ÉLECTRONIQUE | EMBARQUÉ | RADIO | IOT

HACKABLE

L'EMBARQUÉ À SA SOURCE

N° 51
NOV. / DÉC. 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 - 51 - F: 14,90 € - RD

CPPAP : K92470

NFC / BLE / Radio / IR / Hacking

Un outil unique pour explorer NFC, radio, Bluetooth, RFID, USB... ?

FLIPPER ZERO
LE COUTEAU SUISSE DE L'IOT p.24

- Prendre en main le matériel
- Tenir à jour le firmware
- Reconstruire depuis les sources

NFC / RFID

Testez, manipulez et émulez facilement les tags 125 kHz et 13,56 MHz grâce au nouveau Chameleon Ultra p.04

ÉDITO

Et de cinq !

Fin septembre, Eben Upton annonçait l'arrivée de la (ou « du ») Raspberry Pi 5 construite autour du SoC Broadcom BCM2712 à quadruple cœur ARM Cortex-A76 (versus les Cortex-A72 du BCM2711 de la Pi 4). Ceci s'accompagne d'un nouveau GPU VideoCore VII et laisse penser, à première vue, que cette nouvelle génération se résume à un simple (mais notable) gain en performances, plutôt qu'en fonctionnalités, si ce n'est par l'ajout d'un connecteur PCI Express.

Pourtant, cette nouvelle génération de carte est un changement majeur dans l'architecture même du SBC. Nous avons maintenant le RP1 centralisant les fonctionnalités autres que celles qui sont gérées directement par le SoC (HDMI, SD, SDRAM) et communiquant en PCI Express pour fournir l'USB, Ethernet, MIPI, GPIO, UART, SPI, i²c, etc. Cette évolution n'est pas anodine, car en plus de réduire les coûts de fabrication pour la fondation, ceci ouvre toute une galaxie d'évolutions possibles, à commencer par l'exploitation de l'interface PCI Express 2.0 1x permettant d'ores et déjà l'ajout d'un *HAT* disposant d'un connecteur M.2 pour un SSD NVMe.

À l'heure actuelle, une documentation minimale (*initial draft*) existe concernant la puce RP1 **[1]** et sera complétée à l'avenir. Il n'existe à ce stade aucun plan concernant la commercialisation de cette puce, contrairement au RP2040 des Pico, ni d'ouverture complète concernant le design du composant. Le portage d'autres systèmes que GNU/Linux (Raspberry Pi OS) sera donc dépendant des informations issues de la documentation et du code présent dans les sources du noyau. Ceci risque de prendre du temps.

Faut-il craquer de suite pour cette nouvelle carte, encore en précommande alors que je rédige ce texte ? Personnellement, je préfère attendre un peu et éviter l'achat impulsif, juste histoire de passer outre l'effet d'annonce et laisser tout cela mûrir un peu. Diantre ! Deviendrais-je raisonnable ?!

Denis Bodor

[1] https://datasheets.raspberrypi.com/rp1/rp1-peripherals.pdf

Hackable Magazine
est édité par Les Éditions Diamond

LES ÉDITIONS DIAMOND
STIMULANT DE MATIÈRE GRISE

BP 20142 - 67602 SELESTAT CEDEX - France
E-mail : lecteurs@hackable.fr -
Service commercial : cial@ed-diamond.com
Sites : hackable.fr - ed-diamond.com
Directeur de publication : Arnaud Metzler
Rédacteur en chef : Denis Bodor
Réalisation graphique : Kathrin Scali
Régie publicitaire : Tél. : 03 67 10 00 27
Service abonnement : Les Éditions Diamond
BP 20142 - 67602 SELESTAT CEDEX, France, Tél. : 03 67 10 00 20
Impression : Westermann Druck | PVA, Braunschweig, Allemagne
Distribution France :
(uniquement pour les dépositaires de presse)
MLP Réassort : Plate-forme de Saint-Barthélemy-d'Anjou.
Tél. : 02 41 27 53 12
Plate-forme de Saint-Quentin-Fallavier. Tél. : 04 74 82 63 04

Service des ventes : Abomarque - Tél. : 06 15 46 15 88
IMPRIMÉ en Allemagne - PRINTED in Germany
Dépôt légal : À parution
N° ISSN : 2427-4631
CPPAP : K92470
Périodicité : bimestriel - Prix de vente : 14,90 €

Suivez-nous sur Twitter
@hackablemag

Alsace

SOMMAIRE

À PROPOS DE HACKABLE...

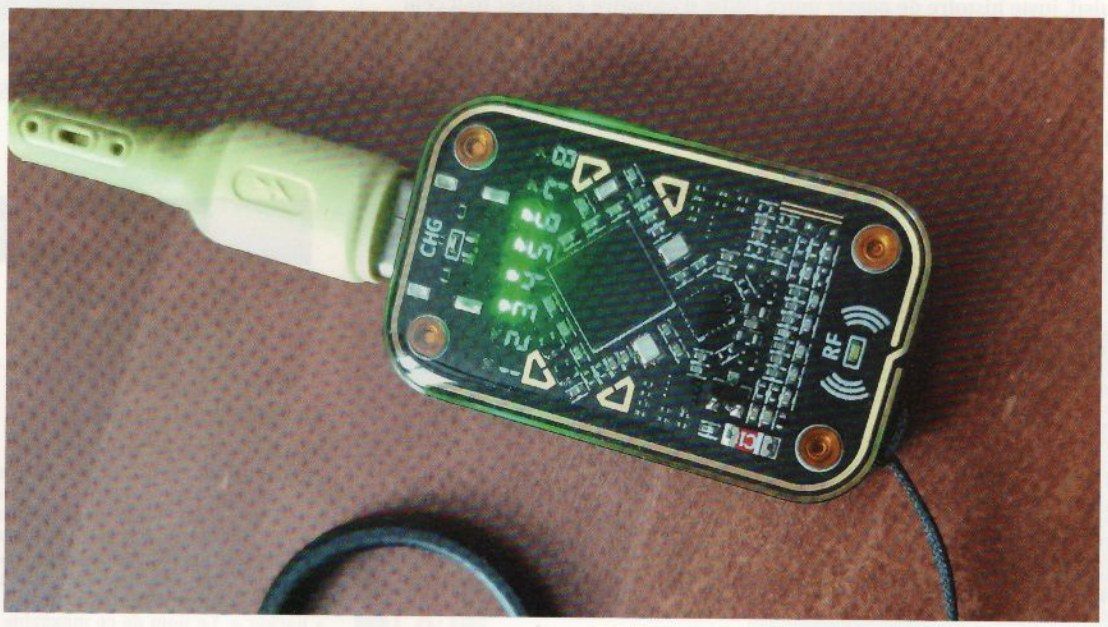
HACKS, HACKERS & HACKABLE

Ce magazine ne traite pas de piratage. Un **hack** est une solution rapide et bricolée pour régler un problème, tantôt élégante, tantôt brouillonne, mais systématiquement créative. Les personnes utilisant ce type de techniques sont appelées **hackers**, quel que soit le domaine technologique. C'est un abus de langage médiatisé que de confondre « pirate informatique » et « hacker ». Le nom de ce magazine a été choisi pour refléter cette notion de **bidouillage créatif** sur la base d'un terme utilisé dans sa définition légitime, véritable et historique.

CHAMALEONULTRA, UN NOUVEL OUTIL POUR VOS EXPÉRIMENTATIONS RFID/NFC

Denis Bodor

Si vous êtes curieux et amateur de technologies RFID/NFC, vous connaissez sans doute le classique lecteur USB ACR122U, le plus compétent/stable SCL3711 et l'incontournable (et coûteux) Proxmark 3 RDV4. Un petit nouveau a fait son apparition dernièrement et le qualificatif de « petit » n'est pas ici utilisé à la légère : le ChameleonUltra. Petite prise en main et essai de ce qui sera, à terme, un véritable bijou indispensable dans votre boîte à outils…



– ChamaleonUltra, un nouvel outil pour vos expérimentations RFID/NFC –

Qualifier le ChameleonUltra n'est pas chose facile, car il reprend certaines fonctionnalités du Proxmark 3, peut fonctionner comme un lecteur RFID/NFC 125 kHz (LF pour *Low Frequency*) et 13,56 MHz (HF pour *High Frequency*), mais aussi, et surtout, dispose de fonctions d'émulation/simulation de *tags* plus performantes que celles proposées par un Proxmark 3 (en termes de vitesse et de réactivité). De plus, le ChameleonUltra est minuscule (2,4 cm par 4 cm et 8 mm d'épaisseur) et peut fonctionner de façon autonome (accu LiPo) ou connecté à un PC/Mac via une liaison USB à connecteur USB-C et interface CDC-ACM (*Communication Device Class - Abstract Control Model*). Certes, un Proxmark 3 RDV4 équipé d'un module *BlueShark Standalone* fera de même, mais on ne parle pas du tout de la même échelle ici, ni en taille (cf. photos) ni en autonomie (~45 min pour le BlueShark contre plus de 6 mois pour le ChameleonUltra).

Ce périphérique, au prix non négligeable de quelque 130 € (chez les Lyonnais de Lab401 par exemple, distributeur officiel pour l'Europe), peut donc être à la fois un lecteur RFID/NFC, un *tag* (8 emplacements d'émulation disponibles, déclinés en HF et LF, donc 16 *tags* en tout) et un outil de *pentest* pour éprouver la sécurité de *tags* comme les traditionnels MIFARE Classic ou des dispositifs de contrôle d'accès utilisant cette même technologie. Je parle ici de MIFARE Classic, car, et c'est très important si vous comptez éviter toute déception ou frustration après la réception de votre ChameleonUltra, c'est le seul type de *tag* HF qui est actuellement supporté par le *firmware* du matériel. Le ChameleonUltra a un énorme potentiel, mais pour l'instant, c'est surtout du potentiel. Il est en effet capable matériellement, par exemple, d'émuler des NTAG21x, des MIFARE Ultralight (+ Ultralight C), des MIFARE DESFire (+ EV1, EV2) et MIFARE Plus, mais l'applicatif en *firmware* n'implémente pour le moment pas ces fonctions. Il en va de même pour la lecture des DESFire et MIFARE Plus, ou encore, côté LF, de la simulation des T5577, HID Prox, ioProx ou encore des EM4305 pour n'en citer que

Lorsque vous commandez un ChameleonUltra (officiel), il n'arrive pas seul. Vous recevez également un câble USB (un peu court), une gaine de protection en silicone, un mini-tournevis Torx pour démonter le produit avec deux vis supplémentaires, et un anneau en céramique pour faire de votre ChameleonUltra un porte-clés. Le tout dans une jolie boîte.

Le ChameleonUltra est véritablement un concentré de technologies RFID/NFC réduit au maximum de compacité. L'objectif est simple : rendre l'outil le plus discret possible, sinon invisible dans le creux d'une main.

quelques-uns. Une page spécifique du dépôt GitHub **[1]**, `docs/technical_whitepaper.md`, liste en détail l'état des fonctionnalités effectivement supportées à la date où sont couchées ces lignes, mais dans l'ensemble, ceci se limite à du MIFARE Classic (HF) et du EM410x (LF). Il s'agit donc d'un potentiel extraordinaire outil en devenir...

Préciser cet état de fait me paraît important, car les caractéristiques listées sur les pages des sites commercialisant le matériel restent relativement vagues sinon trompeuses, en précisant des choses comme « *Cartes supportées [...] HID Prox / Indala / PAC/Stanley* », ce qui est, au mieux, inexact au sens strict du terme. Pour quelque 150 € d'investissement potentiel, ceci mérite pourtant d'être clairement annoncé. Soit dit en passant, la mention des attaques « Emboîté » (*nested*), « Emboîté dur » (*HardNested*) et « Côté obscur » (*DarkSide*) sur MIFARE Classic, sur la même page, est si comique qu'on se demande si ce n'est pas fait exprès.

1. MAIS QU'EST-CE QU'UN CHAMELEONULTRA ?

Le ChameleonUltra est construit autour d'un microcontrôleur nRF52840 de chez Nordic Semiconductor, intégrant un cœur ARM Cortex-M4, 1 Mio de flash, 256 Kio de RAM et différents périphériques, dont un contrôleur *Bluetooth Low Energy* (BLE), une interface USB 2.0 et un support NFC-A initialement présent pour offrir une capacité d'émulation afin de simplifier l'appairage Bluetooth. De base, le nRF52840 est donc limité côté NFC puisqu'incapable de lire ou d'écrire des *tags* ISO/IEC 14443 A (généralement raccourcis en « HF 14a » ou « 14a »). De ce fait, les développeurs, constatant que les capacités d'émulation NFC du microcontrôleur Nordic affichaient des performances très intéressantes et qu'il était même possible d'émuler du MIFARE Classic (qui n'est pas à strictement parler du NFC), décidèrent de compléter le circuit avec une puce MFRC522 de NXP. Celle-ci offrant un *frontend* compatible MIFARE, NTAG et 14a en général, le duo nRF52840+MFRC522 fait du ChameleonUltra un simulateur, un lecteur et un outil de test générique RFID/NFC. La modulation basse fréquence (LF) est prise en charge directement par la partie analogique du nRF52840 avec l'assistance de quelques circuits analogiques (opamp, filtres, détection d'enveloppe, pilotage de l'antenne, etc.). À cela s'ajoutent enfin huit LED RGB et deux boutons formant l'interface utilisateur en mode autonome afin de pouvoir sélectionner le *tag* à émuler et/ou copier l'UID d'un *tag* physique pour l'émuler ensuite (les emplacements « peuplés » LF sont bleus, HF en vert et HF+LF en

rouge). Le choix de l'action liée à l'utilisation des boutons, que ce soit par appui bref ou long, est configurable via l'un des outils supportant le matériel (ligne de commande, application GUI en Flutter ou application module iOS/Android). Une LED blanche est également présente et sert à signaler la présence d'un champ électromagnétique (non configurable, apparemment).

Vous l'avez compris, le ChameleonUltra est excessivement compact. Il se compose de deux PCB prenant en sandwich l'accu LiPo avec l'une des deux faces donnant accès aux boutons et l'autre, noyée dans la résine, montrant les puces et les LED. Sur la tranche se trouve un connecteur USB-C et de l'autre côté, une encoche permettant d'attacher un anneau pour en faire un porte-clés, un *lanyard*, etc. En commandant la version officielle, vous recevrez également un câble USB-C/USB-A (transformable en USB-C/USB-C), une attache porte-clés, une gaine de protection en silicone et un mini-tournevis pour éventuellement démonter le produit (et remplacer l'accu).

Notez que les distributeurs officiels (Lab401, Hackerwarehouse, RRG sur AliExpress, Sneaktechnology) sont naturellement l'option à préférer pour un achat, mais le projet étant *open source*, des ChameleonUltra sont également apparus rapidement chez des vendeurs (autre que RRG) sur AliExpress, avec des niveaux de qualité très variables et des prix sensiblement plus bas (~120 €). Personnellement, je n'opterais pas pour une telle alternative, non seulement parce qu'il est préférable de supporter le projet, mais également en raison de l'incertitude concernant ce qui sera effectivement réceptionné. Certaines photos d'annonces montrent clairement un matériel différent, plus épais et donc des économies évidentes faites sur les composants, comme c'était déjà le cas pour les Proxmark 3. D'autres vous préviennent que des clones existent, mais sont facilement identifiables en raison de leur libellé « Chamele0nUltra », alors que le leur est, bien entendu, le « vrai » ChameleonUltra (sic). Enfin, certaines annonces proposent des ChameleonUltra à un prix défiant toute concurrence (~65 €) alors qu'il s'agit en réalité de *devkit*, une déclinaison du matériel en un seul PCB, beaucoup moins compact (5,3 cm par 8,5 cm), qui offre les mêmes fonctionnalités, mais est destiné au développement et à la mise au point (connecteur SWD, *Serial Wire Debug*) du *firmware*.

