonologie appliquée :) :) :)

– Installer trois démons sur une Raspberry Pi 4 –

```
# mount -t msdosfs /dev/da0p1 /mnt/SDcible

# cp -i work/stage/usr/local/share/u-boot/\
u-boot-rpi4/u-boot.bin /mnt/SDcible
overwrite /mnt/SDcible/u-boot.bin? (y/n [n]) y

# umount /mnt/SDcible

# mdconfig -d -u md0
```

Cette dernière commande détache le disque virtuel dont nous n'avons plus besoin. Nous pouvons maintenant placer la microSD dans le SBC, ajouter un adaptateur USB/série (115200 b/s 8N1) pour avoir la main sur le système dès le début (U-Boot compris) et alimenter.

Nous voyons alors défiler les messages, tout d'abord du *firmware* Raspberry Pi, puis de U-Boot, puis du *bootloader* EFI de FreeBSD, puis du noyau FreeBSD, qui passe la main à **init** (le vrai, pas une horreur tentaculaire comme Systemd), qui lance les services, dont SSH qui génère ses clés d'hôte pour la première fois... tout ça pour enfin arriver à :

```
Performing sanity check on sshd configuration.
Starting sshd.
Configuring vt: blanktime.
Starting background file system checks in 60 seconds.

Wed Mar  8 17:10:40 UTC
FreeBSD/arm64 (generic) (ttyu0)

login:
```

Deux utilisateurs sont déjà présents, l'indispensable **root** et un utilisateur standard **freebsd**, les deux ayant comme mot de passe leur nom d'utilisateur (**root/root** et **freebsd/freebsd** donc).

Si vous vous intéressez à U-Boot, il est possible d'interrompre le démarrage lorsque vous voyez ceci :

```
U-Boot 2021.07 (Mar 08 2023 - 18:26:12 +0100)

DRAM:  1.9 GiB
RPI 4 Model B (0xb03111)
MMC:   mmc@7e300000: 3, emmc2@7e340000: 0
Loading Environment from FAT...
Unable to read "uboot.env" from mmc0:1... In: serial
Out:   vidconsole
Err:   vidconsole
Net:   eth0: ethernet@7d580000
PCIe BRCM: link up, 5.0 Gbps x1 (SSC)
```



```

starting USB...
Bus xhci_pci: Register 5000420 NbrPorts 5
Starting the controller
USB XHCI 1.00
scanning bus xhci_pci for devices... 2 USB Device(s) found
      scanning usb for storage devices... 0 Storage Device(s) found
Hit any key to stop autoboot:  0
U-Boot>

```

Vous n'avez que deux secondes pour réagir, ce qui est peu. Mais nous avons heureusement fait le nécessaire pour corriger cela et pouvons donc :

```

U-Boot> setenv bootdelay 6
U-Boot> saveenv
Saving Environment to FAT... OK
U-Boot>

```

Nous définissons la variable d'environnement **bootdelay** à 6 (précédemment 2) et enregistrons l'environnement. Nous pouvons alors redémarrer avec **reset** et constater que nous avons bien un compte à rebours à partir de 6, mais aussi, après démarrage du système, que le système de fichiers FAT, monté dans **/boot/msdos**, contient désormais un fichier **uboot.env** avec notre environnement à jour.

Nous avons maintenant un système FreeBSD ARM 64 bits prêt à être utilisé et personnalisé. Mais ça, c'est une autre histoire qui n'a rien à faire dans ce magazine. Passons donc à la suite...

3. NETBSD

Étonnamment, NetBSD semble être moins avancé sur plateformes ARM que FreeBSD, alors que c'est normalement le système par excellence en termes de présence sur de nombreuses architectures. Ainsi, pour un support viable sur Raspberry Pi 4, il faudra se tourner vers les travaux de Jun Ebihara, alias « ebijun », mettant à jour régulièrement ses *builds* pour cette plateforme [3]. Celui-ci met à disposition de façon récurrente des images binaires sur <http://cdn.netbsd.org/pub/NetBSD/misc/jun/raspberry-pi>, aussi bien pour ARM 32 bits (*earmv6hf*) que 64 (*aarch64*). À cette date, la dernière version disponible est **2023-02-16-netbsd-raspi-aarch64.img.gz** qu'il conviendra de télécharger et de décompresser avec **gunzip** pour obtenir le fichier **2023-02-16-netbsd-raspi-aarch64.img**. Notez qu'il s'agit d'une version de développement 10.99 et absolument pas d'une *release* stable du système.

Ceci fait, nous sommes en mesure de procéder exactement comme pour FreeBSD, puisque la structure de l'image est toujours la même. Mais attention, le contenu de la partie FAT est très différent puisque NetBSD n'utilise non seulement pas U-Boot, mais repose sur un *firmware* UEFI du projet Tianocore EDK II directement chargé comme *stub* ARM depuis la configuration **config.txt** (**armstub=RPI_EFI.fd**). Ce *firmware* UEFI permet, entre autres choses, d'avoir une « interface de BIOS » assez similaire à ce qu'on trouve généralement sur PC (sur la sortie HDMI

Démonologie appliquée :) :) :)

– Installer trois démons sur une Raspberry Pi 4 –



FreeBSD, OpenBSD ou encore NetBSD se ne limitent pas aux seules Raspberry Pi. Entre ces trois OS, une vaste gamme de SBC est supportée (sauf peut-être l'AVR32 au centre). Il faudra cependant souvent jongler avec les images, les configurations et les documentations, pour arriver à un résultat. Et tout ceci se fera au cas par cas...

et la liaison série), permettant une configuration du système, mais aussi un choix du ou des binaires EFI à lancer.

Bien sûr, ceci n'est possible que parce qu'une configuration précise est utilisée, ce qui implique que le *device tree* est différent de celui servant de base pour FreeBSD. Nous touchons là au problème typique d'un manque d'unification entre les systèmes, puisque contrairement à une architecture PC amd64 (ou i386), il n'existe pas de normes ou standards autres que ceux établis (imposés ?) par le constructeur. De plus,

l'image elle-même utilise un système de partition GPT et non MBR comme c'était le cas avec FreeBSD.

Ceci signifie donc que le nommage des partitions dans `/dev` est sensiblement différent, mais aussi, et surtout que nous ne pouvons absolument pas utiliser le contenu FAT de FreeBSD et devons, pour passer d'un système à l'autre, changer ce contenu à chaque fois. Mon approche pour régler ce petit problème consistera, tout simplement, à conserver le contenu adapté à chaque OS dans un sous-répertoire dédié. Nous aurons donc, à terme, des répertoires **FREEBSD**, **NETBSD** et **OPENBSD** directement sur la première partition de la microSD et il suffira d'en copier le bon contenu à la racine pour « booter » le bon système. Ce n'est pas élégant, mais c'est scriptable et nous pourrions donc créer, par la suite, quelque chose comme un **reboot_prepare.sh** prenant en argument un nom et mettant à jour le contenu à la racine avant de « rebooter ».

La série de commandes pour transférer le contenu de l'image vers notre microSD sera donc :

```
# mdconfig 2023-02-16-netbsd-raspi-aarch64.img
md0

# mount -t msdosfs /dev/md0 /mnt/SDcible
# mount -t msdosfs /dev/md0p1 /mnt/IMGsource

# cd /mnt/SDcible
# mkdir FREEBSD
# cd FREEBSD
# mv ../* .
mv: rename ../FREEBSD to ../FREEBSD: Invalid argument
# cd ..
# ls -a
./      ../      FREEBSD/

# cp -rv /mnt/IMGsource/* .
[...]
/mnt/IMGsource/RPI_EFI.fd -> ./RPI_EFI.fd
/mnt/IMGsource/RPI_EFI.fd.org -> ./RPI_EFI.fd.org
/mnt/IMGsource/start4.elf -> ./start4.elf

# cd
# umount /mnt/IMGsource
```

Notez que le message d'erreur **Invalid argument** est parfaitement normal, puisque le répertoire **FREEBSD** ne peut être déplacé vers lui-même. Nous n'avons pas démonté **/mnt/SDcible**, car nous n'en avons pas fini avec son contenu. En effet, le fichier **config.txt** qui s'y trouve ne me plaît pas. Il fait usage d'un *overlay* du *device tree* qui configure la sortie série en utilisant *mini UART* (**miniuart-bt**), une déclinaison de l'UART non standard qui permet d'utiliser le Bluetooth sur le « vrai » port série (**PL011 UART0**). Je n'ai que faire du Bluetooth/BLE et une configuration alternative permet de désactiver cette fonctionnalité pour utiliser le port **PL011** en guise de console série.

Ceci se fait très simplement en changeant **config.txt** et en remplaçant la ligne :

```
dtoverlay=miniuart-bt
```

par :

```
dtoverlay=disable-bt
```

Mais nous avons un problème, car le *blob overlay* correspondant à cette configuration, **disable-bt.dtbo**, n'est pas présent dans le sous-répertoire **overlays/**. Nous n'y trouvons que ce qui est actuellement utilisé, à savoir **upstream-pi4.dtbo** et **miniuart-bt.dtbo**.

Démonologie appliquée :) :))

– Installer trois démons sur une Raspberry Pi 4 –

Pour régler cela, nous devons pointer notre navigateur sur <https://github.com/raspberrypi/firmware/tree/master/boot>, le dépôt officiel GitHub contenant les binaires précompilés du *firmware* Raspberry Pi. Dans cette pile de fichiers, nous devons récupérer et installer sur le support les fichiers suivants :

- `bcm2711-rpi-4-b.dtb`
- `bcm2711-rpi-400.dtb`
- `bcm2711-rpi-cm4.dtb`
- `overlays/disable-bt.dtbo`
- `overlays/miniuart-bt.dtbo`
- `overlays/upstream-pi4.dtbo`

L'idée étant ici d'éviter toute disparité de versions avec le contenu originellement fourni par l'image, puisque les *overlays* (`dtbo`) sont forcément dépendants du *Device Tree* (`dtb`). Ceci fait, et le `config.txt` dûment modifié, nous pouvons enfin démonter `/mnt/SDcible`.

Mais avant de détacher `md0` comme précédemment (`mdconfig -d -u md0`), nous devons nous occuper du système de fichiers racine, encore une fois avec `dd` mais en visant `/dev/da0p3` :

```
# dd if=/dev/md0p2 of=/dev/da0p3
4824995+0 records in
4824995+0 records out
2470397440 bytes transferred in 1789.905998 secs (1380183 bytes/sec)
```

Puis étendre ce système de fichiers à toute la partition :

```
# growfs /dev/da0p3
It's strongly recommended to make a backup before growing the file system.
OK to grow filesystem on /dev/da0p3 from 2.3GB to 30GB? [yes/no] yes
super-block backups (for fsck_ffs -b #) at:
4825032, 4850032, 4875032, 4900032, 4925032, 4950032,
[...]
62775032, 62800032, 62825032, 62850032, 62875032, 62900032
```

Pour finalement monter cela dans `/mnt/SDcible` et pouvoir accéder au fichier `/mnt/SDcible/etc/rc.conf` pour en modifier le contenu de :

```
rc_configured=YES
hostname=arm64
no_swap=YES
savecore=NO
sshd=YES
dhcpcd=YES
ntpd=YES
ntpd_flags="-g"
creds_msdos=YES
```



```

creds_msdos_partition=/boot
resize_gpt=YES
resize_root=YES
resize_root_flags="-p"
resize_root_postcmd="/sbin/reboot -n"
mdnsd=YES
devpubd=YES
wscons=$(dev_exists wsdisplay0)
ec2_init=$(dev_exists ena0)

```

en :

```

rc_configured=YES
hostname=arm64
no_swap=YES
savecore=NO
sshd=YES
ntptime=YES
dhcpcd=YES
dhcpcd_flags="-qMw genet0"
mdnsd=YES
devpubd=YES
wscons=$(dev_exists wsdisplay0)

```

Nous ajustons quelques éléments importants dont l'interface et les paramètres utilisés pour le DHCP (**dhcpcd_flags**), la désactivation du démon NTP remplacé par un appel unique à **ntptime**, la désactivation du redimensionnement du système de fichiers racine au premier boot (**resize_root**) et, enfin, la désactivation de la prise en charge de la configuration automatique des informations utilisateurs (*login*, mot de passe, clé SSH, etc.) à partir de fichiers présents sur le FS FAT (voir la page de manuel de **creds_msdos**).

Après cet ajustement, le système est désormais prêt pour être démarré. On démonte **/mnt/SDcible** et déconfigure **md0** :

```

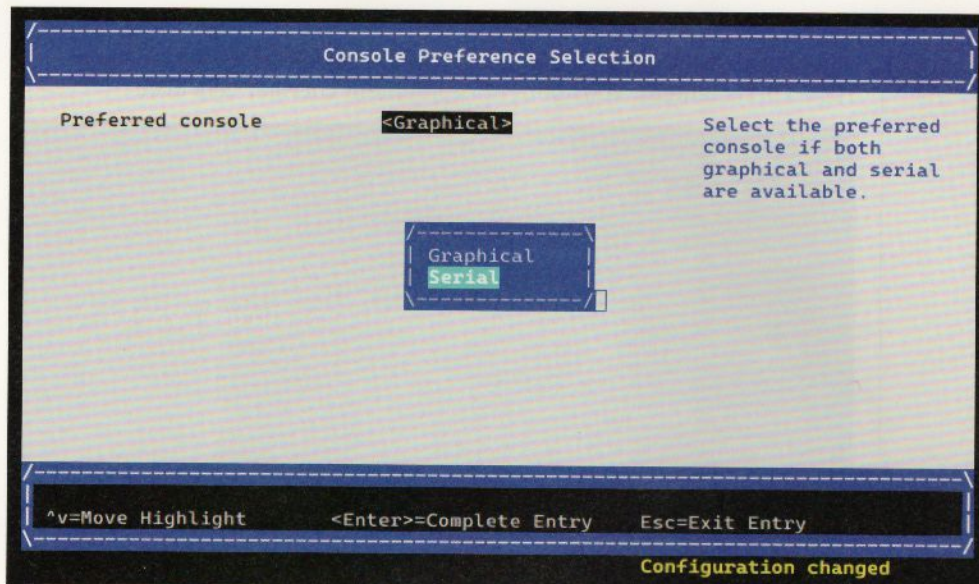
# umount /mnt/SDcible
# mdconfig -d -u md0

```

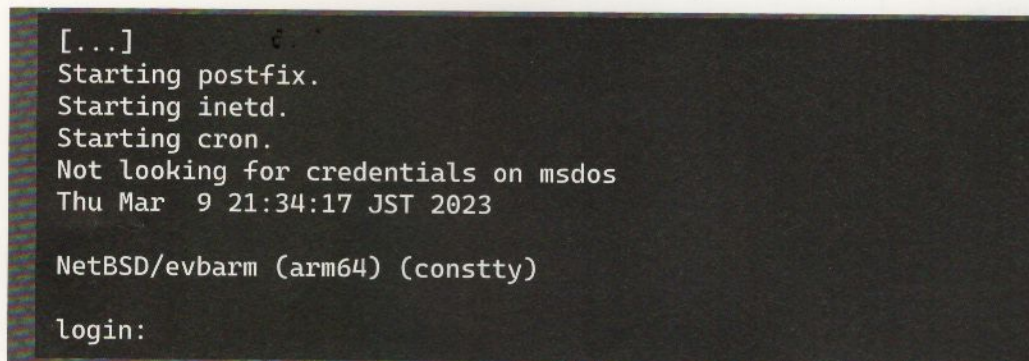
pour placer le support, une nouvelle fois, dans le SBC et démarrer. **Mais attention !** Nous devons impérativement interrompre ce premier démarrage via la liaison série en appuyant furieusement sur la touche *Echap* lorsque le message « *ESC (setup), F1 (shell), ENTER (boot)* » apparaît au bas du terminal. Nous nous retrouvons alors catapultés dans ce qui semble être une interface de BIOS de PC, et pour cause, c'est l'UEFI de EDK II. Là, nous pouvons nous balader dans la configuration via « *Device Manager* », puis « *Console Preference Selection* » pour choisir « *Serial* » comme « *Preferred console* », faute de quoi nous ne verrions rien apparaître via l'interface série lors du démarrage de NetBSD.

Démonologie appliquée :) :) :

- Installer trois démons sur une Raspberry Pi 4 -

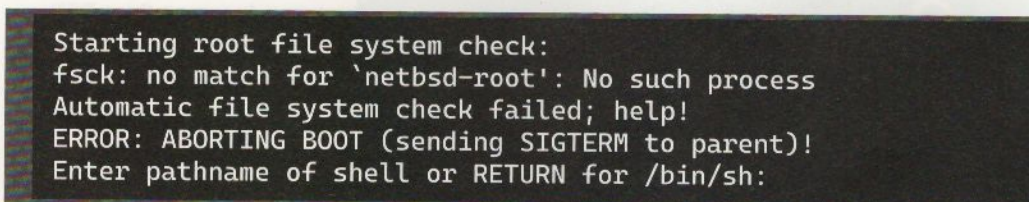


Ce choix validé, on utilise F10 pour enregistrer les changements qu'on confirme immédiatement, on retourne à la page d'accueil avec autant de *Echap* que nécessaire et on valide « Reset » pour redémarrer. Cette fois, nous laissons le processus se poursuivre et...