2. INSTALLATION, COMPILATION ET PRISE EN MAIN

Le ChameleonUltra, en émulateur de *tag*, est autonome, mais sa configuration nécessite, bien entendu, l'utilisation d'un outil spécifique. Vous pouvez parfaitement vous en sortir avec Chameleon Ultra GUI **[2] [3] [4]** ou encore MTools Lite **[5]** pour le fonctionnement en tant que lecteur RFID/NFC, mais l'approche du magazine est généralement plus « spartiate », avec une préférence vers la ligne de commande. Plutôt que de reposer sur un binaire ou une application clé en main, nous allons donc nous baser sur les sources officielles pour tout reconstruire. Ceci vous permettra également d'être systématiquement à jour tout en vous permettant, par la suite, de contribuer au projet plus rapidement.

Commençons donc par récupérer les sources via GitHub et lançons immédiatement la construction des quelques binaires nécessaires pour les attaques sur les MIFARE Classic :

```
$ git clone https://github.com/RfidResearchGroup/ChameleonUltra.git
$ cd ChameleonUltra
$ cd software/src
$ mkdir build
$ cd build

$ cmake ../
-- The C compiler identification is GNU 12.2.0
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: /usr/bin/cc - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Configuring done
-- Generating done
-- Build files have been written to: /home/denis/ChameleonUltra/software/src/build

$ make
[  2%] Building C object CMakeFiles/nested.dir/crapto1.c.o
[  5%] Building C object CMakeFiles/nested.dir/crypto1.c.o
[  8%] Building C object CMakeFiles/nested.dir/bucketsort.c.o
[...]
[ 94%] Building C object CMakeFiles/mfkey64.dir/parity.c.o
[ 97%] Building C object CMakeFiles/mfkey64.dir/mfkey64.c.o
[100%] Linking C executable /home/denis/ChameleonUltra/software/bin/mfkey64
[100%] Built target mfkey64

$ cd ../..
$ ls bin/
darkside  mfkey32  mfkey32v2  mfkey64  nested
```

Ces cinq binaires que nous obtenons dans `software/bin` sont destinés à être exécuté localement, ce sont des programmes pour l'hôte et non pour l'ARM du ChameleonUltra. En effet, le matériel n'est pas capable de « craquer » lui-même les clés de certains *tags* MIFARE, mais repose sur la puissance de calcul de la machine hôte.

Pour piloter notre ChameleonUltra, nous devons utiliser le client écrit en Python se trouvant dans `software/script` sous la forme du fichier `chameleon_cli_main.py`. Bien entendu, Debian (et ses dérivés) étant Debian, ce script nécessite un module dans une version qui n'est pas disponible sous la forme de paquet (voir `requirements.txt` dans le même répertoire). Pour ne pas saccager notre environnement de travail, nous utilisons le mécanisme d'environnement virtuel de Python, via le module `venv` :

```
$ python3 -m venv /kkpart/ChameleonUltra
```

Ceci aura pour effet d'installer un interpréteur Python spécifique, local, et non géré par le système de gestion de paquets, dans `/kkpart`, qui est un répertoire que vous aurez créé

arbitrairement pour l'occasion à l'emplacement qui vous chante. Ceci fait, nous pouvons utiliser pip3, se trouvant là en compagnie d'un lien symbolique distinct vers l'interpréteur Python, pour installer les modules nécessaires :

```
$ /kkpart/ChameleonUltra/bin/pip3 install -r requirements.txt
Collecting pyserial==3.5
  Using cached pyserial-3.5-py2.py3-none-any.whl (90 kB)
Collecting colorama==0.4.6
  Using cached colorama-0.4.6-py2.py3-none-any.whl (25 kB)
Collecting prompt-toolkit==3.0.39
  Using cached prompt_toolkit-3.0.39-py3-none-any.whl (385 kB)
Collecting wcwidth
  Using cached wcwidth-0.2.6-py2.py3-none-any.whl (29 kB)
Installing collected packages: wcwidth, pyserial, prompt-toolkit, colorama
Successfully installed colorama-0.4.6
prompt-toolkit-3.0.39 pyserial-3.5 wcwidth-0.2.6
```

On me souffle dans l'oreillette que ma façon d'utiliser venv est « bizarre ». Étant ce qu'on peut appeler un hérétique aux yeux des dévots de la secte des adorateurs du grand serpent, je prends ça comme un compliment. Ce que vous êtes censé faire, proprement, est en fait :

```
$ source /kkpart/bin/activate
<Vous êtes dans l'environnement comme
l'indique le prompt modifié>

$ pip3 install -r requirements.txt

$ python chameleon_cli_main.py
[...]

$ deactivate
```

Nous pouvons enfin exécuter le script, via l'interpréteur de l'environnement fraîchement créé :

```
$ /kkpart/ChameleonUltra/bin/python chameleon_cli_main.py
 ____ _   _    _    __  __ _____ _     _____ ___  _   _
/ ___| | | |  / \  |  \/  | ____| |   | ____/ _ \| \ | |
| |   | |_| | / _ \ | |\/| |  _| | |   |  _|| | | |  \| |
| |___|  _  |/ ___ \| |  | | |___| |___| |__| |_| | |\  |
\____|_| |_/_/   \_\_|  |_|_____|_____|_____\___/|_| \_|

[Offline] chameleon -->
```

Au lancement, le client n'est pas connecté au ChameleonUltra que vous aurez préalablement branché en USB. Il faut pour cela utiliser la commande `hw` (comme *hardware*) et `connect` :

```
[Offline] chameleon --> hw connect
{ Chameleon connected }
[USB] chameleon -->
```

Si les permissions sont correctement configurées (c'est un port série CDC ACM qui apparaît à la connexion), le client détecte automatiquement le périphérique et vous pouvez alors interagir avec ce dernier, pour demander son identifiant, par exemple :

```
[USB] chameleon --> hw chipid get
- Device chip ID: 09312bf290b41fa0
```

Mais si nous demandons d'autres informations, comme la version du *firmware* (`version`) ou les informations sur les emplacements, ou *slots*, d'émulation (`hw slot list`), nous rencontrons un problème :

```
[USB] chameleon --> hw version
CLI exception: Traceback (most recent call last):
  File "/home/denis/ChameleonUltra/software/script/chameleon_cli_main.py",
line 175, in startCLI
    unit.on_exec(args_parse_result)
  File "/home/denis/ChameleonUltra/software/script/chameleon_cli_unit.py",
line 295, in on_exec
    git_version = self.cmd.get_git_version()
                  ^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/home/denis/ChameleonUltra/software/script/chameleon_cmd.py",
line 309, in get_git_version
    resp = self.device.send_cmd_sync(DATA_CMD_GET_GIT_VERSION, 0x00)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/home/denis/ChameleonUltra/software/script/chameleon_com.py",
line 344, in send_cmd_sync
    raise CMDInvalidException(f"Device unsupported cmd: {cmd}")
chameleon_com.CMDInvalidException: Device unsupported cmd: 1017

[USB] chameleon --> hw slot list
```

Le code du projet évolue rapidement et notre matériel à peine réceptionné dispose d'un *firmware* trop ancien pour le client obtenu via GitHub. Au moment des tests, aucun *firmware* à jour facilement installable n'était disponible (là encore, tout a évolué depuis) et j'ai été agréablement obligé de tout simplement construire un nouveau *firmware* et de le flasher dans le périphérique. Vous pouvez vous passer de ce genre de tracasserie à présent, mais être en mesure de le faire, rapidement, vous permettra de rester dans la course et de toujours utiliser la dernière version après un simple `git pull` dans le répertoire constituant votre *repo* local. C'est la même logique qui s'applique avec un Proxmark 3 et c'est très bien ainsi.

Pour flasher le *firmware*, rien de plus simple, il suffit de passer le ChameleonUltra en mode DFU (*Device Firmware Upgrade*) pour que le nRF52840 démarre sur son *bootloader* et nous permette d'utiliser un outil spécifique Nordic (et à priori propriétaire) pour mettre à jour l'application en mémoire flash. Cet outil, appelé *nRF Util* est disponible sur le site du constructeur **[6]** en version Windows, macOS et GNU/Linux. Pour ce système, c'est directement un binaire ELF x86-64 qui est téléchargé et devra donc être rendu exécutable (`chmod +x`) et placé dans le `$PATH` (`~/bin/` chez moi). Vous devrez également ajuster les règles *udev* pour permettre un accès au périphérique USB redémarré en mode DFU. Un fichier `/etc/udev/rules.d/nrf.rules` contenant ce qui suit fera très bien l'affaire :

```
SUBSYSTEMS=="usb", ATTRS{idVendor}=="1915", MODE="0666", GROUP="plugdev"
```

Le reste sera pris en charge par un script `flash-dfu-app.sh` présent dans le répertoire `firmware/`, qui s'occupera du passage en mode DFU et du flashage. Mais :

```
$ firmware/flash-dfu-app.sh
Flashing ultra
DFU package for ultra not found, aborting.
Build firmware using CURRENT_DEVICE_TYPE=ultra firmware/build.sh
```

Bien sûr, encore faut-il avoir quelque chose à flasher, ce qui n'est pas le cas par défaut. Nous devons donc créer l'archive ZIP normalement présente dans le sous-répertoire `firmware/objects/` et, implicitement donc, compiler le *firmware*.

Là, les choses se compliquent (et deviennent un peu étranges aussi), car nous avons besoin d'outils supplémentaires, encore une fois disponibles sur le site de Nordic **[7]**. Ces *nRF Command Line Tools* pourront être installés sous la forme d'un paquet Debian décliné en version X86-64, i386, ARM 32 bits et ARM 64 bits, directement téléchargeable une fois la bonne plateforme sélectionnée (des RPM sont également présents). Le paquet pourra ensuite être installé via la commande :

Le fonctionnement autonome est grandement facilité par l'interface utilisateur consistant en deux boutons (à l'arrière) et une tripotée de LED RGB en façade. 8 pour la sélection de l'emplacement mémoire du tag à émuler/simuler, un témoin de charge (« CHG ») et un notificateur de présence de champ électromagnétique (en bas).

```
$ sudo dpkg -i nrf-command-line-tools_10.23.0_amd64.deb
```

Ce n'est pas tout, pour développer (et donc compiler) pour le microcontrôleur nRF, nous avons besoin d'un SDK et c'est là que tout ceci prend une tournure vraiment peu conventionnelle. Installer le SDK se fera via la commande `nrfutil`, celle-là même que vous avez téléchargée sous forme de binaire précédemment, avec :

```
$ nrfutil install device nrf5sdk-tools
```

Il n'est pas nécessaire d'être `root` pour cette opération. Fort heureusement d'ailleurs, car sinon il n'y aurait tout simplement pas d'article. Il est parfaitement hors de question que j'exécute un binaire inconnu avec cette identité, même dans un environnement isolé. On pourra ensuite vérifier la présence du SDK avec :

```
$ nrfutil list
Command        Version  Description
device         1.2.2    Work with Nordic Semiconductor devices
nrf5sdk-tools  1.0.1    nRF5 SDK tools that were available in nRF Util 6
   dfu
   keys
   pkg
   settings
   zigbee
Found 2 installed command(s)
```

Je ne vous cache pas que tout ceci ne me plaît aucunement et que la faute n'en incombe ni aux créateurs du ChameleonUltra, ni aux développeurs contribuant au projet. Nordic, si vous lisez ceci, faites les choses proprement et de façon cohérente, s'il vous plaît : *packagez* tous vos outils, pas la moitié d'entre eux, et pour l'amour du ciel, arrêtez de vous éloigner de l'écosystème *open source* (cf. **[8]** et le fichier `LICENSE` présent dans ce dépôt archivé).

Nous y sommes presque, mais nous devons surmonter un nouveau problème. Le processeur au cœur du nRF52840 est un ARM Cortex-M4, nous avons donc besoin d'une chaîne de compilation pour cette cible. Si vous développez pour Raspberry Pico, par exemple, vous avez très certainement déjà ce genre de chose à portée de main (paquet `gcc-arm-none-eabi` et ses dépendances) sous la forme d'un GCC 12.2.rel1-1 (avec Debian 12). Malheureusement, voici ce que vous obtiendrez en tenant de l'utiliser :

```
Compiling file: oberon_backend_eddsa.c
Compiling file: oberon_backend_hash.c
Compiling file: oberon_backend_hmac.c
Linking target: ../objects/bootloader.out
/usr/lib/gcc/arm-none-eabi/12.2.1/../../../arm-none-eabi/bin/ld:
../objects/bootloader.out section `.svc_data' will not fit in region `FLASH'
/usr/lib/gcc/arm-none-eabi/12.2.1/../../../arm-none-eabi/bin/ld:
```

```
region FLASH overflowed with .data and user data
/usr/lib/gcc/arm-none-eabi/12.2.1/../../../arm-none-eabi/bin/ld:
section .mbr_params_page VMA [000fe000,000fefff]
overlaps section .svc_data VMA [000fdff8,000fe003]
/usr/lib/gcc/arm-none-eabi/12.2.1/../../../arm-none-eabi/bin/ld:
section .nrf_balloc VMA [000fe004,000fe02b]
overlaps section .mbr_params_page VMA [000fe000,000fefff]
/usr/lib/gcc/arm-none-eabi/12.2.1/../../../arm-none-eabi/bin/ld:
region `FLASH' overflowed by 132 bytes
collect2: error: ld returned 1 exit status
```

L'éditeur de liens nous explique gentiment (ou pas) que le binaire est trop gros pour la flash et que certaines sections débordent sur d'autres. Mais, d'autre part, la documentation du ChameleonUltra **[9]** précise que les chaînes de compilation testées et validées par le projet sont *gcc-arm-none-eabi-10.3-2021.10* et */arm-gnu-toolchain-12.2.rel1-XXX-arm-none-eabi*. La version « 12.2.rel1 » de Debian devrait, en principe, fonctionner, mais le *Dockerfile* inclus avec les sources (`firmware/Dockerfile`) télécharge en réalité une chaîne de compilation mise à disposition par Keil et non un GCC ARM « générique » (NDLR : la chaîne *AArch32 bare-metal target* en version 12.3rel1 disponible via **[10]** fonctionne également).

Si nous téléchargeons cette chaîne de compilation depuis **https://armkeil.blob.core.windows.net**, nous obtenons une archive `arm-gnu-toolchain-12.2.rel1-x86_64-arm-none-eabi.tar.xz` qu'il nous suffit de décompresser quelque part dans notre système. Il n'est pas même nécessaire de modifier notre `PATH` puisque le `Makefile` du projet référence directement l'emplacement utilisé (dans `firmware/nrf52_sdk/components/toolchain/gcc/Makefile.posix`) :

```
GNU_INSTALL_ROOT ?= /usr/bin/
GNU_VERSION ?= 9.3.1
GNU_PREFIX ?= arm-none-eabi
```

Il nous suffit donc d'exporter deux variables d'environnement et de lancer la construction avec :

```
export GNU_INSTALL_ROOT=/chemin/arm-gnu-toolchain-12.2.rel1-x86_64-arm-none-eabi/bin/

$ CURRENT_DEVICE_TYPE=ultra firmware/build.sh
Building firmware for ultra (hw_version=0)
+ cd bootloader
+ make -j
Chameleon Ultra <Bootloader>: zZZZZZ.
cd ../objects && mkdir bootloader
Compiling file: main.c
Compiling file: dfu_public_key.c
Compiling file: hw_connect.c
Compiling file: libc_nano_stubs.c
Compiling file: nrfx_clock.c
```

```
[...]
Bootloader DFU Settings:
* File:                   settings.hex
* Family:                 NRF52840
* Start Address:          0x000FF000
* CRC:                    0x66C6F3B3
* Settings Version:       0x00000002 (2)
* App Version:            0x00000001 (1)
* Bootloader Version:     0x00000001 (1)
* Bank Layout:            0x00000000
* Current Bank:           0x00000000
* Application Size:       0x0003370C (210700 bytes)
* Application CRC:        0x4D41F056
* Bank0 Bank Code:        0x00000001
* Softdevice Size:        0x00025634 (153140 bytes)
* Boot Validation CRC:    0x0E8DFCD9
* SD Boot Validation Type:  0x00000001 (1)
* App Boot Validation Type: 0x00000001 (1)
[...]
+ zip -j ultra-binaries.zip
    /tmp/cu_binaries_QB5YhzaUq4/application.hex
    /tmp/cu_binaries_QB5YhzaUq4/bootloader.hex
    /tmp/cu_binaries_QB5YhzaUq4/fullimage.hex
    /tmp/cu_binaries_QB5YhzaUq4/softdevice.hex
  adding: application.hex (deflated 62%)
  adding: bootloader.hex (deflated 59%)
  adding: fullimage.hex (deflated 61%)
  adding: softdevice.hex (deflated 59%)
+ rm -rf /tmp/cu_binaries_QB5YhzaUq4
```

Le script `firmware/build.sh` fait tout le travail à notre place et nous obtenons effectivement l'archive tant désirée dans `./firmware/objects/ultra-binaries.zip`. Nous pouvons alors retenter la mise à jour du *firmware* avec le script shell :

```
$ firmware/flash-dfu-app.sh
Flashing ultra
[00:00:09] ###### 100% [2/2 C2737630B3E6] Programmed
```

Victoire ! De retour dans le client `chameleon_cli_main.py`, tout est maintenant fonctionnel :

```
[Offline] chameleon --> hw connect
{ Chameleon connected }

[USB] chameleon --> hw version
- Version: v1.0 (dev-197-gd928ab2)
```

```
[USB] chameleon --> hw slot list
- Slot 1 data (active):
   HF: Empty - Mifare Classic 1k
   LF: Empty - EM410X
- Slot 2 data:
   HF: Empty - Mifare Classic 1k
   LF: Empty - Unknown
- Slot 3 data:
   HF: Empty - Unknown
   LF: Empty - EM410X
- Slot 4 data (disabled):
   HF: Empty - Unknown
   LF: Empty - Unknown
- Slot 5 data (disabled):
   HF: Empty - Unknown
   LF: Empty - Unknown
- Slot 6 data (disabled):
   HF: Empty - Unknown
   LF: Empty - Unknown
- Slot 7 data (disabled):
   HF: Empty - Unknown
   LF: Empty - Unknown
- Slot 8 data (disabled):
   HF: Empty - Unknown
   LF: Empty - Unknown
- Mifare Classic emulator settings:
   Detection (mfkey32) mode: disabled
   Gen1A magic mode: disabled
   Gen2 magic mode: disabled
   Use anti-collision data from block 0: disabled
   Write mode: Normal
```

3. UTILISONS (UN PEU) NOTRE CHAMELEONULTRA

À présent que nous avons un client qui fonctionne, nous pouvons faire un petit tour du propriétaire. Trois « activités » peuvent être envisagées ici : lire des *tags*, attaquer des *tags* (MIFARE) et émuler des *tags*. Je ne vais pas tout couvrir ici, loin de là, étant donné que la plupart des utilisateurs préféreront sans doute l'outil GUI plus intuitif. Je me contenterai donc d'un simple exemple pour présenter la philosophie du client officiel, en configurant une émulation de *tag* de A à Z.

Commençons par lister ce qui existe avec `hw slot list`. Je vous fais grâce de la sortie qui est identique à la précédente (insérée là pour l'effet dramatique), après la mise à jour du *firmware*. Nous décidons d'utiliser l'emplacement 6 pour émuler un *tag* MIFARE Classic 1K (S50) dont nous possédons l'image sous la forme d'un fichier de 1024 octets nommé « lav.bin ».

Le « concurrent » du ChameleonUltra peut être vu comme étant le Proxmark 3 RDV4 (alias PM3) équipé d'un module BlueShark Standalone, mais en réalité, les objectifs poursuivis par ces deux matériels sont sensiblement différents. Le ChameleonUltra est clairement orienté vers l'émulation alors qu'un Proxmark 3 est dédié à l'analyse. Les deux cependant partagent nombre de fonctionnalités (lecture, écriture, émulation, attaques, etc.), mais le PM3 est capable de sniffer les communications et gère également les tags ISO-14443 type B (14b), ISO-15693, Felica, etc.

Nous commençons donc par rendre cet emplacement *enable*. J'utilise ici le terme anglais, car il y a une distinction à faire entre « activé » au sens *enabled* qui veut dire « disponible pour utilisation » et « actif » au sens *active* qui signifie que c'est l'emplacement actuellement utilisé pour faire une simulation. Nous faisons donc :

```
[USB] chameleon --> hw slot enable -s 6 -e 1
- Set slot 6 enable success.
```

`-s` désigne l'emplacement (attention, la liste commence à `1` et non à `0`) et `-e` permet de choisir *enable* (`1`) ou *disable* (`2`). Notez que le client dispose d'une complétion automatique et d'une option `-h` permettant d'obtenir de l'aide pour toutes les commandes. Nous devons ensuite préciser le type de *tag* à émuler en utilisant la correspondance suivante (à cette date) :

- 1 : EM410X ;
- 2 : Mifare Mini ;
- 3 : Mifare Classic 1k ;
- 4 : Mifare Classic 2k ;
- 5 : Mifare Classic 4k ;
- 6 : NTAG 213 ;
- 7 : NTAG 215 ;
- 8 : NTAG 216.

Nous avons une image de MIFARE Classic 1K et nous utilisons donc la commande :

```
[USB] chameleon --> hw slot type -s 6 -t 3
- Set slot tag type success.
```

Tant que nous y sommes, et pour nous faciliter la vie par la suite lorsque nous aurons oublié ce que nous avons stocké dans le ChameleonUltra, nous pouvons donner un nom (ou *nickname*) à l'emplacement avec :

```
[USB] chameleon --> hw slot nick set -s 6 -st 2 -n mflav1k
- Set tag nick name for slot 6 success.
```

`-s` désigne le numéro de l'emplacement comme précédemment, `-st` (*sense type*) précise la technologie LF (1) ou HF (2) et enfin, `-n` permet de spécifier le nom choisi. Tout étant en place, nous n'avons plus qu'à mettre à jour la flash avec :

```
[USB] chameleon --> hw slot update
- Update config and data from device memory to flash success.
```

Et éventuellement vérifier avec un petit `hw slot list` :

```
[...]
- Slot 6 data (active):
   HF: mflav1k - Mifare Classic 1k
   LF: Empty - Unknown
[...]
```

Nous pouvons ensuite rendre actif cet emplacement avec :

```
[USB] chameleon --> hw slot change -s 6
- Set slot 6 activated success.
```

Ceci a le même effet que d'utiliser les boutons pour choisir un emplacement (et les boutons sont utilisables en même temps que le client). Il ne nous reste plus qu'à charger les données en mémoire et finaliser la configuration. Pour cela, nous utilisons une commande `hf mf` et non `hw slot`. Une commande permet de charger des données « brutes » en mémoire :

```
[USB] chameleon --> hf mf eload -f /chemin/lav.bin -t bin
................................................
...........
- Load success
```

`-f` précise le chemin vers le fichier (`~/` ne fonctionne pas pour désigner votre `$HOME`) et `-t` le type de fichier, qui peut être binaire (`bin`) ou texte/hexadécimal (`hex`). Le type peut être omis si l'extension du fichier est respectivement `.bin` ou `.eml` (les formats supportés par le client Proxmark 3). Mais ce n'est pas tout. En effet, pour que l'émulation fonctionne, nous devons préciser des éléments de configuration que sont l'UID du *tag* et les paramètres d'anticollision (SAK et ATQA). Dans le cas d'un MIFARE Classic avec une UID de 4 octets, ces informations sont présentes dans l'image puisqu'il s'agit des quatre premiers octets pour l'UID, suivi de l'octet de vérification (inutile ici, car calculé), puis l'octet de SAK (*Select AcKnowledge*) à `0x08` (parfois `0x88` en réalité avec certains *tags*) et les deux octets de ATQA (*Answer To reQuest type A*) respectivement à `0x04` et `0x00`. Nous définissons cela avec :

```
[USB] chameleon --> hf mf sim --sak 08 --atqa 0400 --uid 12B178C5
- Set anti-collision resources success
```

Nous vérifions avec :

```
[USB] chameleon --> hf mf info
- UID  Size: 4
- UID  Hex : 12B178C5
- SAK  Hex : 08
- ATQA Hex : 0400
```

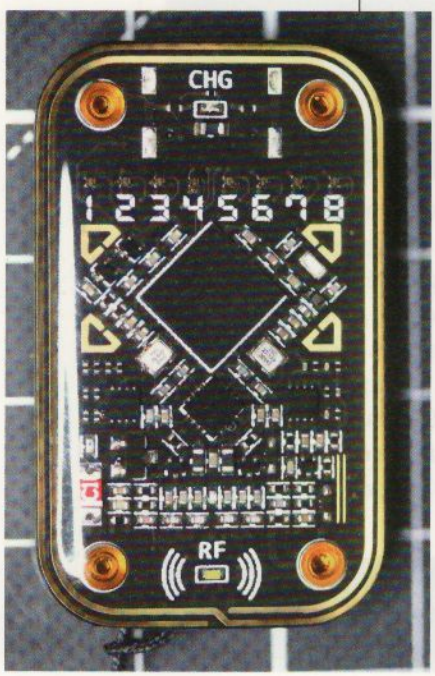
Et nous n'oublions pas de mettre à jour la flash :

```
[USB] chameleon --> hw slot update
- Update config and data from device memory to flash success.
```

Il ne nous reste plus qu'à quitter le client Python et déconnecter le ChameleonUltra pour lire le *tag* émulé en emplacement 6 avec un lecteur et/ou une application Android, par exemple. Dans mon expérimentation, le dispositif utilisant le *tag* d'où provient l'image n'a pas remarqué de différence, pas plus que l'app *NFC Taginfo* de Reasearch Lab Hagenberg, *NFC Tools* de Wakdev ou encore le très connu *MIFARE Classic Tool* (MCT) de Gerhard Klostermeier. Mais l'application TagInfo de NXP signale « *Unknown Mifare class IC, possibly cloned* » ce qui est intéressant (et pas uniquement parce que les dev de l'application n'ont pas lu le mémo précisant que c'est « MIFARE » et non « Mifare »).

Voici pour cette rapide démonstration qui, je vous assure, est plus facile à lire qu'elle a été à produire. La prise en main du ChameleonUltra est, pour l'instant, un peu déroutante. J'avoue que la logique de l'outil CLI en Python et la syntaxe des commandes me paraît très perturbante (non, ce n'est pas parce que je n'aime pas Python). Les numéros d'emplacement débutent à 1, certaines commandes prennent en argument un numéro d'emplacement, d'autres agissent sur l'emplacement actif, les informations sur l'émulateur MIFARE apparaissent avec `hw slot list` mais sont configurés par `hf mf settings`... C'est troublant, mais gageons que ceci comme le reste du projet se résoudra avec le temps. Et puis il y a toujours la possibilité de développer votre propre outil, dans le langage que vous préférez, pour avoir quelque chose qui vous est parfaitement adapté (et peut-être davantage dans un « esprit Unix »).

Un dernier mot cependant sur l'intérêt d'utiliser un ChameleonUltra pour une émulation de *tag* et non une autre solution comme une *Ultimate Magic Card* ou un porte-clés multi-RFID (comme **[11]** ou **[12]**). En réalité, ces deux pseudoéquivalents sont parfaits en comparaison, car le ChameleonUltra combine précisément les avantages des deux. Le porte-clés multi-RFID permet de combiner plusieurs *tags* (3 à 10) en un seul objet, tout en offrant la possibilité de modifier l'UID (selon modèle). L'Ultimate Magic Card (alias Gen4 GTU)



La face avant du ChameleonUltra expose les composants utilisés, ce qui est du plus bel effet. Mais ceci est purement cosmétique, car même si cela protège effectivement le circuit, il devient impossible d'intervenir matériellement sur le produit. Si vous souhaitez un matériel équivalent, offrant ce genre de facilité, c'est vers le ChamleonUltra Dev Kit qu'il faudra vous orienter.

n'est qu'un unique *tag*, mais qui peut être un MIFARE Classic, un MIFARE Ultralight EV1 ou un NTAG21x selon la configuration, tout en offrant des fonctionnalités « spéciales » comme le *shadow mode*. Disposer d'un ChameleonUltra, à terme, reviendra à avoir un jeu de 8 Ultimate Magic Cards qui, de plus, émule aussi les *tags* LF et fait office de lecteur USB. Sachant qu'à ~25 € la Magic Card (dans le meilleur des cas et en cherchant bien, tout en jonglant avec la quantité et le port pseudoaléatoire sur AliExpress), on arrive déjà à 200 €, on comprend mieux pourquoi le ChameleonUltra est à ce point intéressant et n'a pas fini de faire parler de lui.

À propos de fonctionnalités spéciales, l'émulateur de MIFARE Classic est configurable et vous permet d'activer différents modes de fonctionnement. Attention cependant, cette configuration n'est **pas** globale à tous les emplacements concernés (tous les HF pour le moment), même si elle semble ne pas changer entre les emplacements avec un `hw slot list`. En réalité, il s'agit d'un *bug* du script Python (signalé à l'instant par votre humble serviteur), car cette information dépend de l'emplacement actif. Si vous changez d'emplacement avec `hw change -s 6` par exemple (ou les boutons du ChameleonUltra), ce sera la configuration de l'émulateur pour l'emplacement 6 qui sera affichée partout. Nul doute que ceci sera potentiellement corrigé au moment de la publication de l'article, mais cela démontre à quel point toute aide est utile au projet et qu'elle est facile à apporter (même si on n'aime pas Python).

Quoi qu'il en soit, voici ce qui s'affiche et ce qu'il faut savoir :

```
- Mifare Classic emulator settings:
   Detection (mfkey32) mode: disabled
   Gen1A magic mode: disabled
   Gen2 magic mode: disabled
   Use anti-collision data from block 0: disabled
   Write mode: Normal
```

Commençons par le plus évident, étant donné que nous venons de configurer les informations d'anticollision précédemment. Il est possible de directement déduire ces informations du bloc 0 de l'image en flash. Pour activer cette fonction, il suffit d'utiliser `hf mf settings --coll 1` pour les *tags* avec des UID de 4 octets (par opposition à des *tags* avec des UID de 7 octets). Gen1A et Gen2 font référence aux technologies utilisées par les Magic Cards, permettant de changer leur UID (ce qui est impossible avec un *tag* « normal »). Un *tag* « magique » Gen1A permet cette modification en utilisant un enchaînement de commandes spéciales, alors qu'un *tag* Gen2 autorise, tout simplement, l'écriture du bloc 0 comme n'importe quel autre bloc (on parle aussi de *tags* CUID pour *Changeable UID*). Configurer un fonctionnement Gen1A et/ou Gen2 se fait respectivement avec `hf mf settings --gen1a 1` et `hf mf settings --gen2 1`.