Le seul utilisateur disponible est **root** et vous pouvez vous connecter sans mot de passe (à changer impérativement, bien sûr). Bravo, vous avez installé un second démon sur le support microSD !

Notez que si vous voyez apparaître le message :



c'est tout simplement que vous avez oublié de donner le bon label (**netbsd-root**) à la partition 3 et NetBSD ne retrouve pas ses petits et panique.

S'il vous reste de vieilles framboises comme celle-ci, sachez que vous pourrez également faire fonctionner ces OS « exotiques » (et démoniques) dessus. Mais le présent article ne couvre absolument pas ce cas de figure. Si tel est votre envie, je vous conseille de commencer par NetBSD et l'article sur le sujet dans le numéro 41 en particulier (c'était sur Pi 2, mais cela s'adaptera sans trop de problèmes).

Autre point notable, si la date et heure ne vous paraissent pas « normales », pas de panique. Le fuseau horaire n'est simplement pas réglé correctement. Vous pouvez rapidement corriger cela avec :

```
arm64# date
Thu Mar  9 21:42:09 JST 2023

arm64# rm /etc/localtime

arm64# ln -s /usr/share/zoneinfo/Europe/Paris \
/etc/localtime

arm64# date
Thu Mar  9 13:42:51 CET 2023
```

Pourquoi était-on dans le fuseau JST (*Japan Standard Time*) ? Tout simplement parce que *ebijun* vit à Yokohama et qu'il a naturellement configuré son système en conséquence. Il en va de même pour le clavier, si vous désirez ne pas utiliser que la liaison série (et/ou SSH). Éditez le fichier `/etc/wscons.conf` et remplacez simplement `encoding jp` par `encoding fr`, avant de vous plier d'un petit `service wscons restart`. Et vous voici avec une organisation de clavier bien de chez nous.

Pour le reste, il s'agit de configuration parfaitement standard pour un NetBSD. Vous vous installerez confortablement et referez la déco plus tard, nous avons un troisième démon à invoquer et celui-là est un poisson qui pique !



4. OPENBSD

À présent que nous sommes bien habitués aux histoires de `dd`, d'extensions de FS et de modifications de configurations de démarrage, nous pouvons attaquer sereinement OpenBSD et... constater que la procédure sera totalement différente.

En effet, OpenBSD fournit effectivement une image sous la forme du fichier `miniroot72.img`

Démonologie appliquée :) :) :

– Installer trois démons sur une Raspberry Pi 4 –

sur <https://cdn.openbsd.org/pub/OpenBSD/7.2/arm64/>, mais il ne s'agit pas du système tel qu'il sera installé sur le support. C'est un installeur, composé d'un noyau (**bsd**) et d'un RAMDISK (**bsd.rd**) contenant les outils d'installation, le tout dans un *slice* contenant un *disklabel* avec un système de fichiers FFS incluant simplement ces deux éléments. Ceci, en plus d'une partition (ou *slice*) avec un système de fichier FAT, typique d'une Raspberry Pi.

La procédure initiale sera donc similaire à ce que nous avons déjà fait (si ce n'est que nous n'avons pas besoin de redimensionner le *rootfs*), mais uniquement pour démarrer l'installeur qui, avec notre assistance, va détruire le contenu de la partition 4 pour s'installer à la place.

Avant toute chose, commençons par remonter le système de fichiers FAT de la microSD pour déplacer tout ce qui se trouve à la racine (à l'exception de **FREEBSD**) dans un répertoire **NETBSD**. Ceci fait, nous attachons l'image **miniroot72.img** au disque mémoire **md0** et montons le contenu de la partie FAT, que nous copions sur la microSD :

```
# mdconfig OpenBSD72_arm64_miniroot72.img
md0

# mount -t msdosfs /dev/md0s1 /mnt/IMGsource

# cp -rv /mnt/IMGsource/* /mnt/SDcible
[...]
/mnt/IMGsource/start.elf -> /mnt/SDcible/start.elf
/mnt/IMGsource/start4.elf -> /mnt/SDcible/start4.elf
/mnt/IMGsource/u-boot.bin -> /mnt/SDcible/u-boot.bin

# umount /mnt/SDcible
# umount /mnt/IMGsource
```

Si vous prenez le temps de consulter ce qui vient d'être copié, vous constaterez que U-Boot est de retour (sans le *bug* de sauvegarde de l'environnement), mais que le contenu du **config.txt** est sensiblement différent de celui de FreeBSD, tout comme les *overlays* utilisés. Suffisamment différents pour ne pas être interchangeables malheureusement, mais nous avons déjà une solution à ce problème.

Nous pouvons ensuite copier le contenu de la partition 4 de l'image dans la partition 4 de la microSD :

```
# dd if=/dev/md0s4 of=/dev/da0p4 bs=1m
19+0 records in
19+0 records out
19922944 bytes transferred in 1.342951 secs (14835193 bytes/sec)
```


En effet, l'image n'a que deux partitions, mais le partitionnement est en MBR et la seconde est une partition primaire numérotée 4 :

```
The data for partition 1 is:
sysid 12 (0x0c), (DOS or Windows 95 with 32 bit FAT (LBA))
  start 32768, size 16384 (8 Meg), flag 80 (active)
    beg: cyl 1023/ head 255/ sector 63;
    end: cyl 1023/ head 255/ sector 63
The data for partition 2 is:
<UNUSED>
The data for partition 3 is:
<UNUSED>
The data for partition 4 is:
sysid 166 (0xa6), (OpenBSD)
  start 49152, size 38912 (19 Meg), flag 0
    beg: cyl 1023/ head 255/ sector 63;
    end: cyl 1023/ head 255/ sector 63
```

Les manipulations s'arrêtent là et nous pouvons directement détacher l'image avec `mdconfig -d -u md0`. La suite se passera directement, via la console série, sur la Pi4 qu'il ne nous reste plus qu'à démarrer.

Avec ce premier démarrage, nous sommes accueillis directement par l'installateur qui nous propose plusieurs choix :

```
Welcome to the OpenBSD/arm64 7.2 installation program.
(I)nstall, (U)pgrade, (A)utoinstall or (S)hell? S
```

Nous ne pouvons pas immédiatement déclencher le processus d'installation, car la copie brute de l'image laisse le support dans un état incohérent avec un `disklabel` ne correspondant absolument pas à la réalité. Nous demandons donc un shell afin de pouvoir tenir nous-mêmes la tronçonneuse et commençons par vérifier l'état des partitions :

```
# fdisk sd0
Disk: sd0          Usable LBA: 34 to 249737182 [249737216 Sectors]
# : type          [      start:          size ]
-----
0: EFI Sys        [      2048:          614400 ]
1: 516e7cb6-6ecf-11d6-8ff8-00022d09712b [     616448:        62914560 ]
2: 49f48d5a-b10e-11dc-b99b-0019d1879648 [     63531008:        62914560 ]
3: OpenBSD        [    126445568:        62914560 ]
4: Linux files*   [    189360128:        60375040 ]
```

Nous retrouvons effectivement ce que nous avons précédemment configuré et tout est cohérent, mais il n'en va pas de même pour les labels de la quatrième partition que nous décidons donc de réinitialiser (`-E` pour le mode interactif et `-d` pour ignorer ce qui existe et utiliser les valeurs par défaut) :

Démonologie appliquée :) :) :)

– Installer trois démons sur une Raspberry Pi 4 –

```
# disklabel -dE sd0
Label editor (enter '?' for help at any prompt)
sd0> p g
OpenBSD area: 126445568-189360128; size: 30.0G; free: 30.0G
#          size          offset  fstype [fsize bsize  cpg]
c:         119.1G          0      unused
i:          0.3G        2048    MSDOS
j:         30.0G       616448  unknown
k:         30.0G       63531008 unknown
l:         28.8G       189360128  ext2fs
sd0> w
sd0> q
```

w nous permet d'écrire ce nouvel état et **q** quitte l'outil. Nous pouvons alors reprendre le processus d'installation en appelant directement le script :

```
# ./install
```

L'installateur peut être jugé assez spartiate, en comparaison avec d'autres systèmes, mais est en réalité bien suffisant et surtout efficace. Il nous demande tout d'abord de valider le type de terminal puis attend un nom d'hôte pour le futur système :

```
Terminal type? [vt220]
System hostname? (short form, e.g. 'foo') pi4obsd
```

Nous passons ensuite à la configuration de l'interface réseau (**bse0** pour l'Ethernet de la Pi). Nous validons simplement les propositions par défaut, qui conviennent parfaitement :

```
Available network interfaces are: bse0 bwfm0 vlan0.
Network interface to configure? (name, lladdr, '?', or 'done') [bse0]
IPv4 address for bse0? (or 'autoconf' or 'none') [autoconf]
IPv6 address for bse0? (or 'autoconf' or 'none') [none]
Available network interfaces are: bse0 bwfm0 vlan0.
Network interface to configure? (name, lladdr, '?', or 'done') [done]
Using DNS domainname my.domain
Using DNS nameservers at 81.253.149.13 80.10.246.5
```

Vient ensuite la définition d'un mot de passe pour le super-utilisateur (**root**) :

```
Password for root account? (will not echo)
Password for root account? (again)
```


Puis l'activation de SSH et la création d'un utilisateur standard :

```
Start sshd(8) by default? [yes]
Setup a user? (enter a lower-case loginname, or 'no') [no] denis
Full name for user denis? [denis]
Password for user denis? (will not echo)
Password for user denis? (again)
WARNING: root is targeted by password guessing attacks,
pubkeys are safer.
Allow root ssh login? (yes, no, prohibit-password) [no]
```

On passe ensuite à la partie réellement sensible avec la configuration des disques, qui se fait en deux étapes avec d'abord les partitions, puis les *disklabels* :

```
Available disks are: sd0.
Which disk is the root disk? ('?' for details) [sd0]
Disk: sd0      Usable LBA: 34 to 249737182 [249737216 Sectors]
#  type                [ start:      size ]
-----
0:  EFI Sys             [ 2048:      614400 ]
1:  516e7cb6-6ecf-11d6-8ff8-00022d09712b [ 616448:    62914560 ]
2:  49f48d5a-b10e-11dc-b99b-0019d1879648 [ 63531008:   62914560 ]
3:  OpenBSD             [ 126445568:  62914560 ]
4:  Linux files*        [ 189360128:  60375040 ]
Use (W)hole disk or (E)dit the GPT? [whole] E
```

Les partitions sont déjà parfaitement configurées/établies, nous voyons notre partition/slice **OpenBSD** et l'installateur nous propose deux choix : utiliser tout le disque ou éditer ce qui existe. Nous éditons, car il est hors de question de gâcher tout le travail déjà effectué, mais n'avons rien à réellement modifier. Nous nous contentons donc de quitter l'éditeur de partitions pour passer à la prochaine étape :

```
Enter 'help' for information
sd0: 1> q
```

Concernant les *disklabels*, l'installateur propose automatiquement une division « idéale » en fonction de la taille de l'espace disponible. Avec nos quelque 30 Gio, celui-ci estime ceci pertinent :

```
The auto-allocated layout for sd0 is:
#      size      offset  fstype [fsize bsize  cpg]
a:    1024.0M    126445568  4.2BSD 2048 16384    1 # /
b:    2053.8M    128542720    swap
c:    121942.0M      0    unused
d:    1699.0M    132748928  4.2BSD 2048 16384    1 # /tmp
e:    2645.9M    136228512  4.2BSD 2048 16384    1 # /var
f:    3473.8M    141647392  4.2BSD 2048 16384    1 # /usr
g:     976.1M    148761728  4.2BSD 2048 16384    1 # /usr/X11R6
h:    3984.7M    150760832  4.2BSD 2048 16384    1 # /usr/local
```


Démonologie appliquée :) :) :)

- Installer trois démons sur une Raspberry Pi 4 -

```
i:          300.0M          2048    MSDOS
j:          30720.0M        616448 unknown
k:          30720.0M        63531008 unknown
l:          29480.0M        189360128 ext2fs
m:           1894.8M        158921472 4.2BSD   2048 16384   1 # /usr/src
n:           5909.5M        162801920 4.2BSD   2048 16384   1 # /usr/obj
o:           7058.4M        174904608 4.2BSD   2048 16384   1 # /home
Use (A)uto layout, (E)dit auto layout, or create (C)ustom layout? [a] c
```

Ce genre de répartition peut plaire, mais personnellement, la hantise de me retrouver à l'étroit à un endroit, tout en ayant moult Gio inutiles à un autre, est un vrai problème. Je préfère largement, en particulier dans ce cas de figure, tout avoir à un seul endroit, d'autant que **/usr/src** (sources du système) et **/usr/obj** (fichiers objets lors de la construction du système) risquent de ne pas servir du tout ici. Nous optons donc pour une organisation personnalisée (*custom layout*) et créons nous-mêmes un label pour le système de fichiers racine :

```
Label editor (enter '?' for help at any prompt)
sd0> a
partition to add: [a]
offset: [126445568]
size: [62914560]
FS type: [4.2BSD]
mount point: [none] /
sd0*> p g
OpenBSD area: 126445568-189360128; size: 30.0G; free: 0.0G
#          size          offset  fstype  [fsize bsize  cpg]
a:          30.0G        126445568  4.2BSD   2048 16384    1 # /
c:          119.1G           0    unused
i:           0.3G          2048    MSDOS
j:          30.0G        616448    unknown
k:          30.0G        63531008    unknown
l:          28.8G        189360128  ext2fs
sd0*> w
sd0> q
```

Cette étape passée, l'installateur s'empresse d'initialiser le système de fichiers en question :

```
/dev/rsd0a: 30720.0MB in 62914560 sectors of 512 bytes
152 cylinder groups of 202.50MB, 12960 blocks, 25920 inodes each
/dev/sd0a (d094952e2187396c.a) on /mnt type ffs (rw, asynchronous, local)
```

Puis passe à l'installation proprement dite du système. On nous propose tout d'abord de choisir la méthode d'installation qui sera ici via HTTP (nous téléchargeons les *sets*) et devons spécifier où les récupérer :

```
Let's install the sets!
Location of sets? (disk http nfs or 'done') [http]
HTTP proxy URL? (e.g. 'http://proxy:8080', or 'none') [none]
```



```
(Unable to get list from openbsd.org, but that is OK)
HTTP Server? (hostname or 'done') cdn.openbsd.org
Server directory? [pub/OpenBSD/snapshots/arm64] pub/OpenBSD/7.2/arm64
```

L'installateur nous présente alors la liste de ce qui est installable ainsi :

```
Select sets by entering a set name, a file name pattern or 'all'. De-select
sets by prepending a '-', e.g.: '-game*'. Selected sets are labelled '[X]'.
[X] bsd             [X] base72.tgz      [X] game72.tgz      [X] xfont72.tgz
[X] bsd.mp          [X] comp72.tgz     [X] xbase72.tgz    [X] xserv72.tgz
[X] bsd.rd          [X] man72.tgz       [X] xshare72.tgz
Set name(s)? (or 'abort' or 'done') [done]
```

Comme le précise le message, vous pouvez retirer et ajouter des *sets* en utilisant - ou +, suivi de tout ou partie du nom de l'archive du *set* concerné, complété du caractère joker *. Dans notre cas, nous n'avons que faire de X (appelé « Xenocara » pour la version OpenBSD) et des jeux. Nous pouvons donc désactiver cela ainsi :

```
Set name(s)? (or 'abort' or 'done') [done] -g*
Set name(s)? (or 'abort' or 'done') [done] -x*
[X] bsd             [X] base72.tgz      [ ] game72.tgz      [ ] xfont72.tgz
[X] bsd.mp          [X] comp72.tgz     [ ] xbase72.tgz    [ ] xserv72.tgz
[X] bsd.rd          [X] man72.tgz       [ ] xshare72.tgz
Set name(s)? (or 'abort' or 'done') [done]
```

Il ne nous reste ensuite plus qu'à valider (done) pour déclencher l'installation :

```
Get/Verify SHA256.sig 100% |*****| 1544 00:00
Signature Verified
Get/Verify bsd 100% |*****| 14649 KB 00:02
Get/Verify bsd.mp 86% |*****| 12800 KB 00:00 ETA
[...]
Installing bsd 100% |*****| 14649 KB 00:02
Installing bsd.mp 100% |*****| 14712 KB 00:02
Installing bsd.rd 100% |*****| 17269 KB 00:02
Installing base72.tgz 8% |**| 22144 KB 01:56 ETA
[...]
Location of sets? (disk http nfs or 'done') [done]
```

L'installateur nous demande encore notre fuseau horaire et finit son travail :

```
What timezone are you in? ('?' for list) [Canada/Mountain] Europe/Paris
Saving configuration files... done.
Making all device nodes... done.
Multiprocessor machine; using bsd.mp instead of bsd.
fw_update: added bwfm; updated none; kept none
Relinking to create unique kernel...
done.
```


Démonologie appliquée :) :) :)

– Installer trois démons sur une Raspberry Pi 4 –

```
CONGRATULATIONS! Your OpenBSD install has been successfully completed!

When you login to your new system the first time, please read your mail
using the 'mail' command.

Exit to (S)hell, (H)alt or (R)eboot? [reboot] H
syncing disks... done

The operating system has halted.
Please press any key to reboot.
```

En déconnectant et reconnectant l'alimentation du SBC, le système démarre et nous invite à nous connecter :

```
[...]
starting local daemons: cron.
Sun Feb 19 01:28:54 CET 2023

OpenBSD/arm64 (pi4obsd.my.domain) (console)

login:
```

Et voici notre troisième démon prêt à nous servir ! Vous remarquerez sans doute un ralentissement au démarrage, en dehors de la génération des clés d'hôte SSH qui n'intervient qu'une fois, ponctué d'un message « *reordering libraries* ». Ceci est un mécanisme de sécurité appelé ASLR (*Address Space Layout Randomization*), compagnon de KARL (*Kernel Address Randomized Link*) présent dans OpenBSD, consistant à avoir des binaires uniques et avec des structures internes à des positions non prédictibles, à chaque installation, mise à jour ou démarrage. Sur un SBC, ceci peut être pénalisant en termes de rapidité de démarrage. Si vous souhaitez désactiver cette fonctionnalité, vous pouvez utiliser la commande suivante :

```
# rcctl disable library_aslr
```

5. ET RASPBERRY PI OS ?

Oui, nous pourrions poursuivre avec une installation de GNU/Linux et peut-être que tout ceci se verra ainsi complété à un moment, mais pour l'heure ce n'est pas au programme. Raspberry Pi OS, alias Raspbian, alias Debian GNU/Linux ARM adapté aux SBC Raspberry Pi est l'OS par défaut et ceci à mon sens ne présente que peu d'intérêt dans une telle configuration *multiboot*. Après tout, nous avons tous des Raspberry Pi OS qui fonctionnent de-ci de-là et cela reste la plateforme première pour bon nombre de développements et projets. Nous serions en terrain connu.