« *Detection* » référence une technique d'attaque consistant à solliciter non pas un *tag*, mais un lecteur et en collectant des *nonces*, des valeurs arbitraires générées pseudoaléatoirement, qui peuvent ensuite être utilisées pour déduire les clés (voir **[13]** à la partie « *MFKEY32v2 walkthrough* »). Pour activer la collecte de *nonces*, on utilisera `hf detection enable -e 1`. Notez que les clés calculées/déduites ne sont **que** celles utilisées par le lecteur, pas toutes celles potentiellement configurées dans un *tag* normalement utilisé avec ce dernier.

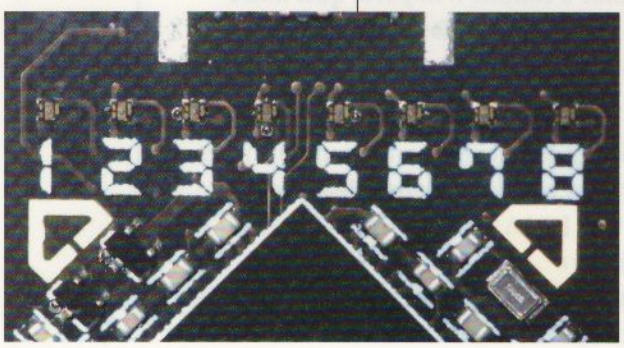
Et enfin, la cerise sur le gâteau, nous avons le mode d'écriture qui peut être :

- *Normal* (`hf mf settings --write 0`), ce qui est écrit par le lecteur est effectivement enregistré ;
- *Denied* (`hf mf settings --write 1`) qui refuse toute opération d'écriture (techniquement, le *tag* émulé est en lecture seule et répond des NACK) ;
- *Deceive* (`hf mf settings --write 2`) pour accuser réception (ACK) de l'opération d'écriture, mais ne pas l'appliquer réellement ;
- et *Shadow* (`hf mf settings --write 3`) pour que le *tags* émulé procède « virtuellement » aux écritures et permette une relecture, mais « en RAM ». Si le *tag* quitte le champ électromagnétique, sa mémoire revient à son état initial.

CONCLUSION

Avant toute chose et même si je peux paraître très critique dans mes propos en début d'article, il est important de bien comprendre que le code, à la fois pour le *firmware* et le client Python, est en permanente évolution. Des ajouts et corrections sont faits régulièrement et à l'heure où cet article est fini d'être rédigé, une partie de ces évolutions peut être disruptive si, comme moi, vous vous lancez dans un développement lié au protocole utilisé par le ChameleonUltra. Ceci pour vous dire clairement que l'achat de ce matériel est un investissement qui va se bonifier dans le temps et à terme valoir son pesant d'or. Pour preuve, je remarque à cet instant précis que Sébastien Dudek, dont le nom ne vous est pas inconnu si vous êtes également lecteur de Misc, vient tout juste d'annoncer sur Twitter (ou « X », ou je ne sais quel nouveau nom inventé par le troll en chef de cette plateforme) qu'il travaille sur l'implémentation de l'émulation MIFARE Ultralight.

Oui, pour le moment le nombre de types de *tags* émulés et/ou lus/écrits est limité. Oui, le fait que le client Python compte les emplacements à partir de 1 est totalement traumatisant (*tousse* Lua *tousse*) et le jeu de commandes un peu désordonné. Et, oui, une application GUI en Flutter implique de nombreuses dépendances à installer (faites un simple `ldd` sur `chameleonultragui` et vous comprendrez). Mais tout ceci sera finalement très secondaire lorsque le ChameleonUltra deviendra pleinement opérationnel et pourra, dans une certaine mesure, remplacer totalement une collection d'Ultimate Magic Cards, et en même temps faire office de lecteur



La résine qui couvre le circuit assure une protection pour un usage courant (même si rayable facilement) et on distingue des bulles d'air emprisonnées à certains endroits. Je ne peux m'empêcher de me questionner sur la façon dont va effectivement vieillir cette finition et s'il n'y a pas un risque, à long terme, pour l'intégrité du produit lui-même (jaunissement, décollage, fissure des soudures dues à une rétractation de la résine, etc.).

RFID/NFC proche de ce qu'offre un Proxmark 3 RDV4 (sans pour autant le remplacer vraiment).

C'est un sujet qu'on pourrait presque qualifier de « en friche » (NDLR : de moins en moins au fil du temps, cf. **[9]**), et la raison pour laquelle je n'ai pas creusé et décrit ici tout ce que fait déjà le ChameleonUltra (attaques, collecte de *nonces*, lecture, etc.). Finalement, deux options s'offrent à vous : vous jeter dans le bain immédiatement pour vivre/suivre l'aventure, ou tout simplement attendre que le support logiciel se bonifie sans votre aide et ne céder à la tentation que plus tard. Personnellement, mon choix était fait dès que le produit est devenu (à nouveau) disponible chez un distributeur officiel, mais ça, c'est peut-être juste moi...

En guise de mots de la fin, je préciserai que la communauté est très active, massive et accueillante, et se retrouve sur le serveur Discord « RFID Hacking » de Iceman **[14]**, en plus de l'écosystème existant via GitHub. Saluons également le fabricant du ChameleonUltra (et du Proxmark 3 RDV4), RRG (*RFID Research Group*), qui a initialement développé le matériel, le *firmware* et l'outil CLI, puis les a remis aux mains de la communauté (prenez-en de la graine, Nordic). Et pour finir, un grand merci à Philippe pour ses conseils avisés et commentaires. **DB**

RÉFÉRENCES

[1] https://github.com/RfidResearchGroup/ChameleonUltra

[2] https://github.com/GameTec-live/ChameleonUltraGUI

[3] https://play.google.com/store/apps/details?id=io.chameleon.ultra

[4] https://apps.apple.com/app/chameleon-ultra-gui/id6462919364

[5] https://play.google.com/store/apps/details?id=com.mtoolstec.mtoolsLite

[6] https://www.nordicsemi.com/Products/Development-tools/nrf-util

[7] https://www.nordicsemi.com/Products/Development-tools/nrf-command-line-tools/download

[8] https://github.com/NordicSemiconductor/pc-nrfutil

[9] https://github.com/RfidResearchGroup/ChameleonUltra/blob/main/docs/development.md

[10] https://developer.arm.com/downloads/-/arm-gnu-toolchain-downloads

[11] https://lab401.com/fr/products/multi-rfid-keyfob

[12] https://fr.aliexpress.com/item/1005002324917760.html

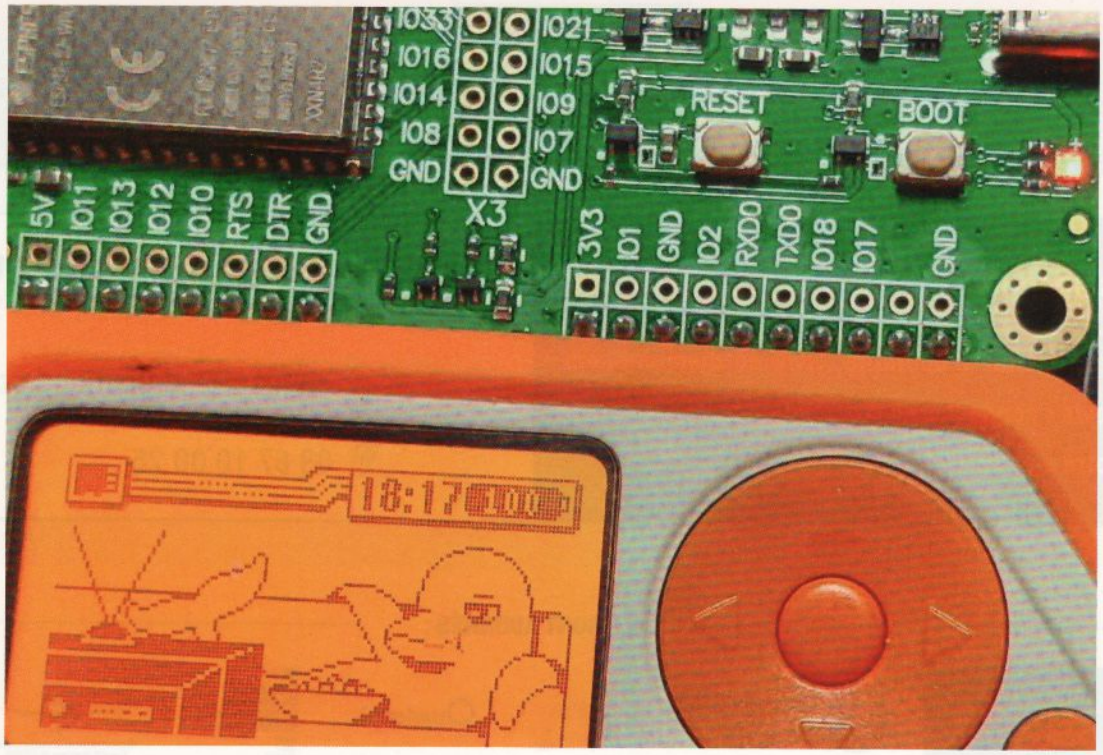
[13] https://github.com/RfidResearchGroup/ChameleonUltra/blob/main/docs/cli.md

[14] https://t.ly/d4_C

FLIPPER ZERO : JOUET OU OUTIL ?

Denis Bodor

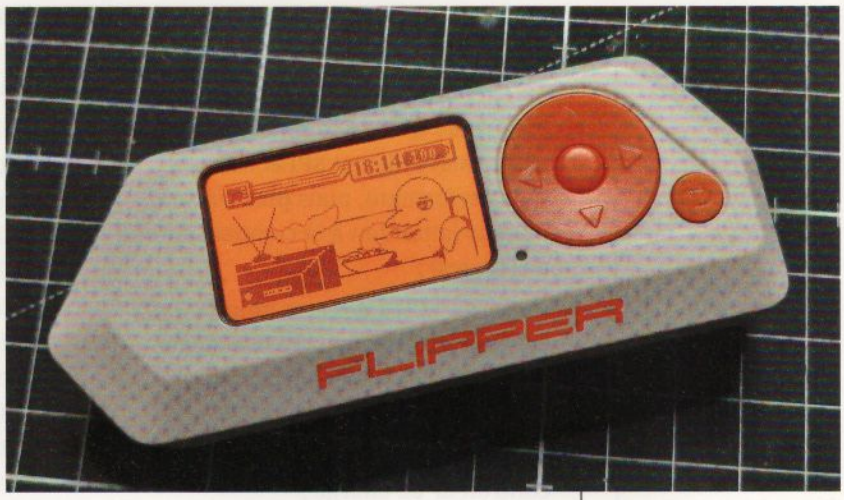
Le Flipper Zero n'est pas une nouveauté, puisque lancé en 2020, mais cet appareil s'est installé, en quelques années, comme un accessoire indispensable pour toute personne curieuse vis-à-vis des technologies RFID, NFC, de radiocommunication (300-348 MHz, 387-464 MHz et 779-928 MHz), des signaux infrarouges, de l'IoT, etc., sur fond de « hacking », voir de « piratage ». Souvent présenté comme un outil de « hacker à capuche », qu'en est-il vraiment ? Ce matériel présente-t-il un réel intérêt pour un bidouilleur, un développeur ou un amateur d'électronique numérique ?



– Flipper Zero : jouet ou outil ? –

Cela fait maintenant plusieurs mois que je joue avec mon Flipper Zero, et davantage encore avec celui que m'avait gentiment prêté un mystérieux « *flic New-Yorkais* » qui se reconnaîtra certainement (merci Toady). L'aura entourant ce produit, si l'on s'en tient à ce qu'on peut lire de-ci de-là de la part de journalistes n'ayant jamais développé la moindre ligne de code de leur vie, se résume à un outil pour pirates anarchistes, excessivement dangereux, qui permet d'ouvrir des coffres-forts, voler des voitures, pirater des portes de garage, changer des prix à la pompe à essence, installer des *malwares* sur n'importe quel PC ou encore rendre vos voisins fous en prenant le contrôle de leur téléviseur.

Certes, l'imagerie accompagnant le Flipper Zero, que ce soit par son aspect, son interface (et celui de ses outils de configuration) ou le côté Tamagotchi du « *Pet dolphin* » n'arrange rien à l'affaire, bien au contraire, et le béotien dépourvu de la moindre once de connaissance technique aura vite fait de voir cela comme l'accessoire faisant du premier venu un Kevin Mitnick en puissance. Pourtant, ce que propose le Flipper Zero n'implémente strictement rien de neuf, si ce n'est de regrouper, dans une moindre mesure, ce que permettent déjà de faire beaucoup mieux d'autres équipements (HackRF, PlutoSDR, Proxmark3, Pi Pico, Arduino, Bus Pirate ou même un simple lecteur RFID/NFC avec les bons utilitaires). En réalité, si controverses il y a autour du Flipper Zero, c'est uniquement parce que la plupart de ceux qui en parlent n'ont pas connaissance de ce qu'il était déjà possible de faire, bien avant son arrivée. En cela, cet appareil est un outil pédagogique hors-norme, qui a le don d'éveiller les utilisateurs aux risques potentiels et de faire réagir les constructeurs pour corriger les failles et les vulnérabilités.



Voici l'appareil qui défraie la chronique depuis quelque temps déjà et pourtant... Il ne s'agit que d'une plateforme de développement basée sur STM32 et équipée de quelques périphériques. Rien de bien nouveau sous le soleil, si ce n'est le format, la compacité et l'imagerie qui l'accompagne.

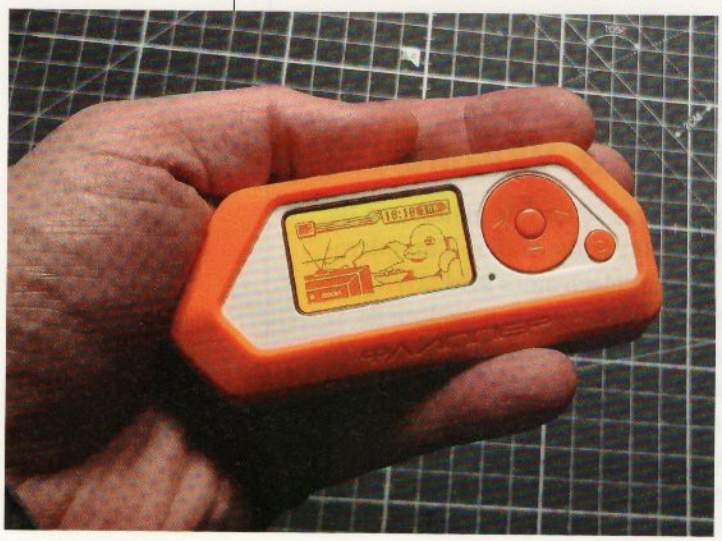
1. QU'EST-CE QUE LE FLIPPER ZERO, VRAIMENT ?

En quelques mots : c'est une plateforme de développement construite autour d'un microcontrôleur STMicroelectronics STM32WB55RG à cœur ARM Cortex-M4. Celui-ci dispose de 1024 Kio de flash et 256 Kio de SRAM, de différents périphériques usuels (SPI, UART, i²c, GPIO, etc.), mais aussi, et surtout, d'un cœur supplémentaire ARM Cortex-M0 dédié aux fonctionnalités réseau, LR WPAN et Bluetooth 5 Low Energy (BLE).