C'est un tout autre sujet lorsqu'il s'agit de ces trois héritiers du très respectable 4.4BSD. Bien sûr, la quasi-totalité des fonctionnalités est au rendez-vous, mais pas toutes. Il y a encore du travail à faire, des choses à tester, à adapter, à affiner et nombre de *bugs* à découvrir et à signaler aux développeurs.

Il en va de même pour mes propres codes, qui sont principalement développés sous GNU/Linux (à quelques exceptions près) et pour GNU/Linux, que ce soit sur amd64 ou aarch64. Là où les inconnues se font jour, c'est forcément sur ces OS qui sont encore très « secondaires » en ce qui concerne mon usage courant, en dehors de quelques exceptions (serveurs, VPN, NAS, etc.) et vis-à-vis desquels je ne possède pas l'expérience dont je peux jouir avec GNU/Linux. Car c'est aussi là la raison d'une telle entreprise : apprendre, et pas seulement sur un FreeBSD installé sur un Mini-PC Ryzen 5, un NetBSD sur un ThinkPad X61s ou un OpenBSD confortablement installé sur un Samsung NB30.

En termes de résultats et de « finitions », le support de NetBSD est encore assez limité, mais les choses progressent rapidement. Ceci reste toutefois une version de développement, même si la stabilité est relativement au rendez-vous. Je ne peux malheureusement pas en dire autant pour OpenBSD dont le comportement est assez étrange et de nombreux problèmes ont été rencontrés, en particulier avec l'accès au réseau, y compris le DHCP échouant assez fréquemment tout comme le démarrage ou l'arrêt de services comme `mdnsd`, par exemple. Le grand gagnant de l'histoire est sans le moindre doute FreeBSD qui n'a montré absolument aucun signe d'instabilité et fonctionne à merveille sur la RPi4.

Comprenez bien que ceci n'est absolument pas une critique et qu'il faut bien comprendre que dans le cas de Raspberry Pi, et même si une bonne partie du code est libre, ce n'est absolument pas le cas de la maigre documentation et du matériel (*firmware* inclus). Si GNU/Linux est si populaire

sur cette plateforme, c'est uniquement parce que c'est le système choisi par la fondation et que les développeurs Broadcom, qui ont plein accès aux éléments techniques, peuvent se charger de développer le support. Même d'autres distributions GNU/Linux, comme Devuan, peinent à atteindre un niveau égal de stabilité, ce n'est pas par hasard et c'est relativement triste. Mais peut-être que l'émergence de SBC RISC-V changera la donne à un moment, grâce à une concurrence féroce et bénéfique entre les deux architectures...

Le seul grand absent ici est DragonFly BSD, un *fork* relativement ancien (2003) de FreeBSD, qui gagne en popularité (en particulier en raison de son système de fichiers HAMMER2), mais n'est pour l'instant disponible que sur architecture amd64. La cinquième partition sera peut-être celle de ce système lorsqu'une version de développement sera proposée pour ARM.

Je doute que vous décidiez d'opter pour une configuration similaire à celle décrite ici, c'est très spécifique, mais cette petite aventure vous aura certainement donné une vision de ce qu'il est possible de faire et, peut-être, vous poussera à élargir vos horizons et à découvrir au moins un de ces systèmes qui reposent sur une philosophie radicalement différente de ce qu'est devenu aujourd'hui GNU/Linux et Raspberry Pi OS... **DB**

RÉFÉRENCES

- [1] <https://connect.ed-diamond.com/hackable/hk-045/personnalisez-automatiquement-vos-installations-de-raspberry-pi-os>
- [2] <https://docs.freebsd.org/en/books/handbook/ports/>
- [3] <https://github.com/ebijun/NetBSD/tree/master/RPI/RPIimage/Image/aarch64>

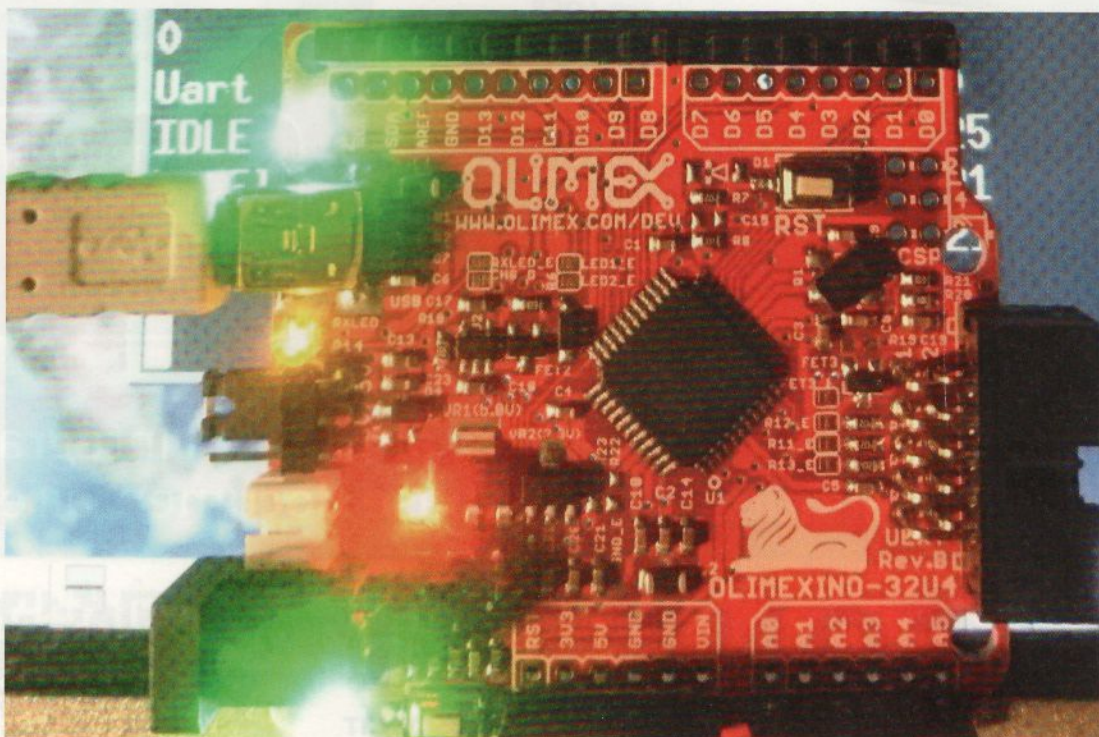
FREERTOS DANS 2,5 KB DE RAM SUR ATMEGA32U4

J.-M Friedt

[Enseignant-chercheur à l'université de Franche-Comté (Besançon)]

FreeRTOS [1], l'environnement exécutif de Richard Barry plébiscité par Amazon Web Services (AWS), fonctionne sur une plateforme matérielle ou son émulateur munis de seulement 2,5 KB de RAM.

La mise en œuvre de FreeRTOS dans aussi peu de ressources fournit une opportunité de plonger dans les détails de l'implémentation de ses fonctions.



Le microcontrôleur Atmega32U4 fait partie des petites architectures 8-bits très faciles à appréhender, compte tenu de ses faibles ressources et de la taille réduite de la documentation technique limitée à quelque 400 pages [2]. Pour son malheur, il s'agit aussi d'une plateforme communément supportée par la bibliothèque Arduino avec toutes les bêtises qui traînent sur le Web autour de cette approche supposée simple de la programmation. Malgré les ressources très réduites des 2,5 KB de RAM, nous nous interrogeons sur la capacité à exécuter l'environnement exécutif FreeRTOS et ainsi aborder les techniques classiques de développement de systèmes embarqués exploitant une multitude de tâches – et donc l'accès concurrent aux ressources protégé par *mutex* et sémaphores – voire le partage d'informations entre tâches par les *queues*.

1. FREERTOS

Nous avons déjà abordé FreeRTOS sur STM32 dans [3] et [4] en argumentant que l'environnement exécutif donne l'impression au développeur de travailler sur un système d'exploitation avec

plusieurs tâches exécutées apparemment simultanément, sous la supervision d'un ordonnanceur (*scheduler*), mais sans autoriser l'allocation dynamique de ressources par le chargement d'un exécutable ou d'une bibliothèque, le binaire exécuté par le système embarqué étant « cross-compilé » de façon statique sur la plateforme de travail hôte – généralement un PC compatible Intel – vers la cible, ici un microcontrôleur 8-bits. Ce faisant, plusieurs développeurs peuvent collaborer sur un même projet en séparant les tâches et en partageant le fruit de leurs calculs, non par une API dont ils décident entre eux, mais par les mécanismes fournis par l'environnement exécutif, par exemple les queues de données [4]. Finalement, seul l'ordonnanceur peut garantir l'atomicité d'une opération, c'est-à-dire de ne pas changer de contexte de tâche au cours d'une opération complexe telle que lire le statut d'un drapeau, modifier ce statut et stocker le résultat : c'est donc à lui de coordonner les requêtes d'accès aux ressources communes que sont les périphériques matériels ou la mémoire, et ce grâce à des drapeaux garantissant un accès mutuellement exclusif nommés *mutex*.

Parmi la pléthore d'environnements exécutifs qui ne cessent de se multiplier, FreeRTOS se limite à seulement 6 fichiers portables en C donc facilement « cross-compilables », et un fichier nommé **port.c** qui contient les implémentations spécifiques à une architecture et un compilateur donné. Dans l'arborescence actuelle de FreeRTOS à <https://github.com/FreeRTOS/FreeRTOS-Kernel>, les 6 fichiers portables sont **event_groups.c**, **list.c**, **queue.c**, **stream_buffer.c**, **tasks.c** et **timers.c** tandis que la partie non portable se trouve dans... **portable** ! On trouvera dans la variable SRCS du **Makefile** de https://github.com/jmfriedt/tp_freertos/blob/master/1basic/Makefile.avr32u4 l'ensemble des fichiers nécessaires à la compilation, à savoir ceux, portables, fournis par FreeRTOS, le gestionnaire d'allocation de mémoire issu de **portable/MemMang** (voir plus bas ce choix), l'implémentation locale de quelques fonctions d'initialisation, de communication et d'abstraction du matériel pour rendre le code FreeRTOS portable (ainsi au lieu de manipuler les registres des GPIO – ports d'entrée-sortie généralistes – pour changer l'état des LED, nous appelons une fonction générique **Led_Hi** et **Led_Lo** dont la bibliothèque supportant un microcontrôleur donné fournit l'implémentation),

et les fonctions spécifiques au microcontrôleur ciblé avec `port.c`. Une inspiration pour notre objectif d'appréhender l'Atmega32U4 pourrait être `portable/GCC/ATMega323`, mais les détails entre les deux microcontrôleurs sont trop éloignés pour fonctionner immédiatement. En partant de <https://github.com/feilipu/miniAVRfreeRTOS> qui a extrait une version donnée de FreeRTOS et a ajouté les fonctions spécifiques pour l'Atmega32U4, nous avons séparé la partie non portable en vue de l'insérer dans l'arborescence officielle de FreeRTOS, et ainsi toujours bénéficier des dernières évolutions sans nous arrêter à une version donnée, qui deviendra inévitablement obsolète. Il semble ainsi peu judicieux, comme le fait <https://github.com/feilipu/miniAVRfreeRTOS>, d'inclure la partie portable de FreeRTOS dans une archive sur GitHub qui ne peut ainsi que devenir rapidement obsolète vis-à-vis des évolutions du projet officiel, tel que nous le constatons déjà ici avec l'utilisation de FreeRTOS 10.5.0 d'il y a 7 mois (version du 16 septembre 2022).

2. FREERTOS SUR ATMEGA32U4

2,5 KB de mémoire ou très exactement (page 18, section 5 de [2]) 2560 octets de RAM, ce n'est vraiment pas beaucoup, tout juste ce que proposait le vénérable ZX81/TS1000 en 1982. Nous avons cependant gagné en performance en quittant le Basic pour un compilateur C performant et une utilisation efficace des pointeurs au lieu de `PEEK` et `POKE`. Par ailleurs, FreeRTOS ne gaspille pas de ressources avec des pilotes et des interfaces standardisées pour accéder au matériel, dont le contrôle passe toujours par la lecture de la *datasheet* et l'écriture dans les registres adéquats. Ainsi, avant toute velléité d'aborder FreeRTOS, il faudra s'assurer d'avoir des bases solides de bibliothèques permettant d'accéder aux périphériques de communication : nous avons ainsi proposé https://github.com/jmfriedt/tp_freertos/blob/master/common/usart_atmega.c le minimum vital pour initialiser le port de communication asynchrone compatible RS232 dans `Usart1_Init()`, l'initialisation des GPIO auxquels sont connectées les LED et une paire de fonctions pour abstraire le matériel et rendre les codes sources FreeRTOS portables entre les

diverses architectures abordées par le dépôt https://github.com/jmfriedt/tp_freertos (STM32 avec la bibliothèque de ST Microelectronics ou `libopencm3`, Stellaris, maintenant Atmel AVR), et quelques fonctions de communication pour afficher un caractère `uart_putc(char)` et donc une chaîne de caractères `uart_puts(char *)`. Nous voici donc équipés pour initialiser et communiquer avec le matériel.

En termes d'arborescence des codes sources, nous avons fait le choix de partir des codes sources de FreeRTOS disponibles à <https://www.freertos.org/a00104.html> (version 202212.01 à la date de rédaction), et de copier (`cp -r`) le contenu de Atmega32U4 de https://github.com/jmfriedt/tp_freertos dans le répertoire `FreeRTOS/Source/portable/GCC/` de l'environnement exécutif pour ajouter le support de ce microcontrôleur. Finalement, dans le premier exemple le plus simple `1basic`, les règles de compilation `Makefile.avr32u4` font appel au contenu du répertoire `FREERTOS_PATH` supposé pointer sur le sommet de l'arborescence de FreeRTOS, les diverses règles se contentant de compiler les 6 fichiers portables en C de FreeRTOS, le fichier spécifique à l'architecture cible, et la bibliothèque de fonctions de communication que nous avons proposée.