Autour de ce microcontrôleur sont ajoutés :

- un support de stockage MicroSD jusqu'à 64 Gio (4 Gio sont largement suffisants) utilisant SPI et non SDIO plus performant, mais plus consommateur (attention à la compatibilité des cartes) ;
- un écran LCD 1,4 pouce monochrome à rétroéclairage orange de 128×64 pixels piloté par un contrôleur ST7565R interfacé en SPI ;
- un accu LiPo 2000 mAh et son contrôleur de charge ;
- une interface USB-C (périphérique uniquement, pas d'OTG ou USB hôte ici, malheureusement) ;
- un contrôleur NFC ST25R3916 permettant le comportement en lecteur et en émulateur, compatible ISO 14443A/B, FeliCa, ISO 15693 (vicinity), MIFARE Classic, etc. ;
- un support (lecture et émulation) purement logiciel pour les *tags* RFID 125 kHz directement pris en charge par le STM32WB55RG et une antenne bibande intégrée. Le RFID 125 kHz est compatible avec les *tags* EM4100, HID H10301, HID Prox, Idteck, T5577, etc. ;
- un émetteur-récepteur (*transceiver*) Texas Instruments CC1101 avec une gamme de fréquences sous le gigahertz (*Sub-GHz*) très vaste, mais limitée par le *firmware* en fonction des régions pour des raisons légales de conformité aux standards. En l'occurrence, la bande ISM pour l'Europe, de 433,05 à 434,79 MHz et de 868,15 à 868,55 MHz. Note : oui, c'est techniquement possible d'outrepasser cela, mais ceci prouve clairement que le but des créateurs du Flipper Zero n'est pas de violer la loi, de la même façon que Laguiole, Opinel et Breizh Kontell ne fabriquent pas des armes pour tueurs en série ;
- un émetteur-récepteur infrarouge 800-950 nm ;
- un *buzzer* (100 à 2500 Hz), un vibreur et une LED d'état RGB.

Le principal avantage du Flipper Zero se résume par cette image. Avec cet outil, vous avez, au creux de votre main, tout le nécessaire pour explorer les technologies radio ISM, RFI, NFC, Bluetooth, USB device, etc., de façon totalement nomade.



À cela s'ajoute 13 GPIO accessibles (plus masse, 3,3 V et 5 V) au-dessus de l'appareil, 6 boutons pour la navigation et la gestion de l'alimentation, ainsi qu'un connecteur iButton permettant lecture/écriture/émulation en protocole 1-wire. Vous remarquerez l'absence évidente de support Wi-Fi intégré, mais qui peut être ajouté via un module « *WiFi Devboard* » (~50 €) intégrant un ESP32-S2. Ce module, qui se présente sous la forme d'un PCB sans protection permet, de base, la mise à jour du *firmware* ainsi que des fonctionnalités de mise au point (*debug*) via USB ou Wi-Fi grâce à une version dédiée de *Black Magic Probe* (dont je vous vous avais parlé dans les numéros 46 **[1]** et 40 **[2]**).

L'ensemble prend la forme d'un petit boîtier blanc compact de 10 cm de long par 4 cm de large et 2,5 cm d'épaisseur, relativement robuste et d'une centaine de grammes. Une

gaine de protection en silicone (~20 €) est disponible en option ainsi que des protections anti-rayures pour l'écran (~10 € les trois). Le Flipper Zero est vendu environ 200 €, littéralement partout (oui, y compris Amazon).

Comme vous pouvez le voir, il n'y a ici rien de réellement extraordinaire puisqu'il s'agit, plus ou moins, d'un *devkit* STM32 accompagné de quelques périphériques. Ce qui fait la différence avec un montage fait maison utilisant les mêmes éléments, c'est bien entendu le côté « manufacturé » de très bonne qualité, mais aussi, et surtout, l'écosystème logiciel qui a représenté une masse de travail conséquente.

Le Flipper Zero utilise un *firmware open source* qu'il est relativement simple de compiler et de flasher dans le périphérique. Si les développeurs s'en étaient tenus à ça, il y a peu de chance que le produit aurait effectivement gagné en popularité comme il l'a fait ces trois dernières années. En plus du *firmware* lui-même, il est possible d'installer des « applications » tierces sous la forme de paquets ou FAP pour *Flipper Application Package* qui ne sont, en réalité, que de simples binaires ELF s'intégrant dans l'architecture du *firmware* basé sur FreeRTOS. Mais, plus important encore, le Flipper Zero (que je nommerai désormais FZ) dispose également d'une application graphique Qt nommée qFlipper disponible pour Windows, macOS et GNU/Linux (sous forme d'AppImage), ainsi que d'une *Flipper Mobile App* pour smartphones Android et iOS. À cela s'ajoute également une application web directement utilisable dans un navigateur à l'adresse **https://lab.flipper.net**. Les développeurs n'ont pas chômé...

Le grand absent ici, à première vue, semble être un outil en ligne de commande digne de ce nom, mais en réalité qFlipper propose une option `cli` permettant quelques opérations, mais aussi, et surtout, le FZ se présente comme un périphérique série CDC ACM lors de la connexion en USB et une console (115200 8N1) est accessible via un émulateur de terminal (type Minicom, GNU Screen, etc.). De quoi satisfaire les utilisateurs *old school* comme moi.

Le Flipper Zero n'est donc pas un outil de « h4ck3urZ » au sens « BFMTV » du terme, c'est, comme l'indique clairement le site officiel **[3]**, un « *Multi-tool Device for Geeks* ». Autrement dit, c'est un couteau suisse pour **vrais** amateurs de technologie et d'électronique.

2. PRISE EN MAIN RAPIDE DU FLIPPER ZERO

Après réception, votre Flipper Zero devra impérativement être équipé d'une carte microSD et être mis à jour avec le *firmware* le plus récent. L'opération est relativement simple. Commencez par mettre l'appareil sous tension en restant appuyé sur le bouton « retour » (celui à droite du *pad*) durant deux secondes, après avoir inséré la microSD. Naviguez ensuite dans les menus après avoir utilisé le bouton au centre du *pad*, choisissez ***Settings***, ***Storage*** et ***Format SD card*** puis confirmez ***Format*** avec le *pad* vers la droite. Ceci n'est pas strictement nécessaire, car vous pourriez le faire sur PC, mais vous assure d'avoir un système de fichiers FAT32 vierge, exactement comme l'attend le FZ.

Une partie de la configuration du FZ se trouve sur le support de stockage externe et la base du *firmware* en flash. Formater la microSD ne suffit donc pas pour un usage normal, il est nécessaire de réinstaller, ou plus probablement à réception du produit, de mettre à jour le *firmware*. Pour cela, plusieurs solutions sont envisageables :

- installer l'application Android (ou iOS) sur votre smartphone, activer le Bluetooth sur le FZ, appairer le périphérique et, depuis l'application, mettre à jour ;
- télécharger l'application qFlipper pour votre système, connecter le matériel en USB et mettre à jour ;
- vous rendre sur **https://lab.flipper.net**, connecter votre FZ, autoriser l'accès au port série depuis le navigateur et utiliser l'interface web pour mettre à jour ;
- télécharger `flipper-z-f7-update-release-(VERSION ACTUELLE).tgz` depuis **https://update.flipperzero.one/builds/firmware/release/**, extraire le contenu de l'archive et recopier l'ensemble des fichiers à la racine de la microSD avec un PC, puis utiliser l'interface série via un émulateur de terminal pour déclencher la mise à jour avec la commande `install /ext/update.fuf` (depuis cette interface, `/int` est la flash interne et `/ext` est le support amovible).

Quel que soit le procédé utilisé, le FZ va redémarrer pour se mettre à jour et vous pourrez suivre la progression des différentes étapes sur son écran. Au terme de l'opération, vous aurez un appareil prêt à servir. La même procédure s'appliquera pour mettre à jour le *firmware* au fil des publications de nouvelles versions (actuellement 0.92.2).

qFlipper ne ressemble pas à une application classique Qt mais est dans le ton du projet. Cet outil vous permettra de mettre à jour votre firmware, mais aussi de transférer des fichiers depuis et vers le Flipper Zero.

Si vous utilisez GNU/Linux et rencontrez des difficultés pour dialoguer avec le FZ, il s'agit très probablement d'un problème de permissions. Premièrement, le périphérique USB CDC ACM apparaît dans le système via une entrée `/dev/ttyACM` suivie d'un numéro. Avec de nombreuses distributions, ce pseudofichier est lisible et inscriptible avec les utilisateurs du groupe `dialout`. Assurez-vous donc que l'utilisateur courant est bien membre de ce groupe (ainsi que de `plugdev`, c'est toujours utile).

Ceci n'est cependant pas suffisant pour qFlipper qui va scanner le bus USB pour trouver le bon matériel et le port. Les permissions doivent donc être également ajustées pour le périphérique USB lui-même. Là, le plus simple est de demander à *udev* de faire le travail pour vous, directement à la connexion du FZ. Pour cela, ajouter simplement un fichier dans `/etc/udev/rules.d/` (`flipper.rules` par exemple), contenant :

```
ATTRS{idVendor}=="0483",ATTRS{idProduct}=="5740", MODE="0666", GROUP="dialout"
```

Vous pouvez ensuite recharger les règles *udev* puis déconnecter/reconnecter le matériel :

```
$ sudo udevadm control --reload-rules
```

3. SE PASSER D'APPLICATIONS BINAIRES ET DE SITE WEB PUBLIC

Utiliser l'interface série CDC ACM présenté par le FZ lors de sa connexion en USB est une solution pour éviter d'utiliser des binaires précompilés (on ne sait jamais) ou une interface web chargée depuis un serveur que vous ne contrôlez pas. Mais la ligne de commande n'est pas forcément du goût de tout le monde et il existe une solution permettant d'avoir une interface plus conviviale, mais locale.

En effet, le service proposé par **https://lab.flipper.net** peut être exécuté localement dans un conteneur Docker. Pour cela, vous devez disposer d'une machine GNU/Linux (réelle ou virtuelle) capable de faire fonctionner Docker, qui s'avère ici très utile, comme je l'avais évoqué dans un précédent article (voir numéro 43 **[4]**). Pour obtenir tout le nécessaire (ou presque), c'est encore une fois vers GitHub qu'il faut se tourner, en clonant tout simplement le projet derrière *lab.flipper.net* :

```
$ git clone --recurse-submodules https://github.com/flipperdevices/
lab.flipper.net.git
Clonage dans 'lab.flipper.net'...
remote: Enumerating objects: 1610, done.
remote: Counting objects: 100% (619/619), done.
[...]
Réception d'objets: 100% (281/281), 57.96 Kio | 8.28 Mio/s, fait.
Résolution des deltas: 100% (158/158), fait.
Chemin de sous-module 'frontend/flipperzero-protobuf' :
'e5af96e08fea8351898f7b8c6d1e34ce5fd6cdef' extrait

$ cd lab.flipper.net
```

Une fois dans le répertoire, on lancera la création de l'image du conteneur avec :

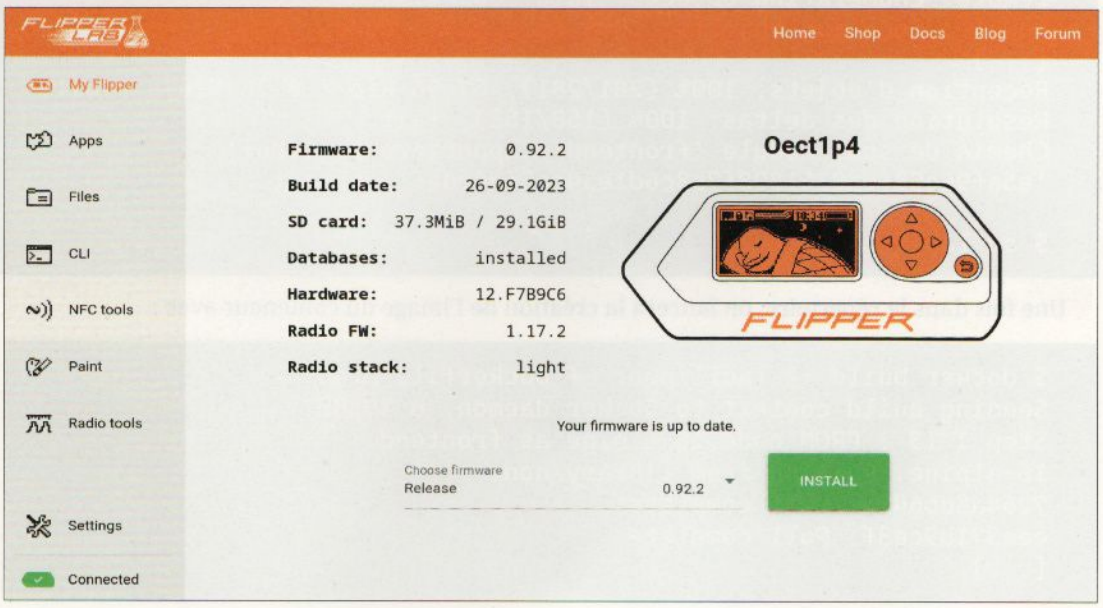
```
$ docker build -t labflipper -f Dockerfile-prod .
Sending build context to Docker daemon  6.395MB
Step 1/13 : FROM node:16-alpine as frontend
16-alpine: Pulling from library/node
7264a8db6415: Pull complete
eee371b9ce3f: Pull complete
[...]
```

```
Removing intermediate container 95f55b930058
---> fed2ccc4b23e
Successfully built fed2ccc4b23e
Successfully tagged labflipper:latest
```

Ceci aura pour effet de télécharger une distribution Alpine pour ensuite y installer NodeJS et les différents éléments nécessaires à l'exécution du service via NGINX. Il suffira alors ensuite d'exécuter le conteneur :

```
denis@newbeast:~/HACK/FLIPPERdev/lab.flipper.net$ docker run -p
127.0.0.1:8000:80/tcp --rm labflipper
2023/10/07 16:26:14 [notice] 1#1: using the "epoll" event method
2023/10/07 16:26:14 [notice] 1#1: nginx/1.24.0
2023/10/07 16:26:14 [notice] 1#1: OS: Linux 6.1.0-10-amd64
2023/10/07 16:26:14 [notice] 1#1: getrlimit(RLIMIT_NOFILE):
1048576:1048576
2023/10/07 16:26:14 [notice] 1#1: start worker processes
2023/10/07 16:26:14 [notice] 1#1: start worker process 7
[...]
```

Les paramètres utilisés redirigeront les requêtes sur l'adresse locale (127.0.0.1), port 8000, vers le service exécuté dans le conteneur. L'option `--rm` permet de supprimer automatiquement le conteneur en fin d'exécution (avec Ctrl+C), mais l'image reste en place (inutile de repasser par la phase de `build`, donc). Ceci fait, il ne vous restera plus qu'à pointer votre navigateur sur **http://127.0.0.1:8000** et vous y trouverez votre interface de gestion du Flipper Zero. Notez que ceci ne fonctionne pas avec Firefox qui n'implémente pas encore le support pour *WebSerial API* (seuls Chrome/Chromium, Edge et Opera sont compatibles) :



Bien entendu, les paramètres habituels concernant l'exécution du conteneur s'appliquent, en particulier concernant le réseau et, selon vos besoins, pourront être ajustés (`--network=host` ou `-p 80:80` par exemple).