L'exemple le plus simple de FreeRTOS consiste en :

```
#if defined (__AVR_ATmega32U4__)
#include "FreeRTOSVariant.h"
#endif
#include "FreeRTOS.h"
#include "task.h"
#include "common.h"

void vLeds1(void* dummy)
{while(1)
{Led_Hi1(); vTaskDelay(120/portTICK_RATE_MS);
Led_Lo1(); vTaskDelay(120/portTICK_RATE_MS);
}
}

void vLeds2(void* dummy)
{while(1)
{Led_Hi2(); vTaskDelay(301/portTICK_RATE_MS);
Led_Lo2(); vTaskDelay(301/portTICK_RATE_MS);
}
}

void vUart(void* dummy)
{portTickType last_wakeup_time;
last_wakeup_time = xTaskGetTickCount();
while(1)
{uart_puts("\nHello World\r\n");
vTaskDelayUntil(&last_wakeup_time, 500/portTICK_RATE_MS);
}
}

int main(void){
Usart1_Init(); // inits clock as well
Led_Init();
Led_Hi1();
if (!(pdPASS == xTaskCreate( vLeds1,(signed char*)
"LedFloat",64,NULL,1,NULL))) goto hell;
if (!(pdPASS == xTaskCreate( vLeds1,(signed char*)
"LedFlash",64,NULL,2,NULL))) goto hell;
if (!(pdPASS == xTaskCreate( vUart, (signed char*) "Uart",
64,NULL,3,NULL))) goto hell;
vTaskStartScheduler();
hell: while(1); // should never be reached
return 0;
}
```

qui se résume à la déclaration de deux tâches **vLeds1()** et **vLeds2()** faisant clignoter une LED sur un GPIO avec une période différente pour chaque tâche – noter le préfixe **v** de chaque procédure indiquant qu'elle ne renvoie rien (**void**) dans la convention de nommage de FreeRTOS –

Figure 1 :
Le microcontrôleur
exécutant FreeRTOS
avec 3 tâches, deux
faisant clignoter les
LED orange et verte
de la carte Olimexino-
32U4, et la troisième
communiquant un
message sur le port
de communication
asynchrone.

et `vUart()` qui affiche un message sur le port de communication asynchrone. Chaque tâche contient des attentes selon deux mécanismes complémentaires que sont `vTaskDelayUntil()` ou `vTaskDelay()` afin de laisser la liberté à l'ordonnanceur d'orchestrer les diverses tâches selon les règles de préemption en vigueur – option `configUSE_PREEMPTION` ou `configUSE_TIME_SLICING` dans `FreeRTOSConfig.h` pour choisir le mode de commutation entre tâches. On retiendra que l'implémentation de FreeRTOS sur Atmega32U4 utilise l'oscillateur du chien de garde (*watchdog*) pour cadencer ses opérations et que cet oscillateur est documenté comme inexact, étant simplement formé d'un grossier circuit RC au lieu d'être

asservi sur un résonateur à quartz, et qu'il ne faut donc pas s'attendre à une exactitude meilleure qu'une dizaine de % sur les intervalles de temps requis. Ces attentes pour retirer la tâche de la queue d'exécution de l'ordonnanceur facilitent la prévision du cadencement des opérations au lieu de s'appuyer sur les priorités puisqu'ici, nous expliquons à l'ordonnanceur qu'une tâche a achevé ses opérations et peut laisser le temps aux autres fonctions de s'exécuter.

La fonction principale `main()` commence par initialiser les périphériques (`Usart1_Init()` et `Led_Init()` pour le port de communication et les GPIO en sortie), puis enregistre chaque tâche auprès de l'ordonnanceur avant de l'exécuter. La pile de la fonction `main()` est détruite au lancement de l'ordonnanceur par `vTaskStartScheduler()` et il faudra prendre soin de stocker toute variable passée en argument aux tâches sur le tas en les préfixant du qualificatif `const`. Nous avons cependant dû bricoler un peu pour définir des constantes en chargeant de façon conditionnelle `FreeRTOSVariant.h` qui romprait sinon avec la portabilité du code FreeRTOS entre diverses architectures. Le test sur la nature du microcontrôleur cible est inspiré des tests définissant le chargement de constantes dans `/usr/lib/avr/include/avr/io.h` qui teste la constante produite par l'argument `-mmcu=` en argument du compilateur `avr-gcc` (paquet `gcc-avr` sous Debian/GNU Linux



et son implémentation des fonctions classiques du C dans le paquet **avr-libc** qui fournit ces fichiers d'en-tête) pour choisir la liste des ports actifs.

La compilation de ce programme par **make -f Makefile.avr32u4**, puis son transfert en mémoire non volatile du microcontrôleur par **avrdude -c avr109 -b57600 -D -p atmega32u4 -P /dev/ttyACM0 -e -U flash:w:output/main.elf** se conclut par son exécution telle qu'illustrée en Figure 1.

3. ÉMULATION DE L'ATMEGA32U4 : SIMAVR

Nous avons exprimé notre admiration [5] pour l'excellent émulateur de microcontrôleurs Atmel qu'est **simavr** et en particulier sa capacité à émuler les périphériques qui entourent le cœur du processeur. Nous allons donc vérifier le bon fonctionnement de FreeRTOS sur Atmega32U4 dans cet environnement de simulation, ne serait-ce que pour que le lecteur ne possédant pas le matériel puisse poursuivre la démonstration par une mise en pratique. Afin de ne pas nous épuiser avec une représentation graphique des LED qui nous détournerait de l'objectif initial de cette présentation, nous activons le mode de déverminage de **simavr** téléchargé depuis <https://github.com/buserror/simavr> en modifiant la ligne 25 de https://github.com/buserror/simavr/blob/master/simavr/sim/avr_ioport.c par :

```
#define D(_w) _w
```

Par défaut, la macro **D** ne fait rien et donc n'affiche pas les messages de déverminage. Une fois la modification faite, la compilation s'obtient sans trop de problèmes par **make...** sauf pour les applications graphiques qui échouent peut-être (toujours elles !) si nous n'ajoutons pas **-LGL** oublié par **pkg-config --libs glu**. Avec cette version mise à jour de **simavr**, l'émulation du microcontrôleur par :

```
simavr --freq 16000000 --mcu atmega32u4 output/main.elf
```

se traduit par l'alternance de messages communiqués sur port série, et le changement d'état des LED indiqué par le changement d'état du registre contrôlant le port GPIO correspondant, tout fonctionne donc bien pour cet exemple basique (Figure 2).

```
OBJCOPY output/main.bin
avr-objcopy -Obinary output/main.elf output/main.bin
OBJDUMP output/main.list
avr-objdump -S output/main.elf > output/main.lst
SIZE output/main.elf
avr-size output/main.elf
```

text	data	bss	dec	hex	filename
14720	80	138	14938	3a5a	output/main.elf

```
jmfriedt@rugged:$ make -f Makefile.avr32u4 simu
simavr --freq 16000000 --mcu atmega32u4 output/main.elf
Loaded 14720 bytes at 0
Loaded 80 bytes at 800100
** DDRB(24) = 20
** DDRE(2d) = 40
** PORTB(25) = 20
.
Hello World..
** PORTE(2e) = 40
** PORTB(25) = 00
** PORTB(25) = 20
** PORTE(2e) = 00
** PORTB(25) = 00
** PORTB(25) = 20
.
Hello World..
** PORTB(25) = 00
** PORTE(2e) = 40
** PORTB(25) = 20
** PORTB(25) = 00
** PORTE(2e) = 00
** PORTB(25) = 20
.
Hello World..
```

Figure 2 : L'émulateur **simavr** exécutant FreeRTOS avec 3 tâches, deux faisant clignoter les LED sur le port B et E de la carte Olimexino-32U4, et la troisième communiquant un message sur le port de communication asynchrone. Le changement d'état des GPIO des ports B et E est indiqué dans le terminal par la valeur prise par le registre définissant l'état du port.

4. DÉPASSEMENT DE PILE

FreeRTOS gère la mémoire de façon un peu originale, car alloue sa propre zone mémoire de la taille définie par la constante `configTOTAL_HEAP_SIZE` dans laquelle il pioche l'emplacement des piles allouées à chaque tâche (oui, c'est bien cela... FreeRTOS pioche des emplacements de piles dans une zone mémoire qualifiée de tas !). Une fois cela compris, nous allouons dans `src/FreeRTOSConfig.h` une taille inférieure à la taille de la RAM de l'Atmega32U4 – nous avons pour notre part :

```
#define configTOTAL_HEAP_SIZE ( ( size_t ) ( 2* 1024 ) )
```

et chaque initialisation de tâche de la forme :

```
xTaskCreate( vPrintUart, (signed char*) "Uart", 64, NULL, 3, NULL);
```

peut prendre un peu de mémoire, ici 64 octets (4e argument d'après <https://www.freertos.org/a00125.html>), pour y stocker ses variables locales. Un problème classique en développement de systèmes embarqués est le dépassement de capacité de la pile, quand le système s'est vu allouer trop peu de mémoire pour stocker ses variables et finit soit par écrire en dehors de la RAM, soit dans la zone mémoire allouée à une autre tâche. FreeRTOS fournit un mécanisme intéressant pour détecter ce type de problème et en informer le développeur, à défaut d'y remédier. Ainsi, d'après <https://www.freertos.org/Stacks-and-stack-overflow-checking.html>, activer l'option `configCHECK_FOR_STACK_OVERFLOW` dans `src/FreeRTOSConfig.h` en lui assignant la valeur 1 ou 2 permet d'appeler la procédure `vApplicationStackOverflowHook()` en cas de détection de dépassement ou de corruption de la pile d'une tâche. Ici encore, ce mécanisme fonctionne fort bien sur Atmega32U4 : si nous modifions le programme d'affichage sur le port série en ajoutant l'allocation d'un tableau `c` de 256 octets et son assignation par la fonction `sprintf()` tel que :

```
void vPrintUart(void* dummy)
{char c[256];
 portTickType last_wakeup_time;
 last_wakeup_time = xTaskGetTickCount();
 while(1)
 {sprintf(c, "\nHello World\r\n");
  uart_puts(c);
  vTaskDelayUntil(&last_wakeup_time, 500/portTICK_RATE_MS);
 }
}
```

et ajoutons un gestionnaire appelé en cas de détection de dépassement de pile :

```
void vApplicationStackOverflowHook(TaskHandle_t xTask, char *pTaskName)
{while(1)
 {uart_puts("\r\nStack error: ");
  uart_puts(pTaskName);
  vTaskDelay(301/portTICK_RATE_MS);
 }
}
```


alors l'exécution du programme se traduit par :

```
$ simavr --freq 16000000 --mcu atmega32u4 output/main.elf
Loaded 12178 bytes at 0
Loaded 62 bytes at 800100
** DDRB(24) = 20
** DDRE(2d) = 40
** PORTB(25) = 20
.
Hello World..
..
Stack error: Uart..
Stack error: Uart..
```

qui aidera le développeur à identifier une cause de dysfonctionnement du programme, voire éventuellement d'agir en conséquence lorsque le système embarqué passerait sinon dans un état instable lors de la corruption du contenu de la pile. Ici, le problème vient de ce que la création de la tâche n'avait alloué que 64 octets sur la pile de `vPrintUart()` alors que nous allouons 256 octets au tableau `c`. Il faut donc augmenter la taille de la pile allouée à `vPrintUart()` lors de l'appel à `xTaskCreate()` en gardant un peu de marge par rapport aux 256 octets prévus pour laisser de la place aux variables implicitement allouées par FreeRTOS pour gérer la tâche. Cet exemple met par ailleurs en évidence l'énorme consommation de ressources de `stdio` puisque c'est bien l'utilisation de `sprintf()` qui induit le dépassement de capacité de la pile de la fonction affichant sur le port série. Il suffit de lire <https://github.com/avrdudes/avr-libc/blob/main/libc/stdio/vfprintf.c> pour se convaincre du grand nombre de tableaux et variables intermédiaires créés par cette fonction quand souvent une procédure ciselée pour n'afficher qu'un type de variable (`char`, `short` ou `long`) sera plus efficace et moins gourmande en ressources que la fonction générique.

5. AFFICHAGE DE LA LISTE DES TÂCHES : LE MALHEUR DU MALLOC()

Finalement, FreeRTOS propose une fonctionnalité sympathique d'affichage de la liste des tâches selon le nom fourni en 2e argument de `xTaskCreate()` et leur statut au sein de l'ordonnanceur ainsi que l'occupation de leur pile respective. Nous désirons donc tester cette fonctionnalité sur Atmega32U4 afin d'afficher une liste de tâches qui n'est pas sans rappeler la sortie de `ps aux` sous GNU/Linux. Pour ce faire (<https://www.freertos.org/a00021.html#vTaskList>), nous activons les options `configUSE_TRACE_FACILITY` et `configUSE_STATS_FORMATTING_FUNCTIONS` en les définissant à 1 dans `src/FreeRTOSConfig.h` et faisons appel à `vTaskList(char *)` qui remplit un tableau de caractères du message à afficher, par exemple au moyen de `uart_puts(char *)`. Après compilation, nous exécutons sur émulateur ou sur microcontrôleur, pour constater l'échec : rien ne s'affiche, et pourtant les LED clignotent. L'émulateur `simavr` indique par ailleurs qu'un unique caractère est transmis par le port série, mais en aucun cas la liste des tâches et leurs attributs.

Puisque les autres fonctions ont opéré comme prévu, le problème vient de l'implémentation de `vTaskList()` que nous trouvons dans <https://github.com/FreeRTOS/FreeRTOS-Kernel/blob/main/tasks.c#L4433> avec :

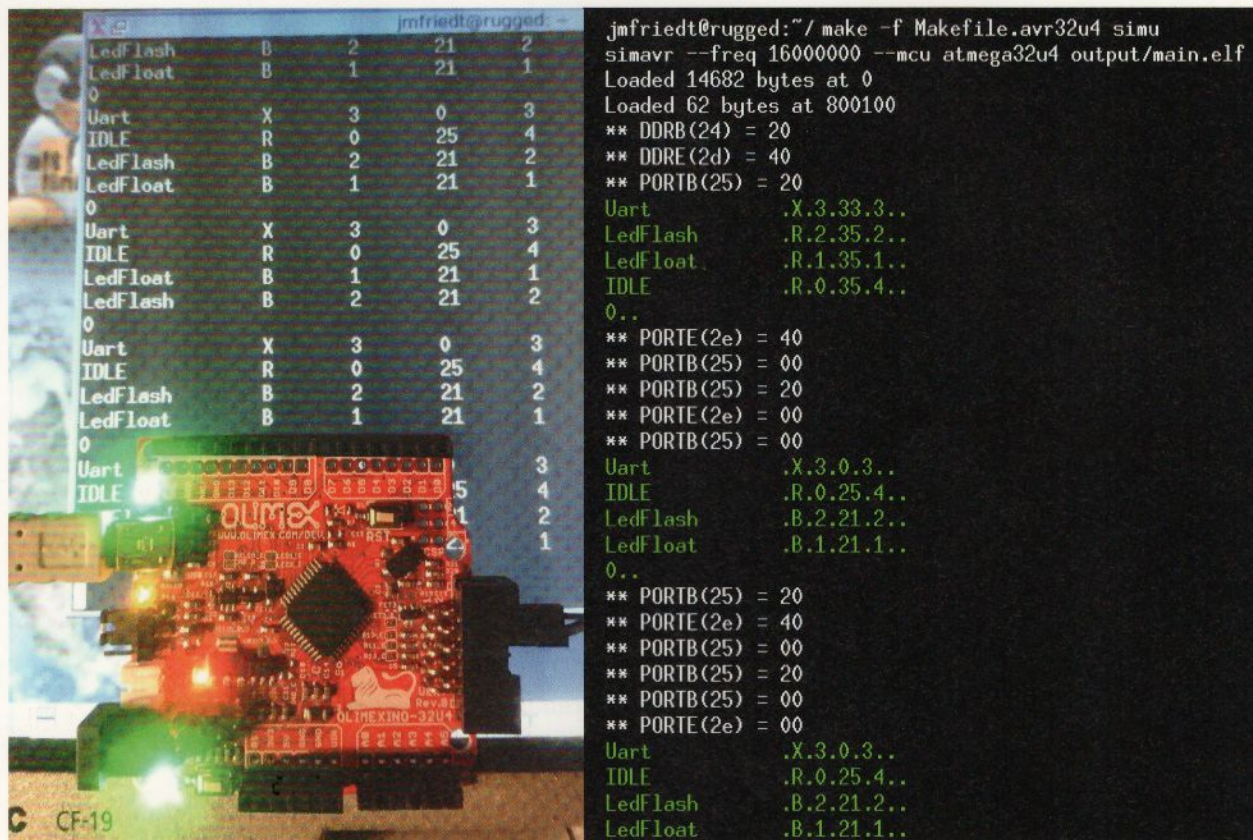
```
TaskStatus_t * pxTaskStatusArray;
UBaseType_t uxArraySize, x;
*pcWriteBuffer = ( char ) 0x00;
uxArraySize = uxCurrentNumberOfTasks;
pxTaskStatusArray = pvPortMalloc( uxCurrentNumberOfTasks * sizeof( TaskStatus_t ) );
if( pxTaskStatusArray != NULL )
{...}
else
{mtCOVERAGE_TEST_MARKER();}
```

et là, nous constatons avec désespoir que FreeRTOS exploite une allocation dynamique de mémoire. Nous savons bien [6] que l'allocation dynamique de mémoire est interdite sur système embarqué : avec une latence non déterministe, une structure de données mémorisant les blocs mémoire alloués en vue de les libérer [7], voire une inefficacité intrinsèque à la conception du code statique qui manipule le plus souvent des structures de données de taille connue à la compilation, le couple `malloc/free` apprécié sur système d'exploitation allouant dynamiquement des ressources n'a pas lieu d'être sur système embarqué, au point d'en être proscrit par la norme MISRA [8]. Mais si FreeRTOS fait le choix d'utiliser l'allocation dynamique de mémoire, essayons au moins de comprendre pourquoi elle échoue, et de la réparer.