4. RECONSTRUISONS LE FIRMWARE

L'environnement de construction et de développement arrive clé en main. Nul besoin de chaîne de compilation tierce ou de SDK spécifique au STM32. Comme avec quelques autres projets, tout le nécessaire est téléchargé, installé localement sans permissions supplémentaires nécessaires et tenu à jour. Tout ce qu'il vous faut sur le système que vous utilisez est Git. L'ensemble de la procédure qui va suivre passe par l'outil `fbt`, FBT signifiant « *Flipper Build Tool* » qui n'est qu'un *wrapper* en shell pour le moteur de production (ou *build tool*, comme Make, CMake, etc.) SCons.

Récupérer et construire le *firmware* officiel se fait en récupérant les sources depuis GitHub via la commande :

```
$ git clone --recursive https://github.com/flipperdevices/flipperzero-firmware
Clonage dans 'flipperzero-firmware'...
remote: Enumerating objects: 94563, done.
[...]
remote: Compressing objects: 100% (9/9), done.
remote: Total 10 (delta 0), reused 10 (delta 0), pack-reused 0
Réception d'objets: 100% (10/10), 4.24 Kio | 4.24 Mio/s, fait.
Chemin de sous-module 'lib/stm32wb_copro/scripts' :
  '781464e628eb694b4c98a79ab524c293f527fc8d' extrait
Chemin de sous-module 'lib/stm32wb_hal' :
  'cfd0dd258cb031c95b2b2d6d04c19f9f625fe3e8' extrait
```

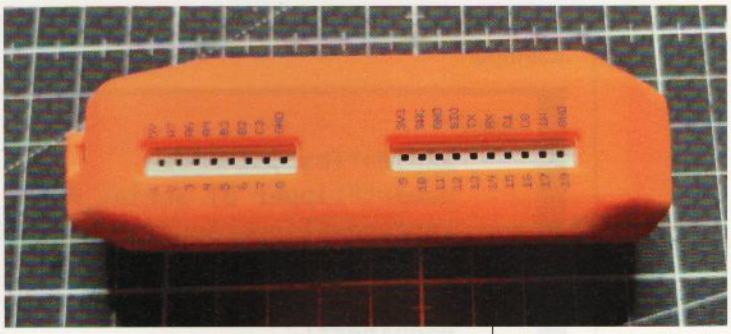
Parmi les quelque 600 Mio téléchargés, nous trouvons non seulement les sources du *firmware*, mais également une douzaine de sous-modules constituant les dépendances de ce dernier (noyau FreeRTOS, support USB, système de fichiers *LittleFS*, bibliothèques cryptographiques, etc.). Mais plus important encore, nous avons `fbt` dont la simple invocation va automatiquement télécharger la chaîne de compilation, si elle n'est pas disponible ou à jour, mettre à jour les sources et les sous-modules depuis GitHub et déclencher la construction :

```
$ ./fbt
Checking for tar..yes
Checking if downloaded toolchain tgz exists..no
Checking curl..yes
Downloading toolchain:
################################## 100.0%
done
Removing old toolchain..done
Unpacking toolchain to '/home/denis/tmp/flipperzero-firmware/toolchain':
```

```
################################ 100.0%
done
Cleaning up..done
  ICONS   build/f7-firmware-D/assets/compiled/assets_icons.c
  PROTO   assets/protobuf/application.proto
  DOLPHIN blocking
  DOLPHIN internal
[...]
API version 39.1 is up to date
[...]
Loaded 92 app definitions.
Firmware modules configuration:
Service:
    bt, cli, dialogs, dolphin, desktop, gui,
    input, loader, notification, power, storage
System:
    updater_app, storage_move_to_sd
Archive:
    archive
Settings:
    bt_settings, notification_settings, storage_settings,
    power_settings, desktop_settings, passport, system_settings,
    about
StartupHook:
    crypto_start, rpc_start, infrared_start, nfc_start,
    subghz_start, lfrfid_start, ibutton_start, onewire_start,
    bt_start, power_start, loader_start, locale, storage_start,
    updater_start, storage_move_to_sd_start
MenuExternal:
    subghz, lfrfid, nfc, infrared, gpio, ibutton, bad_usb, u2f
Package:
    basic_services, main_apps, main_apps_on_start, settings_apps,
    system_apps
Firmware size
.text         556744 (543.70 K)
.rodata       147664 (144.20 K)
.data           1624 (  1.59 K)
.bss            8504 (  8.30 K)
.free_flash   342208 (334.19 K)
Linking build/f7-firmware-D as latest built dir (./build/latest/)
        BIN     build/f7-firmware-D/firmware.bin
        HEX     build/f7-firmware-D/firmware.hex
firmware.bin:  173 flash pages (last page 45.31% full)
        DFU     build/f7-firmware-D/firmware.dfu
2023-10-09 09:43:56,325 [INFO] Firmware binaries can be found at:
        dist/f7-D
```

On préférera cependant lancer `fbt` en utilisant différents paramètres ainsi qu'une cible de compilation (voir **[5]**). Les variables d'environnement à votre disposition (ou argument pour `fbt`) sont :

- `FBT_NOENV` : demande à ne pas utiliser d'environnement de construction local (via `scripts/toolchain/fbtenv.sh`) et donc d'utiliser une chaîne de compilation disponible par ailleurs ;
- `FBT_TOOLCHAIN_PATH` : précise l'emplacement de la chaîne de compilation à utiliser ;
- `FBT_NO_SYNC` : ne synchronise pas avec GitHub, ceci permet de maintenir des sources locales dans une certaine version ;
- `FBT_VERBOSE` : augmente le niveau de verbosité ;
- `DEBUG` : permet de construire une version de mise au point du *firmware* (plus lourd) ;
- `COMPACT` : optimise la construction de manière à réduire la taille.



L'un des côtés du Flipper Zero propose une série de connecteurs au pas de 2,54 mm permettant d'ajouter des montages externes, soit pour étendre matériellement ses fonctionnalités soit pour permettre la mise au point de programmes.

Généralement, on choisira donc de créer un *firmware* de mise à jour directement installable comme décrit précédemment, avec `./fbt COMPACT=1 DEBUG=0 VERBOSE=1 updater_package`. Au terme de l'opération, un message vous indiquera où se trouvent les éléments binaires qui vous intéressent :

```
2023-10-09 09:56:16,827 [INFO] Complete
2023-10-09 09:56:16,887 [INFO] Firmware binaries can be found at:
       dist/f7-C
2023-10-09 09:56:16,888 [INFO] Using guessed radio address 0x080D7000,
  verify with Release_Notes or specify --radioaddr
2023-10-09 09:56:16,960 [INFO] Use this directory to self-update
your Flipper:
       dist/f7-C/f7-update-local
2023-10-09 09:56:17,127 [INFO] Bundling SDK
```

Le fichier à utiliser avec qFlipper ou l'interface web est le `flipper-z-f7-update-local.tgz` se trouvant dans `dist/f7-C`. Si, en revanche, vous souhaitez mettre à jour via l'interface série CDC ACM, les fichiers à copier, y compris le `.fuf` se trouvent dans `dist/f7-C/f7-update-local/`.

4.1 Applications

`fbt` vous permet également de construire des applications complémentaires sous la forme de fichiers `.fap` (*Flipper Application Package*). Pour cela, placez simplement les sources de l'application que vous souhaitez construire dans le sous-répertoire `applications_user/`. S'il s'agit d'un projet externe, comme par exemple l'application de démonstration Bluetooth/série de *maybe-hello-world* disponible sur GitHub **[6]**, vous pouvez cloner directement le dépôt dans ce répertoire.

Jetez ensuite un œil au fichier `application.fam` normalement présent dans les sources. Exemple :

```
App(
    appid="fbs",
    name="Flipper BT Serial App",
    apptype=FlipperAppType.EXTERNAL,
    entry_point="fbs_app",
    stack_size=1 * 1024,
    requires=[
        "bt",
        "gui",
    ],
)
```

Le fichier FAM (*Flipper Application Manifests*) décrit les propriétés du composant et comment il doit être intégré dans le système/*firmware*. Chaque composant est identifié par un ID, ici `fbs`, qui peut alors être utilisé pour provoquer la construction avec :

```
$ ./fbt fap_fbs
        CC      applications_user/fbs/fbs.c
        ICONS   build/f7-firmware-D/assets/compiled/assets_icons.c
        PROTO   assets/protobuf/application.proto
[...]
API version 39.1 is up to date
  APPMETA build/f7-firmware-D/.extapps/fbs.fap
  FAP     build/f7-firmware-D/.extapps/fbs.fap
  FASTFAP build/f7-firmware-D/.extapps/fbs.fap
  APPCHK  build/f7-firmware-D/.extapps/fbs.fap
```

Le FAP obtenu, placé dans un sous-répertoire de `build/`, pourra être copié sur le FZ, sur la microSD (`ext/`), dans un sous-répertoire de `apps/`. Il est cependant plus simple de directement installer l'application depuis les sources, après avoir connecté votre FZ, en utilisant :

```
$ ./fbt launch APPSRC=applications_user/fbs
        SDKCHK  firmware/targets/f7/api_symbols.csv
API version 39.1 is up to date
[...]
2023-10-09 14:52:35,031
[INFO] Sending "build/f7-firmware-D/.extapps/fbs.fap"
to "/ext/apps//fbs.fap"
<100%, chunk  1 of  1 @ 52.53 kb/s
2023-10-09 14:52:35,108
[INFO] Launching app: /ext/apps//fbs.fap
```

Le message est trompeur, car l'application n'est pas exécutée mais simplement installée. Notez également que, selon les sources utilisées, l'emplacement de l'application ne sera pas nécessairement spécifié par son auteur (comme ici) et celle-ci se retrouvera alors à la racine de `apps/` et non dans un sous-répertoire.

Vous pouvez placer autant de sources d'applications que vous le désirez dans `applications_user/` et compiler l'ensemble avec une unique commande :

```
$ ./fbt fap_dist
[...]
  INSTALL dist/f7-D/debug_elf/t_rex_runner_d.elf
  INSTALL dist/f7-D/apps/Debug/uart_echo.fap
  INSTALL dist/f7-D/debug_elf/ibutton_d.elf
  INSTALL dist/f7-D/debug_elf/speaker_debug_d.elf
```

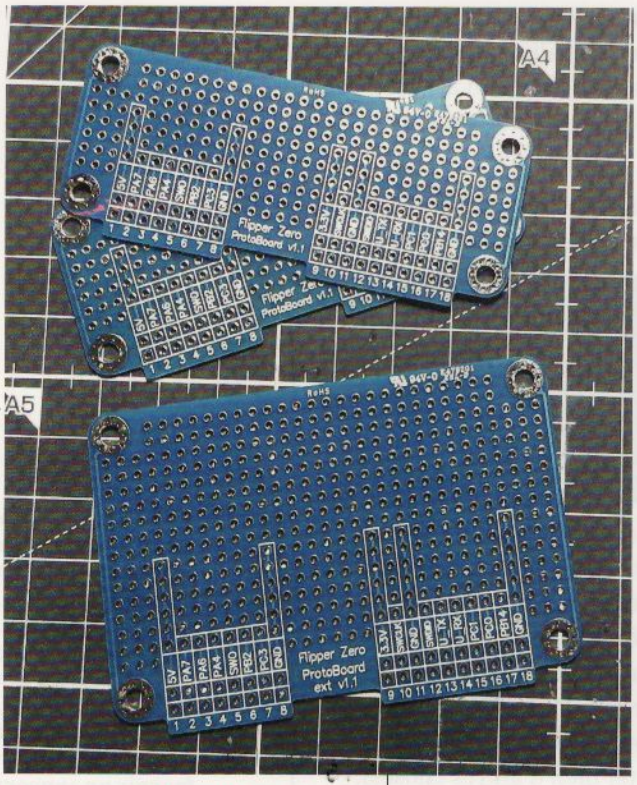
Vous retrouverez alors les `.fap` dans `dist/f7-D/apps/` et pourrez soit les copier tous sur la microSD, soit choisir ceux qui vous intéressent dans la liste et les installer individuellement avec la méthode que vous préférez. Notez qu'en utilisant la cible `updater_package`, vous intégrerez toutes ces applications dans la mise à jour consécutive. Il existe plusieurs « catalogues » d'applications, les deux plus connus étant « *flipperzero-good-faps* » du projet officiel **[7]** et « *all-the-plugins* » de *xMasterX* **[8]**. Le dépôt « *Awesome Flipper* » sur GitHub **[9]** est également une fantastique source pour les applications (et bien d'autres choses).

Si lors de l'exécution, vous obtenez un message « *Preload failed API version mismatch* », ceci provient certainement du fait que vous venez d'utiliser les sources (ou le SDK) d'une version du *firmware* mais que votre appareil en exécute une autre. Il est impératif de compiler vos applications avec un environnement correspondant à la version du *firmware* que vous utilisez. Ce qui nous amène au point suivant...

La WiFi Devboard du Flipper Zero est un simple circuit imprimé équipé d'un module ESP32-S2. Il permet, à la base, de faciliter de debug du firmware et/ou des applications, mais peut également être reprogrammé pour fournir un support (limité) en termes de communications Wi-Fi (projet « ESP32Marauder »).

5. FIRMWARES ALTERNATIFS

Le *firmware* officiel n'est pas le seul disponible. Peu après la mise en vente du Flipper Zero, certains développeurs ont entrepris de *forker* le projet et de maintenir leur propre version en mettant l'accent sur des éléments précis et en suivant des lignes



N'importe quel circuit fait maison peut venir compléter un Flipper Zero et lui ajouter de nouvelles interfaces de communication, mais ceci sera grandement facilité en optant pour des cartes de prototypage avec une forme déjà parfaitement adaptée.

directrices particulières. La racine reste cependant le système de construction original et les commandes qui viennent d'être détaillées fonctionneront sans problème avec ces dérivés.

Le premier projet de ce type est le « *unleashed-firmware* » **[10]** qui, comme son nom l'indique, vise à exploiter davantage de fonctionnalités du matériel, en particulier au niveau radio en se débarrassant des limitations en termes de gamme de fréquences (ce qui, soyons clairs, ne vous dispense aucunement de respecter la loi, et le *fork* est très explicite à ce propos). *Unleashed* se veut être le plus stable des *firmwares personnalisés* et ressemble à s'y méprendre à l'original (graphiquement). Parmi les différences notables, on trouve la suppression des limitations radio, l'inclusion d'algorithmes de *rolling code* **[11]** comme KeeLoq, l'intégration de plus de codes IR pour les télécommandes, davantage de clés dans les dictionnaires RFID MIFARE Classic et un certain nombre d'ajouts cosmétiques, comme l'horloge sur l'écran d'accueil.

Si vous êtes cependant à la recherche d'un *firmware* plus personnalisable pour adapter l'interface à vos besoins/envies, ne cherchez pas plus loin que le *firmware* Xtreme ou XFW **[12]**. Là, la personnalisation est le maître mot et vous pourrez à loisir changer l'interface sans avoir à recompiler quoi que ce soit. Animations, menus, organisation de l'interface, activation/désactivation de fonctionnalités... tout y passe. Il existe même une « gestion de thèmes » (*asset-packs*) permettant d'installer des jeux d'animations directement depuis le Web **[13]**. Personnellement, je n'ai que faire des animations qui sont, pour moi, une perte de temps (et de flash). Mais je conçois que le FZ puisse également avoir un aspect plus ludique pour certains.