Il est connu que FreeRTOS propose plusieurs modèles de gestion du tas : <https://www.freertos.org/a00111.html> explique la différence entre les diverses implémentations de `FreeRTOS/FreeRTOS-Kernel/tree/main/portable/MemMang` (un répertoire dans `portable` qui est pourtant générique !) et en particulier `heap_3.c` qui s'appuie sur `malloc/free` de la bibliothèque implémentant les fonctions C (`avr-libc` dans notre cas) contre `heap_2.c` qui implémente sa propre gestion du pointeur de tas et d'allocation/libération de ressources. Par défaut, <https://github.com/feilipu/miniAVRfreeRTOS> a fait le choix de `heap_3`, et bien entendu, l'implémentation de `malloc` dans `avr-libc` est cassée (ou tout au moins nécessite une initialisation qui n'est pas faite par défaut, voir ci-dessous). Remplacer `heap_3.c` par `heap_2.c` dans le `Makefile` résout en effet le problème, et l'affichage de la liste de tâches devient fonctionnel, sous émulateur comme sur plateforme matérielle (Figure 3).

La solution n'est pas complètement satisfaisante : changer de modèle de gestion de la mémoire pour arriver à ses fins est pour le moins une méthode de programmation scabreuse. Suite à réclamation auprès de l'auteur du dépôt GitHub original dont nous nous inspirons, nous apprenons [9] que le pointeur de tas n'est pas correctement initialisé par `avr-libc` pour répondre à son modèle de gestion de la mémoire [10]. Ainsi, [9] préconise d'initialiser le pointeur de tas par `__malloc_heap_end = (char *) (RAMEND - __malloc_margin)` avant toute utilisation de l'allocation dynamique de

– FreeRTOS dans 2,5 KB de RAM sur Atmega32U4 –



mémoire, les constantes étant définies dans `stdlib.h` qu'il faut inclure au préalable. En effet, après cette modification, l'allocation dynamique de mémoire par `malloc` fourni par `avr-libc` appelé par `heap_3.c` devient fonctionnelle.

CONCLUSION

Nous avons exploré la compatibilité de FreeRTOS avec une cible munie d'une très faible quantité de mémoire – 2,5 KB – pour constater que le système est fonctionnel avec l'utilisation d'une centaine d'octets par tâche, laissant présager la capacité à exécuter quelques dizaines de tâches en plus de l'ordonnanceur et sa tâche de veille *idle*. Ainsi, le développeur pourra à moindres frais, voire sans frais avec un émulateur, se familiariser avec les préceptes du développement sous FreeRTOS avant d'aborder des cibles plus ambitieuses et plus complexes à maîtriser. Nous constatons néanmoins qu'une compréhension détaillée du fonctionnement du compilateur **avr-gcc** et surtout de l'éditeur de liens **avr-ld** et de la gestion et l'allocation

Figure 3 : Gauche, le microcontrôleur exécutant FreeRTOS avec 3 tâches, deux faisant clignoter les LED orange et verte de la carte Olimexino-32U4, et la troisième communiquant la liste des tâches sur le port de communication asynchrone. Droite, le même code exécuté sur émulateur simavr. Dans ce second cas, le changement d'état des GPIO est indiqué dans le terminal par la valeur prise par le registre définissant l'état du port.

de mémoire lors de l'exécution du programme reste nécessaire lors de l'ajout de l'abstraction de l'environnement exécutif pour identifier et pallier les causes potentielles de dysfonctionnement.

L'ensemble des codes sources discutés dans cet article est disponible à https://github.com/jmfriedt/tp_freertos/ et en particulier les sous-répertoires **lbasic** pour l'applicatif et **Atmega32U4** pour le support de l'Atmega32U4 dans FreeRTOS. **JMF**

RÉFÉRENCES

- [1] A. Filiz, FreeRTOS introduction, FOSDEM 2015 à :
<https://archive.fosdem.org/2015/schedule/event/freertos/> et R. Barry, FreeRTOS on RISC-V, FOSEM 2019 à :
<https://archive.fosdem.org/2019/schedule/event/riscvfreertos/>
- [2] Documentation technique de l'Atmega32U4 à :
https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7766-8-bit-AVR-ATmega16U4-32U4_Datasheet.pdf
- [3] Q. Macé, J.-M Friedt, « *FreeRTOS : application à la réalisation d'un analyseur de réseau numérique sur STM32* », GNU/Linux Magazine France 207 (sept. 2017).
- [4] J.-M Friedt, « *Intercorrélation par transformée de Fourier rapide sur microcontrôleur sous FreeRTOS, et les pointeurs de pointeurs* », Hackable 43 (juill.-août. 2022).
- [5] J.-M Friedt, « *Émulation d'un circuit comportant un processeur Atmel avec simavr* », Hackable 34 (juill.-sept. 2020).
- [6] Al Williams, Embedded Memory Allocation, Dr. Dobbs's Journal (13 octobre 2014) reproduit à :
<https://rosetta.vn/short/2017/05/12/embedded-memory-allocation-dr-dobbs/>
- [7] Memory Allocation Hooks à :
https://www.gnu.org/software/libc/manual/html_node/Hooks-for-Malloc.html
- [8] Norme de programmation de la Motor Industry Software Reliability Association à :
<https://www.misra.org.uk/app/uploads/2021/06/MISRA-C-2012-AMD2.pdf>
qui indique clairement "AMD2.78: The identifiers `calloc`, `malloc`, `realloc`, `aligned_alloc` and `free` shall not be used and no macro with one of these names shall be expanded."
- [9] <https://github.com/feilipu/miniAVRfreeRTOS/issues/7>
- [10] Memory Areas and Using `malloc()` à :
<https://www.nongnu.org/avr-libc/user-manual/malloc.html>

EXPLOITATION DES SIGNAUX DE RÉFÉRENCE DE NAVIGATION PAR SATELLITE POUR UN POSITIONNEMENT CENTIMÉTRIQUE : RTKLIB FAIT APPEL À CENTIPÈDE ET L'IGN POUR AFFICHER DANS QGIS

J.-M Friedt

[FEMTO-ST département temps-fréquence, Besançon]

Nous exploitons un récepteur de navigation par satellites « faible » coût U-Blox Zed-F9P s'appuyant sur divers réseaux de stations de référence (Centipède, IGN) pour utiliser efficacement la bibliothèque RTKLIB pour positionner un dispositif mobile avec une résolution centimétrique. Les informations ainsi produites sont exportées en temps réel dans des Systèmes d'Informations Géographiques (GIS) tels que QGIS (GNU/Linux) ou QField (Android) pour être intégrées dans l'ensemble des informations géoréférencées considérées au cours d'une étude sur le terrain.



Nous avons récemment discuté [1] de la mise en œuvre de récepteurs de signaux de navigation par satellite (GNSS) en vue d'un positionnement centimétrique, et en avons profité pour développer un système de communication radiofréquence longue portée s'appuyant sur le protocole LoRa à 868 MHz afin de s'affranchir de restrictions sur les émissions radiofréquences dans les bandes de téléphonie mobile ou celles industrielles, scientifiques et médicales (ISM) autour de 2,45 et 5,8 GHz sur notre site d'étude. Ce faisant, nous avons découvert le projet Centipède qui vise à produire un maillage dense de stations GNSS de référence (dites stations de base) en vue de permettre un positionnement centimétrique de dispositifs mobiles (dits *rovers*) en s'affranchissant des délais ionosphériques et autres effets corrélés entre station de base et *rover*. En effet, le délai ionosphérique, variable au cours du temps et de l'activité solaire, est la principale source d'incertitude du temps de vol des ondes électromagnétiques depuis les satellites de navigation en orbite moyenne (20 000 km de la surface de la Terre) vers le récepteur

au sol, induisant une incertitude de position de quelques mètres (pour rappel, 1 ns de retard se traduit par 30 cm d'erreur de trilatération du récepteur). En imposant la position d'une station de base supposée connue, un *rover* suffisamment proche – typiquement dans un rayon d'une trentaine de kilomètres – subira les mêmes délais ionosphériques que la station de base, effet qui pourra donc être corrigé par un logiciel approprié tel que RTKLIB. Centipède dissémine ces informations de stations de base par réseau compatible Internet (IP), accessible en particulier par le réseau de téléphonie mobile, moyennant un abonnement relativement courant actuellement.

Nous allons explorer trois utilisations des connaissances acquises pour le positionnement centimétrique de GNSS exploitant la bibliothèque libre RTKLIB et sa déclinaison pour récepteurs faible coût Demo5 de <https://github.com/rinex20/RTKLIB-demo5> : la configuration et l'utilisation de RTKLIB en console, s'affranchissant ainsi des déboires des interfaces graphiques pour rendre l'application compatible avec les systèmes embarqués à faible empreinte mémoire et ressources de calcul, l'utilisation des signaux communiqués par Centipède depuis la station de référence la plus proche du site d'observation et une comparaison avec les résultats issus du post-traitement en exploitant les signaux de référence communiqués par l'Institut Géographique National (IGN), et finalement la cartographie en temps réel des signaux GNSS centimétriques dans l'outil de gestion des informations spatialisées QGIS.

Nous précisons dès le début de cet exposé que l'objectif n'est pas une résolution centimétrique d'un objet en mouvement, mais une géolocalisation avec une telle résolution après observation des constellations de satellites pendant plusieurs minutes, voire dizaines de minutes. La solution convergera petit à petit en passant de *Single* (mesure simple récepteur) à *Float* (en cours de recherche de la solution) à *Fixed* (solution obtenue). Le passage de *Float* à *Fixed* peut être long, voire ne jamais arriver tel que nous l'avons constaté sous certains couverts forestiers.

Nous avons dans la description précédente [1] expliqué comment configurer deux U-Blox Zed-9FP montés sur le circuit imprimé commercialisé par MikroE sous la référence 4456 au moyen du logiciel propriétaire U-Center, mais exécutable au moyen de l'émulateur Wine

Pour cette étude, mais surtout par conviction qu'un institut de recherche public (FEMTO-ST) a pour vocation de fournir un service au public, nous avons installé une station de référence Centipède qui servira, à côté de la station de référence de l'Institut National de l'information Géographique et forestière (IGN – il faut chercher les bonnes lettres dans le nom actuel pour justifier de l'ancien acronyme) de l'observatoire de Besançon, de *basestation* par rapport à laquelle localiser le *rover*. L'installation est illustrée ci-dessous en haut, avec une antenne Septentrio PolaNt-x MF multiconstellations sur le toit d'un bâtiment de l'École Nationale Supérieure de Mécanique et des Microtechniques (ENSMM) qui héberge le département Temps-Fréquence de l'institut FEMTO-ST, en vue dégagée du ciel, et en bas l'interface graphique des stations de Centipède identifiant les propriétés de notre station de base et celles environnantes. La documentation de Centipède (<https://docs.centipede.fr/docs/base/>) pour installer une station de base est impressionnante de clarté et de simplicité. L'identifiant de cette station de base est ENSMM, avec la figure de droite indiquant des cercles à des multiples de 10 km. Il est habituellement considéré que la correction RTK est efficace jusqu'à une trentaine de kilomètres de la station de base utilisée comme référence.



sous GNU/Linux, pour ne transmettre que les messages binaires au format UBX et ne pas émettre les messages NMEA. Ainsi, avec les trames RAWX communiquées en 115 200 bauds 8N1 pour toutes les constellations supportées (GPS, Galileo, Beidou et éventuellement GLONASS, même si ce dernier est plutôt à proscrire compte tenu de sa communication multiplexée en fréquence très différente des autres modes de communication différenciant chaque satellite sur le code), nous avons proposé d'implémenter une station fixe (*basestation*) communiquant avec une station mobile (*rover*) dont la position relative est identifiée au centimètre près. Nous allons poursuivre cette étude en nous appuyant cette fois sur des réseaux de *basestations* de référence aux positions connues avec une exactitude subcentimétrique (un point que nous avons omis d'aborder auparavant) en vue de localiser un *rover* avec une exactitude centimétrique en s'appuyant sur ce réseau de référence dont les informations sont transmises en temps réel, et accessibles notamment par réseau de téléphonie mobile ou Wi-Fi (Centipède) ou en post-traitement (IGN).

1. RTKLIB EN MODE CONSOLE

Moins de trois mois après avoir publié la méthode d'acquisition et de traitement de signaux centimétriques par récepteurs UBlox de la série 9 (Zed-F9P), les outils sont déjà cassés ! L'interface graphique de `rtknavi-qt` ne compile plus, et nous n'allons évidemment pas utiliser les binaires précompilés pour MS-Windows : afin de nous affranchir des affres de bibliothèques dont l'interface de programmation applicative (API) ne cesse de changer et de nous libérer d'interfaces graphiques illisibles sur un écran illuminé par le soleil, nous allons explorer les interfaces en ligne de commande de RTKLIB, plus stables, compatibles avec les applications embarquées ou avec une automatisation des acquisitions et des traitements.

Les outils en ligne de commande se compilent sans surprise (testé sous Debian GNU/Linux) en exécutant `make` depuis le répertoire `app/consapp` de `RTKLIB-demo5`. Notez que la *cross-compilation* vers une cible embarquée d'architecture autre que l'hôte, telle que le processeur ARM 64 bits de la Raspberry Pi 4 dont nous supposons l'environnement de développement produit par Buildroot [2] s'obtient simplement en commentant dans le `Makefile` la ligne `export CC = gcc`, puis en lançant la compilation par `CC=aarch64-linux-gcc make` en ayant pris soin d'ajouter à son `PATH` le répertoire `output/host/usr/bin` du Buildroot configuré pour la cible (p. ex. `make raspberrypi4_64_defconfig && make` depuis Buildroot). Les binaires se regroupent dans un unique répertoire, par exemple `/tmp/bin` prêt à expédier sur la cible, en commentant dans le `Makefile` la ligne `BINDIR = /usr/local/bin`, puis en exécutant `mkdir -p /tmp/bin && BINDIR=/tmp/bin make install` afin de se contenter de copier le contenu de `/tmp/bin` vers la carte SD de la Raspberry Pi 4. Lors de la connexion du récepteur U-Blox Zed-9FP, nous penserons à charger le pilote du port série virtuel sur bus USB `modprobe cdc_acm`, puisqu'il n'est pas chargé automatiquement.

Nous apprenons dans la documentation de RTKLIB [3] que la commande `rtkrcv -s -o RTKlib_dualZedF9P.conf` fait appel au logiciel approprié pour traiter un flux de mesures portant d'une part les informations de la station de référence, et d'autre part les observations du *rover*, selon un fichier de configuration un peu volumineux et disponible à https://github.com/jmfriedt/RIOT_NyAlesund pour être lancé dès son chargement (option `-s`), mais dont les principales informations sont :


```
pos1-posmode      =kinematic
pos1-frequency    =l1+l2
...
inpstr1-type      =serial
inpstr2-type      =serial
inpstr3-type      =off
inpstr1-path      =ttyACM0:115200:8:n:1:off
inpstr2-path      =ttyUSB3:115200:8:n:1:off
inpstr1-format    =ubx
inpstr2-format    =ubx
...
outstr1-type      =file
outstr2-type      =file
outstr1-path      =/tmp/1
outstr2-path      =/tmp/2
outstr1-format    =enu
outstr2-format    =llh
logstr1-type      =file
logstr2-type      =file
logstr3-type      =off
logstr1-path      =/tmp/rover.log
logstr2-path      =/tmp/base.log
...
```

donc deux flux venant sur les ports série `/dev/ttyACM0` (UBlox servant de rover) et `/dev/ttyUSB3` (sortie du flux de la station de base communiqué par LoRa), au débit de 115 200 bauds, et au format UBX binaire (UBX). Les sorties sont d'une part des données issues des traitements en divers formats (latitude/longitude/hauteur ou XYZ dans un repère d'origine le centre de la Terre – on vérifie que la racine de la somme des carrés est bien de l'ordre du rayon terrestre [4] de $40\,000/(2\pi) = 6366$ km), ainsi que le stockage des mesures brutes acquises par le rover et transmises par la station de base. Le résultat de ces traitements lorsque rover et station de base sont en vue du ciel (Fig. 1 ci-contre) est de la forme :

% GPST	x-ecef(m)	y-ecef(m)	z-ecef(m)	Q	ns	sdx(m)	sdym(m)	sdz(m)
2023/02/11 12:58:28.000	4313722.1776	452854.1904	4661052.7555	2	11	2.6268	1.1727	1.9213
...								
2023/02/11 13:04:13.000	4313719.6932	452855.1056	4661050.6927	2	16	0.0467	0.0312	0.0141
2023/02/11 13:04:14.000	4313719.6938	452855.1075	4661050.6920	2	16	0.0465	0.0311	0.0140
2023/02/11 13:04:15.000	4313719.1227	452854.8761	4661050.5032	1	16	0.0065	0.0029	0.0057
2023/02/11 13:04:16.000	4313719.1237	452854.8776	4661050.5013	1	16	0.0065	0.0029	0.0057
2023/02/11 13:04:17.000	4313719.1255	452854.8769	4661050.5040	1	16	0.0065	0.0029	0.0056
2023/02/11 13:04:18.000	4313719.1206	452854.8745	4661050.5025	1	16	0.0065	0.0029	0.0057
...								

avec dans l'ordre le temps GPS, la position dans un repère centré sur le centre de la Terre ECEF, une qualité de mesure Q, un nombre de satellites visible ns et un écart type sur la mesure (donc une résolution) sdx, sdy et sdz pour les trois directions de l'espace. Nous avons tronqué cet affichage des autres informations redondantes par souci de compacité.

– ... RTKLIB fait appel à Centipède et l'IGN pour afficher dans QGIS –

La documentation de RTKLIB à la page 104 de https://www.rtklib.com/prog/manual_2.4.2.pdf nous aide à interpréter le statut de l'analyse en fournissant la nature de la solution dans la colonne Q :

The flag which indicates the solution quality.

- 1 : Fixed, solution by carrier-based relative positioning and the integer ambiguity is properly resolved.
- 2 : Float, solution by carrier-based relative positioning but the integer ambiguity is not resolved.
- 3 : Reserved
- 4 : DGPS, solution by code-based DGPS solutions or single point positioning with SBAS corrections
- 5 : Single, solution by single point positioning

Ainsi, une solution de type 1 est garante de la position avec une résolution subcentimétrique et son obtention sur le terrain permet de s'assurer de la qualité de la mesure en imposant d'observer suffisamment longtemps pour que la solution converge, parfois un peu pénible sous la pluie ou dans la neige.

Figure 1 : Contexte expérimental, un rover (mobile) est connecté à un ordinateur exécutant les outils de RTKLIB pour s'affranchir des sources corrélées de biais lors du positionnement par satellite, en recevant un signal de référence soit d'une station de base communiquant par LoRa, soit d'un récepteur Centipède dont les informations sont diffusées par protocole compatible Internet IP.

2. EXPLOITATION DES SIGNAUX DE RÉFÉRENCE DE CENTIPÈDE

La documentation de Centipède explique fort bien comment utiliser les informations diffusées et fournit à https://docs.centipede.fr/docs/Rover_rtklib_pc/RTKlib_windows.html un fichier d'exemple de configuration. Nous nous inspirons de ce document pour d'une part fournir un flux venant du port série connecté au récepteur Zed-F9P – le même qui a servi de rover auparavant – mais cette fois la station de base n'est pas un flux du port série, mais un flux IP venant de la connexion Wi-Fi de



L'ordinateur effectuant les acquisitions et connecté au *caster* de Centipède diffusant ses informations sur le port 2101. Nous prenons soin de mentionner la station de référence qui nous intéresse, ici ENSMM puisque nous sommes à proximité de cette localisation ; ce fichier de configuration `RTKlib_centipede.conf` contient donc :

```
...
inpstr1-type      =serial
inpstr2-type      =ntripcli
inpstr3-type      =off
inpstr1-path      =ttyACM0:115200:8:n:1:off
inpstr2-path      =:@caster.centipede.fr:2101/ENSMM
inpstr1-format     =ubx
inpstr2-format     =rtcm3
...
outstr1-type      =file
outstr1-path       =./sortie.txt
outstr1-format     =xyz
...
```

pour être utilisé par `rtkrcv` au moyen de `rtkrcv -s -o RTKlib_centipede.conf`.

Le résultat du traitement en temps réel des observations est de la forme :

%	GPST	x-ecef(m)	y-ecef(m)	z-ecef(m)	Q	ns	sdx(m)	sdy(m)	sdz(m)
...									
2248	565873.000	4313714.4614	452855.6378	4661048.9718	1	8	0.0074	0.0041	0.0072
2248	565874.000	4313714.4521	452855.6252	4661048.9851	1	8	0.0074	0.0041	0.0072
2248	565875.000	4313714.4583	452855.6248	4661048.9842	1	8	0.0074	0.0041	0.0072
2248	565876.000	4313714.4483	452855.6108	4661049.0041	1	8	0.0074	0.0041	0.0072

avec une qualité Q de 1 qui indique une solution fixe et un écart type dans les trois directions inférieur au centimètre.

Nous observons un écart de 5 m entre la station de référence que nous avons fournie initialement et la solution fournie par Centipède. Dans la première solution, la position de la station de référence n'a pas été imposée et a été mesurée par une solution simple (non différentielle) : cette position fluctue, comme toute mesure GNSS sans référence, de +/- 7 m et un écart de 5 m n'est pas surprenant. Cette erreur pourrait être corrigée en plaçant notre station de base sur un point de position connue et en s'y référant lors de chaque campagne de mesures. Au contraire, la mise en place de la station de référence Centipède impose une période d'acquisition de sa position et son positionnement relativement au réseau de référence de l'IGN, garantissant la cohérence des positions. C'est justement cette position du réseau RGP (Réseau GNSS Permanent) de l'IGN que nous allons considérer ci-dessous pour du post-traitement.

– ... RTKLIB fait appel à Centipède et l'IGN pour afficher dans QGIS –

Notons en cas de difficulté à se connecter une commande très pratique de **rtkrcv** qu'est **stream** qui annonce la nature des flux reçus par RTKLIB :

```
rtkrcv> stream
Stream      Type  Fmt  S  In-byte  In-bps  Out-byte  Out-bps  Path                                Message
input rover serial ubx  C  607740  10804      0      0 ttyACM0:115200:8:n:1:off /dev/ttyACM0
input base  ntrips rcm3 C  657002  10327      0      0 :@caster.centipede.fr:21 caster.
centipede.fr/ENSM
...
```

prouve que le port série du récepteur mobile (**ttyACM0**) a été détecté et le flux en provenant de Centipède (<http://caster.centipede.fr>) est reçu. En cas d'échec de connexion à Internet, nous aurions un message du type :

```
rtkrcv> stream
Stream      Type  Fmt  S  In-byte  In-bps  Out-byte  Out-bps  Path                                Message
input rover serial ubx  C    17592  10402      0      0 ttyACM0:115200:8:n:1:off /dev/ttyACM0
input base  ntrips rcm3 C      0      0      0      0 :@caster.centipede.fr:21 address error
(caster.centipede.fr)
...
```

Par ailleurs, la commande satellite de **rtkrcv** permet de voir les constellations de satellites vu leur statut :

```
rtkrcv> satellite
SAT C1  Az  El  L1 L2  Fix1  Fix2  P1Res  P2Res  L1Res  L2Res  Sl1  Sl2  Lock1  Lock2  Rj1  Rj2
G07 - 292.4 10.5 - - - - 0.000-24.794 0.0000 0.0000 27 15 0 0 0 0
G08 OK 301.2 52.1 OK OK HOLD HOLD -0.342 -0.587 -0.0063 -0.0013 1 1 361 361 0 0
G10 OK 117.9 60.1 OK OK HOLD HOLD 0.046 -0.669 -0.0022 -0.0030 1 0 361 361 0 0
G15 - 31.7 5.9 - - - - 0.000 0.000 0.0000 0.0000 0 0 0 0 0 0
G16 OK 193.3 48.1 OK - HOLD - 0.952 0.000 -0.0016 0.0000 0 0 361 0 0 0
G18 OK 63.4 17.5 OK OK HOLD HOLD -1.008 -1.523 0.0012 -0.0064 0 0 361 361 0 1
G21 OK 255.0 24.7 OK - HOLD - -1.122 0.000 -0.0118 0.0000 1 0 361 0 0 0
G23 OK 61.4 44.1 OK OK HOLD HOLD -0.481 -0.934 -0.0008 0.0040 1 0 361 361 0 0
G27 OK 22.8 85.5 OK OK HOLD HOLD 0.000 0.000 0.0000 0.0000 0 1 361 361 0 0
E08 OK 40.6 11.8 OK OK FLOAT FLOAT 1.210 -0.735 0.0045 -0.0063 0 0 0 0 0 0
E12 OK 252.6 14.3 - - - - -9.481 -2.435 0.0000 0.0000 3 6 0 0 0 0
E24 OK 319.3 19.5 - - - - -26.056 1.688 0.0000 0.0000 12 7 0 64 0 0
E26 OK 55.5 57.3 OK OK HOLD HOLD 0.135 0.175 0.0038 -0.0045 0 0 360 361 0 0
E31 OK 271.2 53.0 OK OK HOLD HOLD 2.000 0.696 0.0064 0.0127 0 0 360 361 0 0
E33 OK 270.2 66.1 OK OK HOLD HOLD 0.000 0.000 0.0000 0.0000 0 0 360 361 0 0
```

et de s'assurer par exemple que l'antenne est bien connectée et alimentée.

3. EXPLOITATION DES SIGNAUX DE RÉFÉRENCE DE L'IGN

Figure 2 : Interface de téléchargement des signaux de référence mis à disposition par l'IGN. Comme pour Centipède, on choisira une station de référence aussi proche que possible du site d'observation, ici l'observatoire de Besançon à quelques centaines de mètres du site de mesure. Les observations de l'IGN sont disponibles environ 1 h après leur acquisition et ne permettent donc pas une analyse en temps réel comme le propose Centipède.

Le site de l'IGN qui propose l'accès à leurs observations de référence de signaux de navigation par satellites – <https://rgp.ign.fr/DONNEES/diffusion/> – se configure en sélectionnant la station de référence aussi proche que possible du site d'observation, pour nous **station BSCN** -> **flèche+** pour ajouter la station, activer GPS C2 et Galileo, sélectionner la plage horaire autour de l'observation, 1 s de pas d'échantillonnage, et le format Rinex2.11 (Fig. 2). Une fois ces configurations achevées et validées par **Valider** les filtres, **Ajouter** au panier et **Télécharger**. On appréciera la qualité de ces données en sachant qu'une ligne téléphonique dédiée est installée à l'observatoire de Besançon pour collecter et communiquer ces mesures.

The screenshot shows the 'Diffusion des données' interface on the website <https://rgp.ign.fr/DONNEES/diffusion/>. The interface is divided into three main sections: RECHERCHE, AFFICHAGE ET SÉLECTION, and PANIER.

- RECHERCHE (Left Panel):**
 - 1** **Valider les filtres >>** (Red button)
 - PÉRIODE:**
 - Début: 11/02/2023
 - Heures: de 13h à 14h
 - TU (selected), Heure locale
 - Fin: 11/02/2023
 - TYPE DE DONNÉES *:**
 - ☒ GPS (obligatoire) toutes les observables avec les suivantes:
 - ☒ C2 ☐ L5
 - ☐ GLONASS toutes les observables
 - ☒ GALILEO toutes les observables
 - Echantillonnage:**
 - ☒ 1s ☐ 5s
 - ☐ 10s ☐ 15s ☐ 30s
 - Version Rinex: ☐ 2.10 ☒ 2.11
 - AFFICHAGE CARTE:**
 - Voir la légende
- AFFICHAGE ET SÉLECTION (Center):**
 - 5** **Outils de sélection** (Red circle around the selection tools)
 - 6** **Ajouter au panier** (Red circle around the 'Ajouter au panier' button)
 - Map showing the selected station **BSCN** near **BESANÇON**.
- PANIER (Right Panel):**
 - 7** **Télécharger** (Red circle around the 'Télécharger' button)
 - Avancement du traitement
 - Summary: 11/2/2023 de 13h à 14h (TU), sur 1 jour BSCN

– ... RTKLIB fait appel à Centipède et l'IGN pour afficher dans QGIS –

Nous apprenons dans la page de manuel de **rnx2rtkp** <https://manpages.debian.org/testing/rtklib/rnx2rtkp.1.en.html> qui fait partie de RTKLIB que le post-traitement des fichiers au format d'échange standard RINEX est pris en charge par cet outil qui prend comme argument le nom du fichier d'observations du rover, le nom du fichier d'observations de la station de référence que nous venons d'obtenir auprès de l'IGN (extension **??o** avec **??** l'année de la mesure) ainsi que les éphémérides associées au moment de l'observation contenues dans le fichier d'extension **??n**. Nous ajoutons comme options le type de solution recherchée – ici un rover statique (**-p** parmi **0** : single, **1** : dgps, **2** : kinematic, **3** : static, **4** : moving-base, **5** : fixed, **6** : ppp-kinematic, **7** : ppp-static) et un masque d'élévation en dessous duquel les observations sont rejetées (**-m**). Comme **rnx2rtkp** n'accepte que des fichiers au format RINEX, nous devons convertir les enregistrements au format UBlox en RINEX au moyen de **convbin** fourni lui aussi par RTKLIB :

```
convbin -r ubx rover.log -o rover.obs
rnx2rtkp -p 5 -m 15 -e rover.obs telechargement_RGP_198748/recherche_1/bscn042z.23o \
telechargement_RGP_198748/recherche_1/bscn042z.23n
```

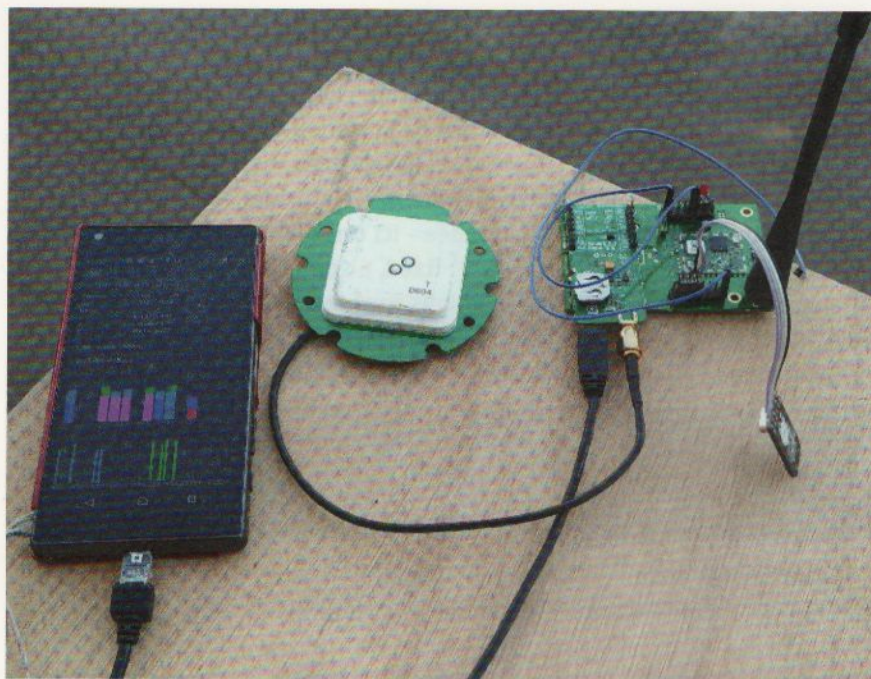
À l'issue de ce traitement, nous avons la satisfaction d'obtenir à nouveau des solutions subcentimétriques, cette fois en excellent accord avec la solution calculée avec la position de la station de référence Centipède :

% GPST	x-ecef(m)	y-ecef(m)	z-ecef(m)	Q	ns	sdx(m)	sdym(m)	sdz(m)
2248 565715.000	4313714.2435	452855.2018	4661049.2597	2	8	2.1218	1.1354	2.1171
...								
2248 565739.000	4313714.1418	452855.4004	4661048.8065	2	8	0.4437	0.2456	0.3715
2248 565740.000	4313714.1851	452855.4162	4661048.8337	2	8	0.4338	0.2407	0.3596
2248 565741.000	4313714.2165	452855.3601	4661048.8608	1	8	0.0075	0.0042	0.0072
2248 565742.000	4313714.2037	452855.3567	4661048.8472	1	8	0.0075	0.0042	0.0072
2248 565743.000	4313714.2199	452855.3657	4661048.8398	1	8	0.0075	0.0042	0.0072
2248 565744.000	4313714.2384	452855.3703	4661048.8627	1	8	0.0075	0.0042	0.0072
2248 565745.000	4313714.2335	452855.3763	4661048.8459	1	8	0.0075	0.0042	0.0072
...								

Certes, la solution est obtenue a posteriori, mais cette approche peut s'avérer fort utile sur des sites où une liaison téléphonique n'est pas disponible pour récupérer en temps réel les informations de référence de Centipède, au détriment de ne pas savoir avant post-traitement si la solution a convergé. On prendra donc soin d'attendre quelques dizaines de minutes sur chaque site de mesure pour enregistrer suffisamment de mesures pour maximiser les chances de convergence de la solution *Fixed*.