RogueMaster **[14]** est un *fork*, mais cette fois de *Unleashed*, également très axé sur les animations et les thèmes graphiques, mais reprenant également un certain nombre de fonctionnalités améliorées d'*Unleashed*. C'est sans doute le *firmware* le plus « rock'n'roll » (à défaut de meilleur terme) dans le sens où les dernières « fonctionnalités » à la mode sont rapidement intégrées. Je n'aime cependant pas l'aspect un peu trop « mercantile » du projet, incitant au sponsoring Patreon dans le but d'avoir les derniers ajouts avant les autres. Je comprends que le

fait de maintenir un tel projet demande des ressources qui ne sont pas gratuites, mais c'est surtout la façon de faire qui ne va pas dans le sens de mes préférences.

CONCLUSION ET AVIS

Oui, c'est vrai, il y a clairement un côté *script kiddies* lié au Flipper Zero. C'est impossible à ignorer tant on voit de « démonstration de hacker » ouvrant le garage du voisin, affichant des *popups* sur un iPhone ou clonant sa carte de bibliothèque. Ce sont ces mêmes démonstrations, sans aucune connaissance technique préalable, qui font dire aux journaleux de bas étage que l'objet est dangereux et terrifiant. Seulement voilà, les personnes qui utilisent bêtement le Flipper Zero de cette manière sont précisément l'antithèse du hacker. Ils ont simplement dépensé 200 € pour un jouet qu'ils ne comprennent pas et qu'ils utilisent à 10 % de ses capacités pour impressionner leurs amis.

Et c'est bien triste, car le matériel est intéressant, non pour ses capacités qui n'ont finalement rien de nouveau, mais en raison de sa compacité et de sa transportabilité. Il ne remplacera pas un Proxmark3 côté RFID/NFC, pas plus qu'il ne le fera pour un équipement SDR tel qu'un HackRF, mais c'est effectivement un couteau suisse technologique. Et comme un couteau suisse, il fait un peu tout, mais rien à la perfection. Au final, même si le coût est conséquent, c'est effectivement un outil que j'utilise régulièrement et je suis très heureux de l'avoir ajouté à ma panoplie. **DB**

RÉFÉRENCES

[1] **https://connect.ed-diamond.com/hackable/hk-046/esp32-black-magic-probe-debug-swd-en-wi-fi**

[2] **https://connect.ed-diamond.com/hackable/hk-040/black-magic-probe-deboguez-vos-codes-avec-facilite**

[3] **https://flipperzero.one/**

[4] **https://connect.ed-diamond.com/hackable/hk-043/execution-d-anciennes-applications-binaires-sur-un-gnu-linux-recent-l-exemple-quartus-ii**

[5] **https://github.com/flipperdevices/flipperzero-firmware/blob/dev/documentation/fbt.md#fbt-targets**

[6] **https://github.com/maybe-hello-world/fbs.git**

[7] **https://github.com/flipperdevices/flipperzero-good-faps**

[8] **https://github.com/xMasterX/all-the-plugins**

[9] **https://github.com/djsime1/awesome-flipperzero**

[10] **https://github.com/DarkFlippers/unleashed-firmware**

[11] **https://fr.wikipedia.org/wiki/Code_tournant**

[12] **https://github.com/Flipper-XFW/Xtreme-Firmware**

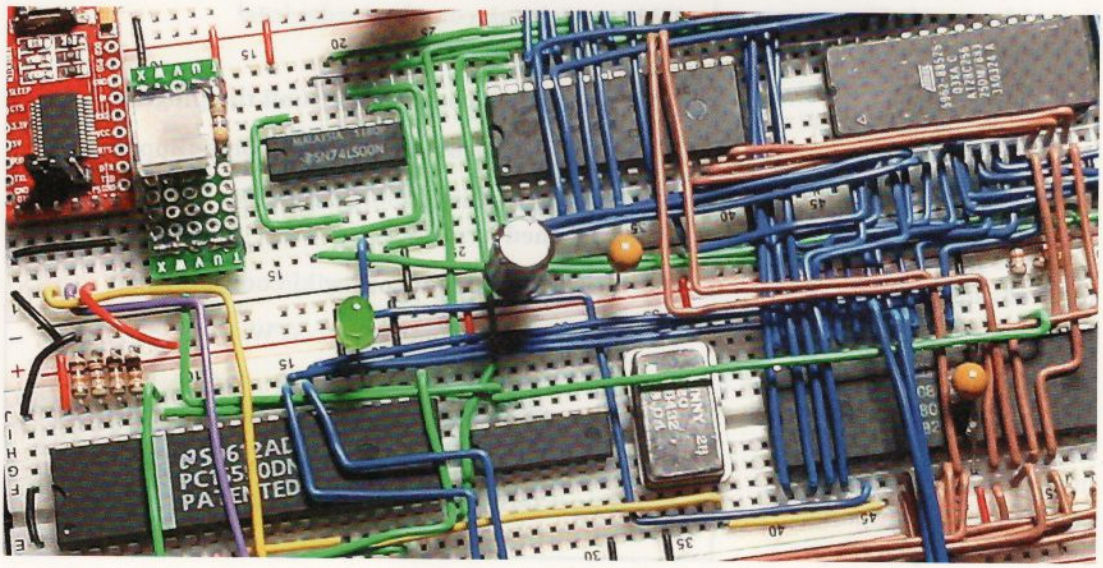
[13] **https://flipper-xtre.me/asset-packs/**

[14] **https://github.com/RogueMaster/flipperzero-firmware-wPlugins**

Z80 & Z180 : L'ASSEMBLEUR C'EST BIEN, LE C C'EST MIEUX

Denis Bodor

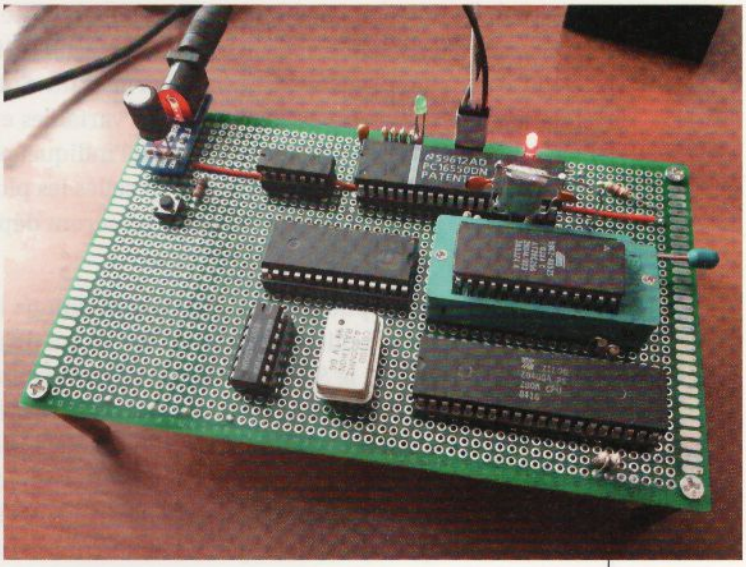
Dans les précédents articles, qu'il s'agisse de ceux concernant notre ordinateur 8 bits sur platine à essais ou de la carte industrielle vintage à base de Z180, nous nous en sommes tenus à utiliser le langage le plus proche du matériel, l'assembleur. Mais l'informatique moderne ne serait pas ce qu'elle est aujourd'hui si les développeurs d'antan s'étaient limités à un langage, certes puissant, mais peu abstrait et souvent considéré comme difficile à apprendre et à maîtriser. Nous allons donc marcher sur les pas de ces développeurs et élever notre niveau de programmation en utilisant le plus plaisant des langages, celui-là même créé par Dennis Ritchie en 1972, le C ! Mais pour cela, il y a un peu de travail à faire…



– Z80 & Z180 : l'assembleur c'est bien, le C c'est mieux –

Notre précédent code assembleur nous a permis de faire communiquer notre ordinateur sur platine via son interface série, que nous avions dûment configurée en initialisant quelques registres. De la même façon, nous l'avons également fait avec la carte industrielle et son UART intégrée au processeur Z180. Mais afficher ainsi, sur l'écran de l'émulateur de terminal série, une suite de « A » ou même une chaîne stockée en ROM n'est pas très intéressant. Bien sûr, nous pourrions poursuivre en assembleur et obtenir quelque chose de plus interactif, mais à quel prix ! Au-delà de la simple envie de revenir en territoire connu avec un langage de haut niveau tel qu'on l'utilise sur Arduino, Pi Pico ou ESP32, basculer vers le C offre un avantage qui tient en un seul mot, qui est également sa raison d'être historique : la portabilité.

Bien entendu, la portabilité n'est pas quelque chose de magique et d'absolu, et doit être vue comme une notion relative. Vous ne pouvez pas prendre un code C pour un PC GNU/Linux et tout simplement le compiler pour le RP2040 d'une carte Pi Pico ou pour ESP32. Mais la masse de travail nécessaire pour adapter ce code sera bien moindre que s'il s'agissait de code assembleur à migrer de x86 vers ARM, AVR ou Xtensa LX6. L'idée derrière la portabilité du C, et d'autres langages, est de minimiser les modifications à apporter, non de les réduire à néant. Un très bon exemple est celui que nous allons découvrir prochainement puisque nous allons, encore une fois, nous contenter d'envoyer des données via l'UART pour les afficher dans un émulateur de terminal. Mais,



et c'est un très important « mais », nous allons le faire avec un code que vous connaissez déjà, puisqu'il s'agit tout simplement de la fonction `printf()`. Et ce, exactement de la même manière que vous le feriez sous GNU/Linux, FreeBSD, Windows, le SDK Pico ou l'ESP-IDF des microcontrôleurs Espressif. C'est ce qui se cache derrière ce `printf()` qui devra être adapté au Z80 et à notre architecture utilisant l'UART 16550 (ou l'UART intégrée, dans le cas du Z180).

Pour réaliser ce petit tour de force et donc nous ouvrir, à terme, toute une galaxie de codes préexistants, nous devons tout d'abord satisfaire un certain nombre de besoins du langage et de la chaîne de compilation...

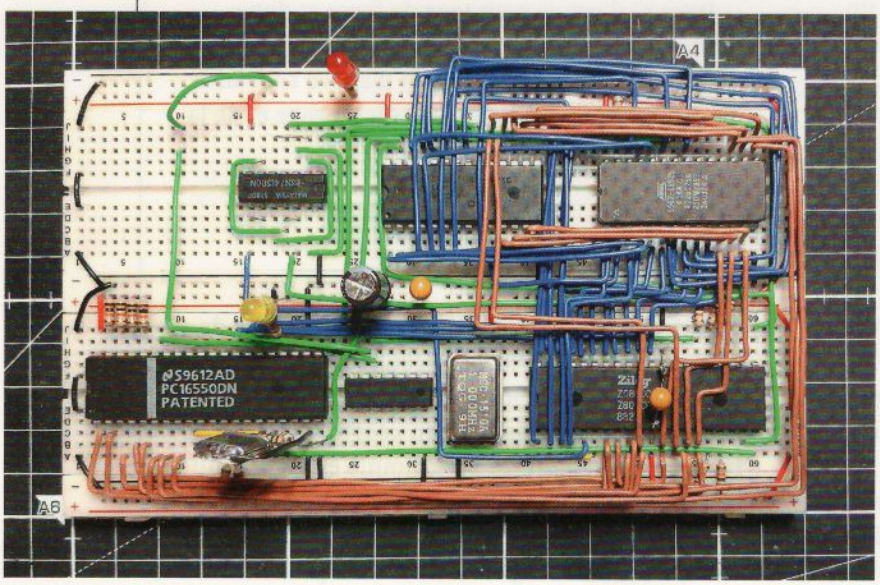
1. RAM, TAS ET PILE

Notre ordinateur 8 bits sur platine dispose de 32 Kio de ROM (EEPROM AT28C256) et de 32 Kio de SRAM. Notre premier code (celui affichant « AAAAAA ») ne faisait aucun usage de la RAM puisque l'ensemble du programme était contenu et surtout exécuté directement depuis la ROM. Tout ce que nous avions à faire était de placer des valeurs dans des registres de l'UART pour animer

l'ensemble. En C, les choses sont sensiblement différentes, la RAM est indispensable et utilisée de deux façons différentes. Nous avons d'une part le tas (*heap*), permettant d'allouer des morceaux de mémoire désignés par des variables et d'autre part la pile (*stack*) qui, comme son nom l'indique, permet d'empiler et de dépiler des données où celles les plus récemment empilées sont également celles qui seront dépilées en premier (LIFO pour *Last In First Out*).

La gestion de la pile avec un processeur comme le Z80, et bien d'autres, est facilitée par le processeur qui dispose d'un registre particulier, le *Stack Pointer* ou SP, contenant toujours l'adresse mémoire du haut de la pile. Ce registre est, bien entendu, de 16 bits et n'est généralement pas accédé directement (sauf optimisation), mais change en fonction de l'utilisation d'instructions comme `push` (empiler) et `pop` (dépiler). Une chose importante à bien garder en mémoire est le fait que la pile grossit vers le bas ou, en d'autres termes, quand on empile, SP est **décrémenté**. Son fonctionnement est inverse de celui du tas, où une zone mémoire sera allouée en partant de l'adresse la plus basse vers une adresse plus haute.

Un autre intérêt de disposer d'une pile est de pouvoir faire usage de sous-routines. Dans le numéro précédent, nous avons vu l'instruction `jp` permettant de procéder à un saut à une adresse et donc de faire des boucles. En ayant une pile, il devient possible d'utiliser `call` qui est également un saut, mais cette instruction empile la valeur du compteur ordinal (registre PC) plus trois (taille d'un `call`), avant de le charger avec l'adresse de destination. L'instruction `ret` peut ensuite être utilisée pour reprendre l'exécution juste après le `call`, puisque celle-ci dépile l'adresse sur la pile et la charge dans PC pour revenir au flux d'exécution principal. Ceci n'est, bien entendu, possible que si nous avons de la RAM et une pile configurée (SP initialisé avec une valeur arbitraire).



Cependant, dès lors qu'on parle de C, il ne s'agit plus simplement d'empiler et de dépiler des adresses pour le compteur ordinal, mais également d'utiliser la pile et/ou certains registres pour passer des arguments aux fonctions qui sont appelées. `call` et `ret` ne s'occupent pas de cela et il n'y a rien dans le jeu d'instructions du Z80 en rapport direct avec ce type de mécanique. C'est au compilateur de décider de la façon de traduire votre code en C vers l'assembleur et donc de définir comment les arguments sont transmis. Ceci s'appelle une *calling convention* qui n'a que peu d'importance si vous vous en tenez au C (comme avec Arduino, Pi Pico, ESP-IDF ou un code pour PC). Mais ce n'est pas notre cas ici, car nous souhaitons configurer et utiliser l'UART via les

instructions `in` et `out`. Ceci implique un morceau d'assembleur, en plus de celui que nous allons voir dans un instant, fournissant les fonctions C `inb` et `outb`. En résumé, nous devons donc faire un pont entre assembleur et C, et donc connaître la *calling convention* à utiliser. Et celle-ci dépend de la version de SDCC !