4. TRAITEMENT SUR TÉLÉPHONE MOBILE

RTKLIB existe pour Android sous le nom de RTKlibDroid disponible à <https://github.com/jancelin/RTKlibDroid> ou RtkGps+ à <https://github.com/jancelin/RtkGps> avec une archive APK disponible dans les *releases* du dépôt afin qu'il ne soit pas nécessaire de se prostituer auprès de Google pour installer l'application. C'est cette seconde option que nous avons testée en



configurant une source de rover venant du Zed-F9P connecté au téléphone mobile par câble USB-OTG – donc avec la broche ID du connecteur USB micro-B connectée à la masse – et la seconde source de station de base venant du caster Centipède, toujours en référant comme point de montage la station de référence la plus proche (dans notre cas ENSMM). Après quelques minutes en espace libre avec une bonne vue du ciel, la solution *Fixed* est obtenue et l'écart type sur la position chute sous le centimètre (Fig. 3). Dans le cas particulier du téléphone mobile Sony Xperia Z5 Compact utilisé au cours de ces essais, la détection du périphérique connecté au port USB nécessite une recherche explicite dans **Settings -> Device connection -> USB Connectivity -> Detect USB Device**. Une fois le port série virtuel identifié, RtkGps+ valide la lecture des données en faisant clignoter un des deux indicateurs verts en

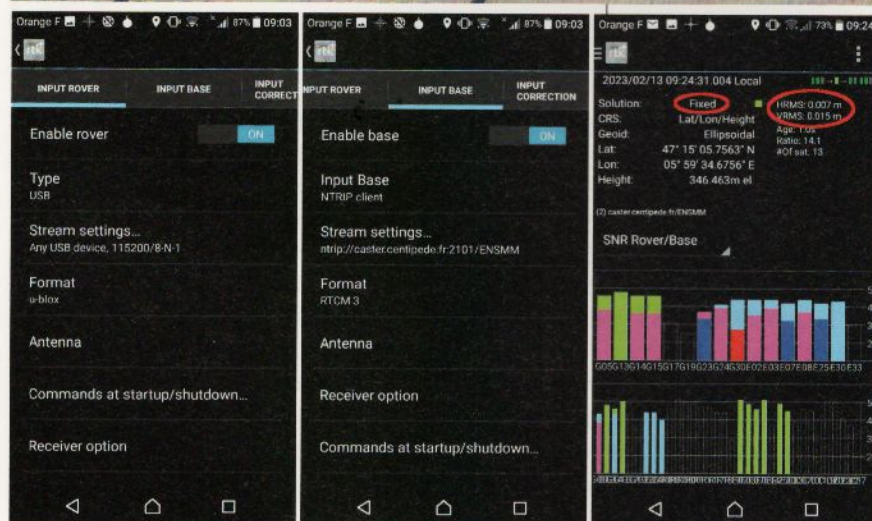


Figure 3 : En haut, montage expérimental avec le Zed-F9P connecté par câble USB-OTG au téléphone mobile. En bas, de gauche à droite, la configuration de la première source comme flux série asynchrone compatible RS232 au débit de 115 200 bauds en accord avec la configuration du récepteur GPS, la configuration de la seconde source connectée par lien de téléphonie mobile au caster Centipède, et le résultat du positionnement subcentimétrique.

haut à droite de l'interface graphique signifiant que les trames sont lues et traitées.

Obtenir une solution subcentimétrique sur téléphone mobile est bien, mais les utilisateurs argumenteront qu'ils veulent cette solution dans leur outil favori de gestion d'informations spatialisées, donc QField sous Android – la version embarquée de QGIS. Il faut donc transmettre les informations issues de RTKLIB vers QField en passant outre du récepteur GPS matériel

– ... RTKLIB fait appel à Centipède et l'IGN pour afficher dans QGIS –

équipant le téléphone mobile. Pour ce faire, nous exploiterons l'option d'injection de position en mode développeur sous Android : RTKLIB produit une information centimétrique au travers de ce protocole qui supplante la position déduite par le récepteur matériel et indique à QField sa position depuis le périphérique externe. Après avoir généré un projet QGIS trivial qui ne contient qu'une couche OpenStreetMaps et l'avoir importé dans QField,

nous pourrions valider avec une résolution centimétrique sur plateforme de calcul mobile les informations spatialisées qui nous intéressent (Fig. 4).

L'activation du mode d'injection de trames GNSS dans Android s'obtient en activant le mode développeur et en autorisant RtkGps+ à injecter les trames dans l'onglet **Select mock location app** (Fig. 4, gauche).

5. UTILISATION DE TÉLÉPHONES MOBILES ÉQUIPÉS DE RÉCEPTEURS MULTIBANDES

Certains téléphones portables sont équipés de récepteurs GNSS multiconstellations et multibandes tels que décrits à [5] et en particulier lorsqu'ils sont basés sur un *chipset* Qualcomm Snapdragon. Nous avons ainsi acquis un Xiaomi Mi 10 Lite muni de son Snapdragon 765G et tenté d'estimer la résolution de positionnement en post-traitement s'appuyant sur les données fournies par le récepteur de l'IGN à Nanterre.

Les mesures brutes du téléphone portable Xiaomi Mi 10 Lite sont enregistrées au moyen de l'application Geo++ RINEX Logger. La nature des constellations observées est précisée dans l'entête du fichier RINEX, avec *G* pour GPS américain, *R* pour GLONASS russe, *E* pour Galileo européen, *C* pour Beidou chinois et *J* pour QZSS japonais tandis que le type des observations suit avec *C* pour les *pseudorange* (temps de vol du code), *L* la phase de la porteuse radiofréquence, *D* le décalage Doppler de la porteuse dû au mouvement du satellite et *S* la puissance du signal. Nous constatons donc que le Xiaomi Mi 10 Lite observe les quatre grandes constellations en orbite moyenne (*G*, *R*, *E* et *C*) sur les bandes 1 et 5 de GPS et Galileo en code et en phase :

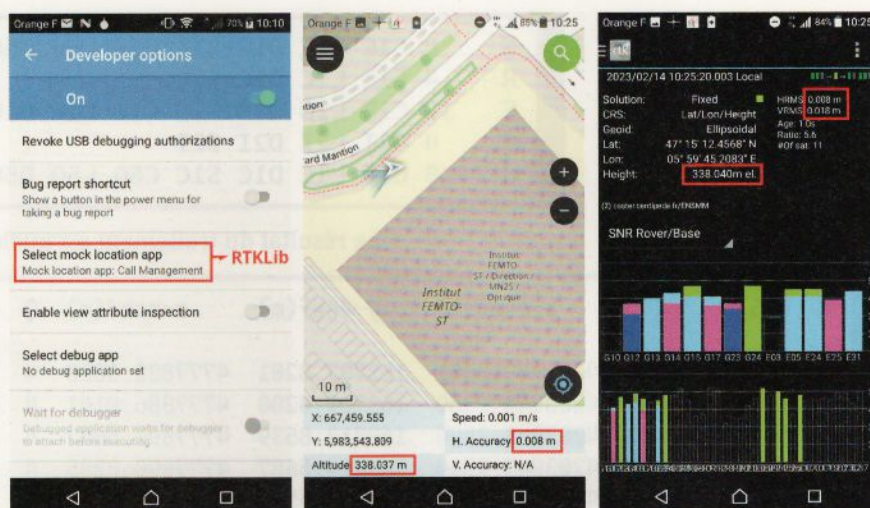


Figure 4 : De gauche à droite, activation du mode d'injection de position en mode développeur sous Android autorisant RTKLIB avec sa solution centimétrique à supplanter la solution du récepteur du téléphone ; la position du récepteur vue par QField avec une carte OpenStreetMaps indiquant un écart type subcentimétrique sur la position (encadré rouge) compatible avec la solution de RTKLIB (droite).


```
G 8 C1C L1C D1C S1C C5Q L5Q D5Q S5Q
R 4 C1C L1C D1C S1C
E 12 C1B L1B D1B S1B C1C L1C D1C S1C C5Q L5Q D5Q S5Q
C 4 C2I L2I D2I S2I
J 8 C1C L1C D1C S1C C5Q L5Q D5Q S5Q
```

Le résultat du traitement est quelque peu décevant avec :

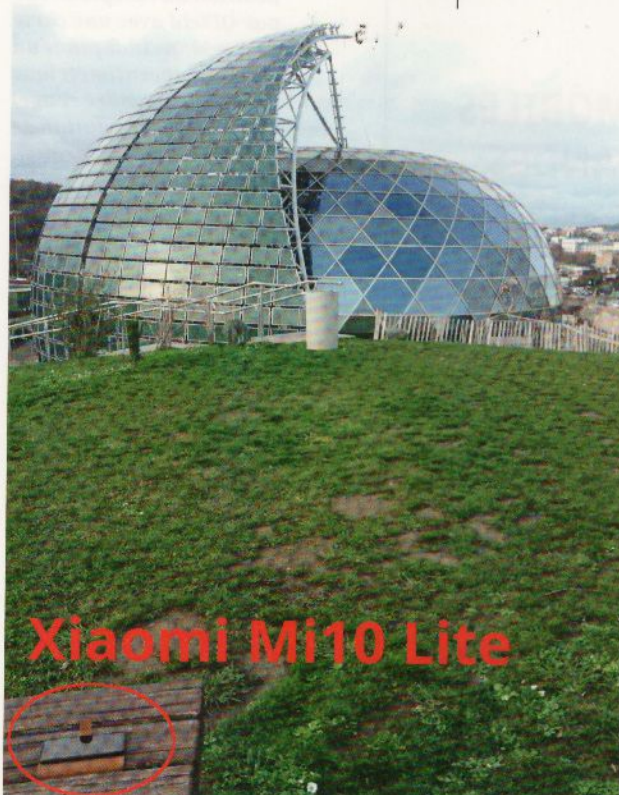
% GPST	x-ecef(m)	y-ecef(m)	z-ecef(m)	Q	ns	sdx(m)	sdym)	sdz(m)
...								
2138 311127.448	4203940.0881	163707.8201	4777882.5088	4	0	1.9344	1.1780	2.5586
2138 311128.448	4203944.2380	163709.4200	4777886.0141	4	0	1.8424	1.1506	2.5183
2138 311129.448	4203948.3544	163712.8539	4777891.5647	4	0	1.7981	1.1389	2.4892
2138 311130.448	4203947.9964	163714.5687	4777890.2285	4	0	1.7642	1.0951	2.2640
2138 311131.448	4203946.8551	163715.4582	4777888.3513	4	0	1.7400	1.0817	2.1458

Figure 5 : Enregistrement de données brutes GNSS par un téléphone mobile Xiaomi Mi 10 Lite sur l'île Seguin à l'ouest de Paris.

un écart type dans les trois directions de l'ordre du mètre. Nous avons bien été prévenus de la qualité médiocre des antennes de réception GNSS équipant les téléphones mobiles et placer le téléphone à quelques dizaines de centimètres du sol sur un banc est loin d'une condition idéale pour éviter les rebonds et interférences des chemins multiples, mais nous sommes loin du centimètre recherché (Fig 5).

6. INJECTION DES SIGNAUX GNSS DANS QGIS

Maintenant que nous sommes capables de traiter en temps réel des mesures acquises par le récepteur mobile en se référant à une station de base de coordonnées connues, il peut sembler utile de placer sur une carte le fruit de ces traitements. Bien que Google Maps propose un mode de positionnement en temps réel depuis un récepteur GNSS (**Tools -> GPS -> Realtime**), il s'agit d'un jouet propriétaire qui ne possède pas toutes les couches d'analyse que nous aurions été susceptibles d'obtenir lors de traitements antérieurs à la séance d'observation sur le terrain, et nous allons donc nous efforcer d'interfacer Quantum-Gis (QGIS) avec le flux issu du traitement de RTKLIB. En effet, au moins dans sa version 3.28 utilisée pour cette démonstration, un



– ... RTKLIB fait appel à Centipède et l'IGN pour afficher dans QGIS –

flux de données GNSS peut être acquis depuis **gpsd** par QGIS en activant **View -> Panels -> GPS Information**. La nouvelle fenêtre qui apparaît sous la liste des couches propose de se connecter de diverses façons à une source de données de localisation et en particulier **gpsd**. Si le démon de gestion des données de positionnement est actif et communique par défaut sur son port 2947, alors QGIS indique une cible au niveau de la position du récepteur. Aucune configuration n'est modifiée dans QGIS puisque **gpsd** communique par défaut au travers de ce port de *localhost*.

Cependant, nous ne sommes pas intéressés par la connexion d'un simple GPS à QGIS, mais par l'exploitation du résultat du traitement en temps réel par RTKLIB. Ainsi, il faudra remplacer l'interface physique (Bluetooth, USB ou RS232) par un flux de données, et naturellement un *pipe* nommé ou FIFO vient à l'esprit. Cependant, avant de se lancer dans le flux continu de données, commençons par essayer de communiquer un fichier au format NMEA issu du traitement par **rnx2rtkp** à **gpsd** en vue de le transmettre à QGIS.

Les auteurs de **gpsd** ont prévu la capacité à tester le démon avec un jeu de données fictive et proposent pour cela le script Python **gpsfake**. Ce programme accepte en argument un fichier contenant des données de positionnement au format NMEA et injecte ces informations à une instance spécifique de **gpsd**. Afin d'exécuter **gpsfake**, il faut s'assurer que le port 2947 de *localhost* est libre et qu'aucune instance de **gpsd** n'y est associée (**sudo lsof -i:2947 -n -P** ne doit rien répondre, sinon **systemctl stop gpsd** et **systemctl stop gpsd.socket**) avant de lancer **gpsfake -S fichier.nmea**. Nous avons pris un temps significatif à découvrir que les pseudoterminaux créés par **gpsfake** sont protégés par un superviseur de sécurité AppArmor dont seule la politesse nous interdit de s'en débarrasser tant ses restrictions sont handicapantes. En effet, les messages de **dmesg** du type :

```
[V.W] audit: type=1400 audit(X:Y): apparmor="DENIED" operation="open"
profile="/usr/sbin/gpsd"
name="/dev/pts/10" pid=Z comm="gpsd" requested_mask="r" denied_mask="r"
fsuid=1000 ouid=1000
```

indiquent que AppArmor a non seulement interdit l'accès au pseudoterminal **/dev/pts/10**, mais en plus il interdit à l'administrateur d'en changer les permissions. Pour une fois, nous allons suivre l'approche rationnelle consistant à modifier correctement les fichiers de configuration de AppArmor au lieu de simplement contourner le problème ; ces fichiers dans **/etc/apparmor.d/usr.sbin.gpsd** contiennent des informations de la forme :

```
# common serial paths to GPS devices
/dev/tty{S,USB,AMA,ACM}[0-9]*    rw,
/sys/dev/char                    r,
/sys/dev/char/**                 r,
```

qui autorisent à **gpsd** d'accéder aux périphériques de communication de récepteurs matériels, que nous complétons de :

```
/dev/pts/*    rw,
```

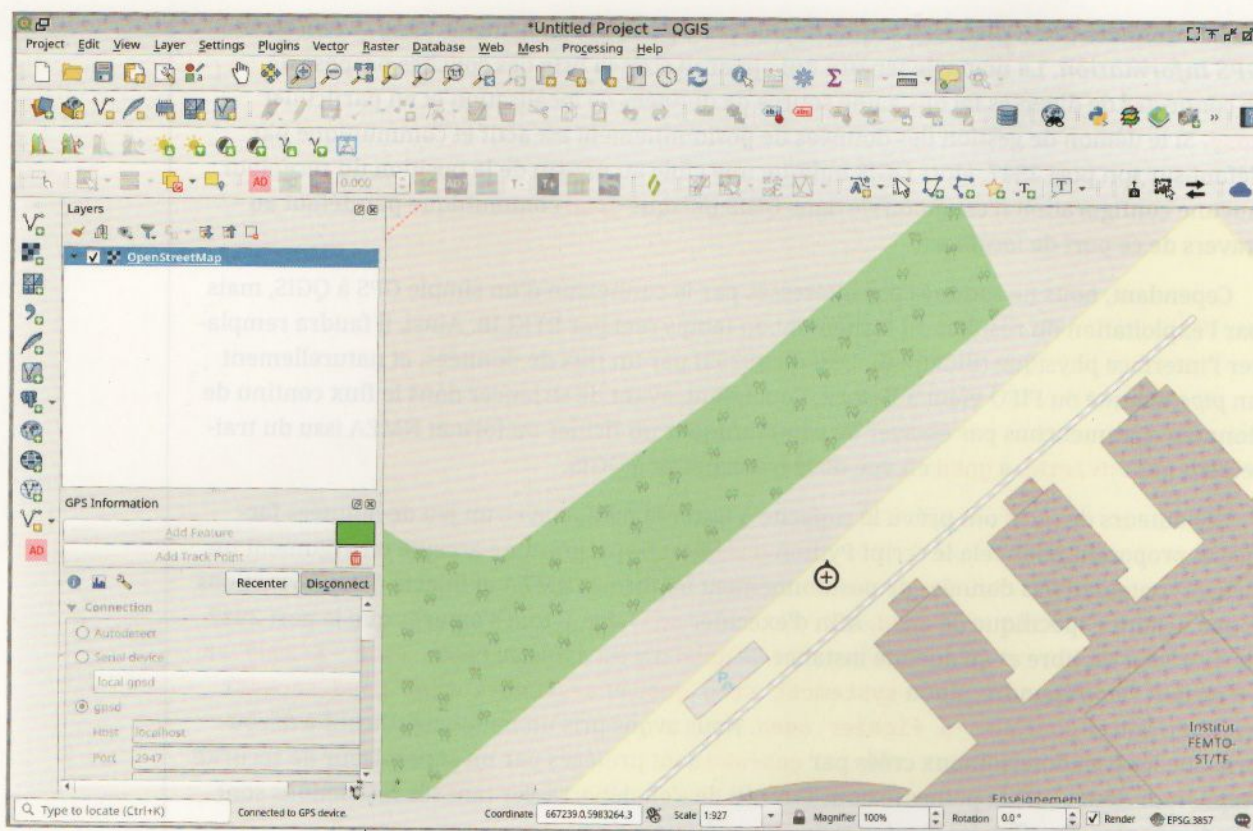



Figure 6 : Transfert des mesures traitées par RTKLIB à QGIS pour affichage sur un fond de carte – ici OpenStreetMaps – et pouvant contenir les diverses couches issues des traitements antérieurs sur le même site. La cible est placée avec une résolution et une exactitude subcentimétrique à une position transmise par `gpsd`, ici alimenté par un fichier produit par RTKLIB.

pour autoriser l'accès aux pseudoterminaux. À l'issue de cette correction prise en compte par `apparmor_parser -r /etc/apparmor.d/usr.sbin.gpsd`, l'erreur disparaît au lancement de `gpsfake` avec l'argument `-S` pour ralentir le flux de données suivi du nom de l'archive au format NMEA, et en connectant QGIS, nous avons la satisfaction de voir le voyant vert s'allumer et une cible se placer sur la position issue du traitement de RTKLIB (Fig. 6).

Nous pourrions vérifier que `gpsd` comprend bien les informations fournies au moyen des outils en ligne de commande que sont `cgps` ou `gpsmon` pour valider la position proposée dans le fichier NMEA.

Finalement, nous voudrions dynamiquement mettre à jour QGIS avec les mesures issues du flux continu de traitement de RTKLIB et non d'un fichier issu d'un traitement statique. La FIFO vient évidemment à l'esprit pour effectuer ce traitement : un *pipe* nommé est créé au moyen de `mkfifo /tmp/mafifo` mais une fois de plus, AppArmor va nous empêcher d'en modifier les permissions et nous prenons soin d'ajouter `/tmp/mafifo rw`, après les autorisations déjà discutées auparavant de `/etc/apparmor.d/usr.sbin.gpsd`. Si nous avons pris soin de produire un fichier au format NMEA comme attendu par `gpsd`, par exemple au moyen de :

- ... RTKLIB fait appel à Centipède et l'IGN pour afficher dans QGIS -

```
rnx2rtkp -p 5 -m 15 -n rover.obs telechargement_RGP_198748/recherche_1/
bscn042z.23o \
telechargement_RGP_198748/recherche_1/bscn042z.23n > fichier.nmea
```

(notez que le **-e** de cette fonction fournie auparavant a été remplacée par **-n** pour remplacer le format de sortie de XYZ à NMEA), alors lancer **gpsd** par **/usr/sbin/gpsd -N -D 1 /tmp/mafifo** suivi de **while true; do cat fichier.nmea > /tmp/mafifo ;sleep 1;done** va continuellement communiquer par la FIFO le contenu du fichier NMEA. En connectant QGIS à **gpsd**, nous verrons le curseur se positionner au bon endroit. Nous étant convaincu de la validité de la procédure, nous appliquons le concept à la communication en temps réel en ajoutant dans le fichier de configuration de **rtkrcv** une nouvelle sortie au format NMEA pointant vers **/tmp/mafifo** :

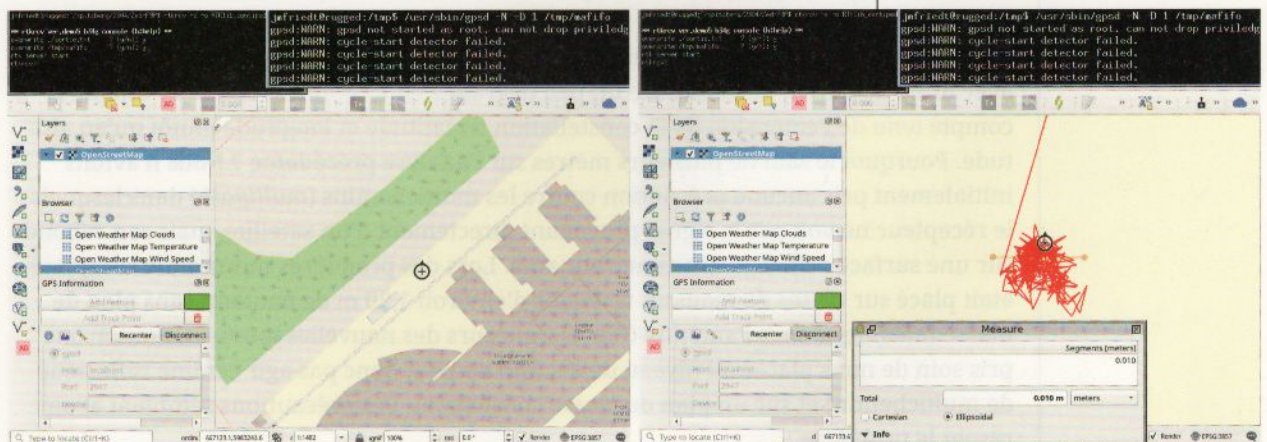
```
outstr2-type      =file
outstr2-path      =/tmp/mafifo
outstr2-format    =nmea
```

puis :

1. Lancer **gpsd** avec **/usr/sbin/gpsd -N -D 1 /tmp/mafifo**.
2. Lancer **rtkrcv -s -o RTKlib_centipede.conf**.
3. Connecter QGIS.

Le résultat est présenté en Fig. 7 avec un référencement sur la station Centipède ENSMM, avec une mesure de la largeur du nuage de points acquis en cinq minutes du centimètre, acquis en temps réel. On notera que comme tout bon *pipe* nommé, les données ne commencent à circuler que lorsque les deux extrémités du tuyau sont connectées, donc **rtkrcv** demandera à écraser **/tmp/mafifo** lors de la première connexion de QGIS qui échouera, et ce n'est que lors de la deuxième connexion

Figure 7 : Gauche, vue d'ensemble de QGIS recevant en temps réel de nouvelles mesures produites par RTKLIB (terminal en haut à gauche) et communiquées au travers de gpsd (terminal en haut à droite) au travers d'un pipe nommé tel qu'en atteste l'icône verte du menu GPS Information. Droite, zoom sur la position du curseur, indiquant que la position a varié de l'ordre du centimètre au cours des cinq minutes d'acquisition.



que QGIS recevra avec succès les données de position au format NMEA. Parfois, les trames ne commencent pas sur l'information attendue et il faut relancer la tentative de connexion jusqu'à ce que QGIS finisse par comprendre les messages qui lui sont transmis. Les positions sont simultanément sauveées dans un fichier indiquant la position dans un référentiel d'origine le centre de la Terre et indiquant un écart type de la position centimétrique de la forme :

% GPST	x-ecef(m)	y-ecef(m)	z-ecef(m)	Q ns	sdx(m)	sdz(m)	sdz(m)
2023/02/16 07:36:08.000	4313719.3079	452854.5832	4661050.3111	1 13	0.0077	0.0037	0.0077

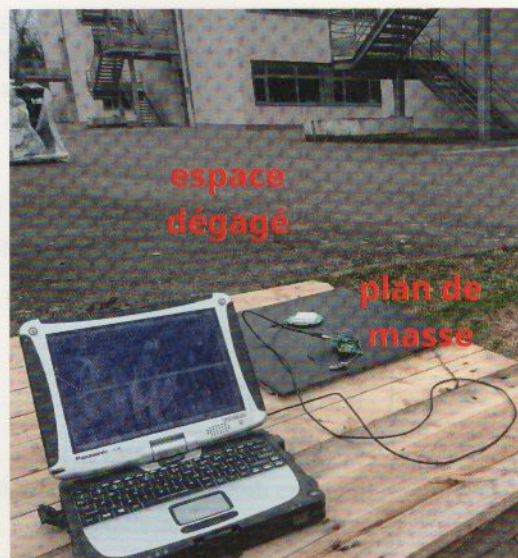
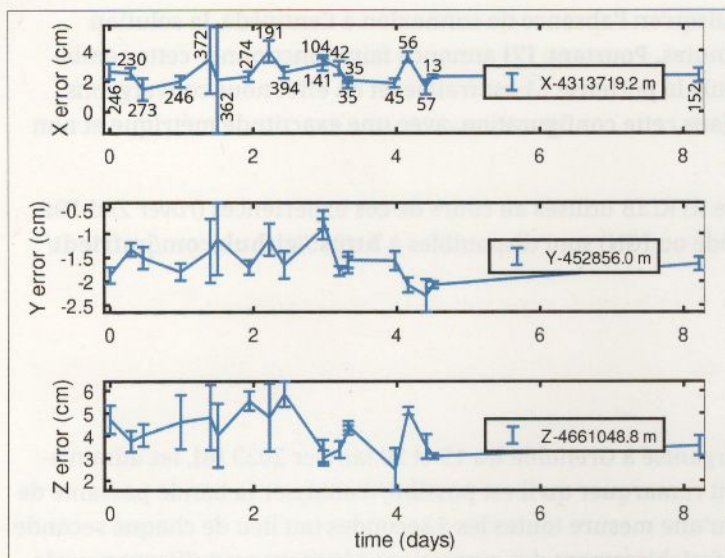
On pourrait s'inquiéter de voir des coordonnées différer de 5 m sur X et d'un mètre sur Y et Z, mais ce système de coordonnées est peu intuitif et l'utilisation du convertisseur de référentiels <https://proj.org/> mis en paquet par Debian GNU/Linux sous le nom **proj-bin** indique que les positions obtenues à partir de Centipède ou IGN ne diffèrent de cette nouvelle position séparée de plusieurs jours de 1,5 m, inacceptable si nous visons un positionnement reproductible centimétrique, avec une altitude qui a même fait un saut de 4 m. En effet, la conversion depuis ECEF vers WGS-UTM31N indique que :

```
$ cs2cs +proj=geocent +datum=WGS84 +units=m +no_defs +to
+init=epsg:32631 -f "%.8f" fichier
X Y Z
726464.90031035 5237471.83799679 342.33079235 # Centipede 11/02/23
726464.68704738 5237471.89734084 342.05299559 # IGN 11/02/23
726463.47093372 5237469.19760583 346.49832384 # Centipede 16/02/23
```

il faut donc prendre soin de vérifier la reproductibilité des mesures à long terme.

Nous répétons cette mesure régulièrement sur plusieurs jours afin d'analyser l'impact de la géométrie de la constellation des satellites de navigation, en nous rappelant qu'un satellite en orbite moyenne met 12 h pour parcourir sa circonférence et que pendant ce temps, la Terre a fait un demi-tour. Ainsi, la constellation aura considérablement changé en quelques heures, permettant de valider l'insensibilité de la mesure à la position des satellites dans le ciel. En effet, nous constatons sur plusieurs jours que l'observation ne varie que de quelques centimètres, en accord avec nos attentes (Fig. 8), avec une fluctuation de la mesure un peu dégradée en Z comme on peut s'y attendre, compte tenu de l'extension de la constellation en latitude et longitude plutôt qu'en altitude. Pourquoi le saut de plusieurs mètres sur l'analyse précédente ? Nous n'avions initialement pris aucune précaution contre les multichemins (*multipath*) dans lesquels le récepteur ne voit pas le signal provenant directement d'un satellite, mais son rebond sur une surface réfléchissante environnante. Lors des premières mesures, le récepteur était placé sur un tas de mousses isolantes d'environ 1,20 m de hauteur, sans plan de masse métallique sous l'antenne (Fig. 1). Au cours des nouvelles mesures, nous avons pris soin de nous placer au niveau du sol (qui ne peut donc pas agir comme réflecteur de multichemin) et sur un plan de masse conséquent. Ces précautions semblent avoir résolu le problème.

– ... RTKLIB fait appel à Centipède et l'IGN pour afficher dans QGIS –



CONCLUSION

Alors que les vendeurs de babioles ne rêvent que de localiser les clients avec une résolution centimétrique pour leur transmettre la publicité de produits inutiles au bon endroit, l'avènement de récepteurs de localisation par constellations de navigation par satellites propose des perspectives d'applications fascinantes, pour qui prend le temps d'apprendre à s'en servir. Nous avons exploré le positionnement avec une résolution subcentimétrique en exploitant une station de référence supposée soumise aux mêmes sources de biais que le dispositif mobile, avec une exactitude limitée par la localisation de la station de base. Avec une station de base librement positionnée, l'exactitude reste métrique mais avec une résolution subcentimétrique, alors que placer la station de base sur un emplacement connu ou exploiter une station de référence de position connue ramène l'exactitude à la résolution du centimètre. Afin de valider sur le terrain la qualité de la mesure et de l'interpréter dans un contexte géographique, nous avons transmis le fruit de ces calculs à QGIS sous GNU/Linux et QField sous Android, qui par ailleurs affichent les fonds de cartes ou résultats des analyses antérieurs sur le site d'étude.

Nous avons mentionné notre capacité à « cross-compiler » `rtkrcv` sur Raspberry Pi 4 : étrangement, nous ne sommes jamais arrivés à obtenir une position centimétrique, l'instruction satellite de `rtkrcv` indiquant tout au plus deux satellites en mode RTK alors que le même fichier de configuration avec l'antenne au même emplacement permet d'obtenir une solution sans problème sur ordinateur portable, et ce même en passant le processeur de la Raspberry Pi 4 en mode performance pour le cadencer à 1500 MHz. Le problème doit être lié à la puissance de calcul

Figure 8 : Mesure sur plus d'une semaine de la position du récepteur placé sur une plaque métallique au niveau du sol (droite) avec une antenne remplacée à chaque séance avec une précision centimétrique sur le support, chaque mesure étant initialisée par un démarrage à froid indépendamment de toutes les précédentes. Nous constatons (gauche) une variabilité de l'ordre du centimètre de la mesure sur plusieurs jours d'observations, en accord avec les objectifs visés. De haut en bas, l'écart de position en X, Y et Z après en avoir soustrait la valeur moyenne (légende). Le nombre indiqué au-dessus de chaque barre d'erreur dans le cadran X du haut indique le nombre de mesures (1 par seconde) exploitées lors du calcul statistique, séquences contiguës de solutions « Fix » (qualité 1).

pour atteindre la solution RTK, puisqu'en l'absence de connexion à Centipède, la solution *Single* est atteinte en quelques minutes. Pourtant, [7] annonce faire fonctionner cette combinaison : nous y constatons que seule la porteuse L1 est traitée, et en effet nous convergeons vers une solution de type *Single* dans cette configuration, avec une exactitude métrique et non centimétrique.

Les fichiers de configuration de RTKLIB utilisés au cours de ces expériences (rover Zed-F9P vs basestation Zed-F9P ou Centipède ou IGN) sont disponibles à https://github.com/jmfriedt/RIOT_NyAlesund.

REMERCIEMENTS

Au cours de l'Eclipse IoT Day organisé à Grenoble les 19 et 20 janvier 2023 [6], les administrateurs de Centipède nous ont fait remarquer qu'il est possible d'abaisser la bande passante de communication en n'acquérant qu'une mesure toutes les 5 secondes (au lieu de chaque seconde actuellement) et que le taux de rafraîchissement des corrections résultant est suffisant pour la correction RTK, un point intéressant lorsque le débit de communication est limité. **JMF**

RÉFÉRENCES

- [1] J.-M Friedt, « Communication LoRa au moyen de RIOT-OS pour la mesure centimétrique par GPS différentiel avec RTKLIB », *Hackable* 45 (nov.-déc. 2022) - <https://connect.ed-diamond.com/hackable/hk-045/communication-lora-au-moyen-de-riot-os-pour-la-mesure-centimetrique-par-gps-differentiel-avec-rtklib>
- [2] G. Goavec-Merou, J.-M Friedt, « On ne compile jamais sur la cible embarquée : Buildroot propose GNU Radio sur Raspberry Pi (et autres) », *Hackable* 37 (2021) - <https://connect.ed-diamond.com/hackable/hk-037/on-ne-compile-jamais-sur-la-cible-embarquee-buildroot-propose-gnu-radio-sur-raspberry-pi-et-autres>
- [3] RTKLIB Manual : Demo5 version (2021) à : https://github.com/rtklibexplorer/RTKLIB/blob/demo5/doc/manual_demo5.pdf
- [4] F. Trystram, *Le procès des étoiles*, Ed. Seghers (1979) ou J. Verne, *Aventures de trois Russes et de trois Anglais dans l'Afrique australe*, Ed. Hetzel et Cie (1872) replacé dans un contexte historique à : https://lesia.obspm.fr/perso/jacques-crovisier/JV/verne_3R3A.html
- [5] S. Barbeau, *Crowdsourcing GNSS features of Android devices* (2021) à : <https://barbeau.medium.com/crowdsourcing-gnss-capabilities-of-android-devices-d4228645cf25>
- [6] Programme, transparents et vidéos des présentations de *Eclipse IoT Day Grenoble 2023* à : https://wiki.eclipse.org/Eclipse_IoT_Day_Grenoble_2023#January_19.2C_2023
- [7] T. Everett, *Raspberry Pi based PPK and RTK solutions with RTKLIB* (11/2022) à : <https://rtklibexplorer.wordpress.com/2022/11/10/raspberry-pi-based-ppk-and-rtk-solutions-with-rtklib/>