2. CALLING CONVENTION

En effet, à partir de la version 4.1.2, SDCC change sa *calling convention* pour presque toutes les plateformes supportées, dont la famille Z80. Jusqu'à cette version, le compilateur n'utilisait que la pile pour passer les arguments des fonctions. Si celle-ci est écrite sous la forme d'une routine en assembleur, son premier travail sera de récupérer ces éléments, sachant que le `call` utilisé, lui aussi, empile une adresse de 16 bits (deux octets, donc). Pour illustrer le mécanisme utilisé, imaginons une simple ligne de C appelant la fonction `outb` que nous devons écrire en assembleur :

```
outb(UART_IER, 0);
```

Celle-ci place `0x00` dans le registre `UART_IER` de l'UART 16550 pour désactiver les interruptions. Nous pouvons compiler le code C (dans `prem.c`) avec `sdcc -mz80 --verbose -c -o prem.rel prem.c` pour créer un fichier objet `prem.rel`. Mais plus important, nous obtenons également un `prem.lst` qui est le listing assembleur correspondant. Pour cette ligne de C, nous y voyons ceci :

```
                             60 ;prem.c:11: outb(UART_IER, 0);
0000 AF            [ 4]   61         xor     a,a
0001 57            [ 4]   62         ld      d,a
0002 1E 01         [ 7]   63         ld      e,#0x01
0004 D5            [11]   64         push    de
0005 CDr00r00      [17]   65         call    _outb
```

La quatrième colonne est le numéro de ligne et je désignerai donc chaque instruction avec ce numéro. Ligne 60, le `xor` est une astuce rapide pour mettre le registre A à zéro, puis cette valeur est chargée dans le registre D. Ligne 63, c'est la valeur littérale `0x01`, le premier argument de la fonction, qui est placée dans le registre E, et le tout, D et E, est ensuite placé sur la pile avec `push` (qui ne peut empiler que des valeurs 16 bits avec `push bc`, `push de`, `push hl` ou `push af`).

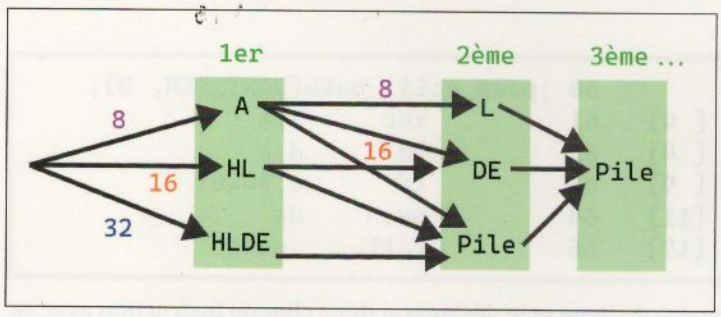
Le même code C avec un SDCC plus récent, comme 4.2.0 tel que disponible pour des systèmes à jour (la 4.3.0 date de juin dernier), donne un résultat très différent :

```
                             60 ;prem.c:11: outb(UART_IER, 0);
000000 2E 00       [ 7]   61         ld      l, #0x00
                          62 ;       spillPairReg hl
                          63 ;       spillPairReg hl
000002 3E 01       [ 7]   64         ld      a, #0x01
000004 CDr00r00    [17]   65         call    _outb
```

On retrouve notre `xor`, mais le second argument est simplement placé dans le registre L et le premier dans A, avant d'appeler la routine `_outb` via `call`. La pile n'est pas utilisée. Et ceci est parfaitement détaillé dans le « *SDCC Compiler User Guide* » **[1]** page 72, chapitre 4.3.3. Le choix des registres utilisés est fonction de la taille et du nombre d'arguments. Ceci est important, car la façon d'écrire nos routines `_in` et `_out` dépendra totalement de cette convention et donc de la version de SDCC. Ici, nous partirons du principe que vous utilisez une version stable récente (4.2.0 ou 4.3.0, donc) et nos prototypes de fonction (placés dans `util.h`) ressembleront à ceci :

```
extern void outb(const char port, const char byte);
extern unsigned char inb(const char port);
```

`outb` prend deux arguments, tous deux de 8 bits. Le premier sera donc passé dans le registre A et le second dans L. Pour `inb`, l'argument est aussi de 8 bits et passé dans A, mais cette fonction retourne une valeur de 8 bits, qui devra être placée dans A également. Un schéma dans la documentation résume cela très clairement :



Puisque nous avons tous les éléments en main, nous pouvons alors écrire nos deux routines assembleur dans `util.s` :

```
        .module util
        .area   _CODE

_outb::
        push bc
        ld c,a
        out (c),l
        pop bc
        ret

_inb::
        push bc
        ld c,a
        in a,(c)
        pop bc
        ret
```

Notez les labels suffixés d'un double double point afin que les symboles soient exportés et donc accessibles depuis d'autres unités de compilation (dont notre code principal). Remarquez également que l'instruction `out` ne peut s'utiliser qu'avec le registre C (ou un littéral) pour désigner l'adresse du port concerné, nous devons donc copier le contenu reçu dans A vers C avant de l'utiliser, et il en va de même pour `in`. De ce fait, et c'est là une tâche qui incombe à la routine, il est nécessaire de sauvegarder le contenu des registres utilisés/modifiés afin de ne pas perturber le fonctionnement normal du code. C'est pour cela que nous devons empiler les valeurs dans BC (combinaison des registres B et C), puis dépiler avant l'utilisation de `ret` pour revenir au flux d'exécution normal avec les registres dans le même état que celui de départ.

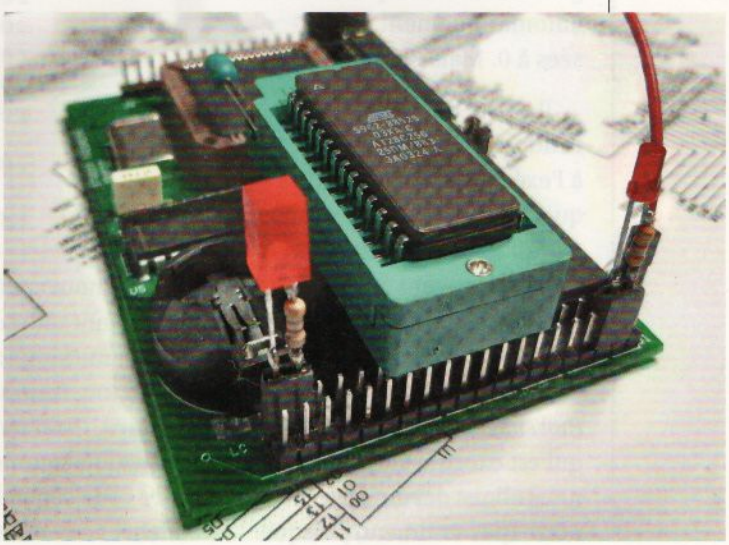
À titre d'information (et/ou d'exercice de compréhension), voici un petit morceau de code qui devrait vous rappeler quelque chose et que nous utiliserons plus tard :

```
int putchar(int c) {
        /* attendre que l'UART soit prête */
        while( !(inb(UART_LSR) & 0x20) ) { ; }
        /* envoi du caractère */
        outb(UART_RBR, c);
        return(0);
}
```

Et voici sa traduction en assembleur par le compilateur SDCC (version 4.3.0) :

```
; ---------------------------------
; Function putchar
; ---------------------------------
_putchar::
;prem.c:21: while( !(inb(UART_LSR) & 0x20) ) { ; }
00101$:
        push    hl
        ld      a, #0x05
        call    _inb
        pop     hl
        bit     5, a
        jr      Z, 00101$
;prem.c:23: outb(UART_RBR, c);
        xor     a, a
        call    _outb
;prem.c:24: return(0);
        ld      de, #0x0000
;prem.c:25: }
        ret
```

La question est « *Pourquoi ne voit-on pas le registre L chargé avec la valeur de* `c` *?* » Je vous laisse chercher, mais je l'ai dit précédemment...



Notez également que les anciennes versions de SDCC, et plus exactement des fichiers d'entête qui l'accompagne (`sdcc/include/stdio.h`) utilisaient un prototype différent pour `putchar()` : `void function (char fixed)`. Ceci a été modifié après la version 3.5.0 afin de se conformer au standard et est une bonne raison de garder votre compilateur, et vos codes, à jour, sous peine d'avoir des surprises désagréables (vécu).

3. PRÉPARER LE TERRAIN POUR LE C

Quel que soit le système ou la plateforme utilisée pour développer en C, il existe un certain nombre de choses que vous ne voyez pas lorsque l'exécution de votre programme démarre. Il est commun de désigner `main()` comme le point d'entrée d'un tel programme et donc en déduire que c'est ce par quoi débute l'exécution. Même si ceci peut être une notion suffisante pour la plupart des développements (Arduino, Pico, ESP, etc.), nous sommes ici à un tout autre niveau et nous devons donc faire nous-mêmes ce qui est invisible habituellement. Partons donc d'une affirmation aussi vraie que simple : en C, les variables globales (ou statiques) sont automatiquement initialisées à 0. Mais qui fait ça ?

Il existe un ensemble de routines de démarrage lié à l'exécution de n'importe quel programme en C et son travail est d'initialiser tout le nécessaire avant de passer la main à la fonction principale du code (`main()`). Cet ensemble est appelé le « *C RunTime* » ou crt0. C'est lui qui est exécuté en premier et nous allons devoir écrire le nôtre (pas vraiment, puisque SDCC fournit un crt0 pour toutes les plateformes, mais « faire » c'est « comprendre »). Ce sera, en compagnie des routines `inb`/`outb`, la seule partie en assembleur de notre projet.

Si vous avez bien suivi en début d'article, vous aurez sans doute compris que l'une des initialisations à effectuer concerne la pile et le registre SP. Avant de passer la main au C, nous devons initialiser SP avec une adresse correspondant à un point sur le haut de la pile, et donc l'adresse la plus haute en RAM. Avec notre architecture, ceci correspond à l'adresse `0xffff` (rappelez-vous, la pile grossit vers le bas). Nous pouvons reprendre notre exemple en assembleur du numéro précédent et l'adapter. Ce nouveau fichier, `mycrt0.s`, débute donc par :

```
        .module crt0
        .globl  _main

        .area   _HEADER (ABS)

        .org    0x0

        jp      init

        ; Reset vector
        .org    0x08
        reti
        .org    0x10
        reti
        .org    0x18
        reti
        .org    0x20
        reti
        .org    0x28
        reti
        .org    0x30
        reti
        .org    0x38
        reti

        .org    0x40
init:
        ld      sp,#0xffff  ; stack init
```

Petite nouveauté par rapport au code précédent, nous avons maintenant une succession d'emplacements mémoire consistant en une simple instruction `reti` signifiant la fin du traitement d'une routine d'interruption logicielle. Nous verrons cela plus tard, mais sachez simplement que lorsqu'on utilise une instruction comme `rst`, c'est ici que l'exécution se poursuit. Notre code cependant saute cette table de vecteurs et reprend au label `init` où nous initialisons notre pointeur de pile.

Notre crt0 est ensuite complété de :

```
        call    gsinit
        jp      _main

        ; Ordering of segments for the linker.
        .area   _HOME
        .area   _CODE
        .area   _INITIALIZER
        .area   _GSINIT
        .area   _GSFINAL
        .area   _DATA
        .area   _INITIALIZED
        .area   _BSEG
        .area   _BSS
        .area   _HEAP

        .area   _CODE
```

Ici, rien de bien extraordinaire, comme nous avons une pile fonctionnelle, nous pouvons utiliser `call` pour appeler la routine `gsinit` avant de sauter au `main()`, le point d'entrée de notre code en C. C'est cependant `gsinit` qui constitue la partie la plus importante et la plus intéressante du crt0 :

```
        .area   _GSINIT
gsinit::
        ; variables globales init par défaut
        ld      bc, #l__DATA
        ld      a, b
        or      a, c
        jr      Z, init_data
        ld      hl, #s__DATA
        ld      (hl), #0x00
        dec     bc
        ld      a, b
        or      a, c
        jr      Z, init_data
        ld      e, l
        ld      d, h
        inc     de
        ldir
init_data:
        ; variables globales initialisées
        ld      bc, #l__INITIALIZER
        ld      a, b
        or      a, c
        jr      Z, gsinit_next
        ld      de, #s__INITIALIZED
        ld      hl, #s__INITIALIZER
        ldir
gsinit_next:
        .area   _GSFINAL
        ret
```

Notez tout d'abord le fait que nous ne sommes plus dans `_CODE`, mais dans une section spécifique. Et ceci a son importance, car ici, tout est une question de sections (*area*) et de symboles. Lorsque le programme est chargé et exécuté, il se divise en plusieurs morceaux. Voici, par exemple, ce que nous apprend la table des symboles (fichier `.sym` produit pendant la compilation) :

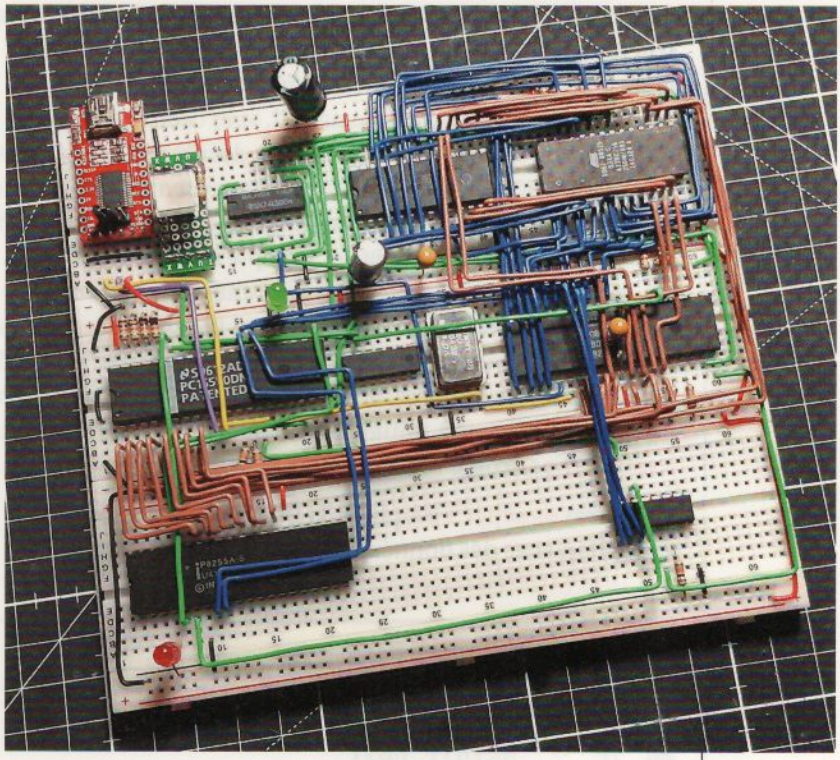
```
Area Table
   0 _CODE            size    300   flags    0
   1 _DATA            size      2   flags    0
   2 _INITIALIZED     size      2   flags    0
   3 _DABS            size      0   flags    8
   4 _HOME            size      0   flags    0
   5 _GSINIT          size      0   flags    0
   6 _GSFINAL         size      0   flags    0
   7 _INITIALIZER     size      2   flags    0
   8 _CABS            size      0   flags    8
```

Nous retrouvons également les différents symboles en relation, dans un fichier `.map` :

```
       Value  Global
       -----  --------------------
     00000000  .__.ABS.
     00000000  l__BSEG
     00000000  l__BSS
[...]
     00000002  l__HEADER7
     00000002  l__INITIALIZED
     00000002  l__INITIALIZER
     00000003  l__HEADER0
     00000004  l__DATA
     00000009  l__HEADER8
     00000028  l__GSINIT
     00000100  s__CODE
     000003FF  l__HEAP
     00000FDF  l__CODE
     000010DF  s__HOME
     000010DF  s__INITIALIZER
     000010E1  s__GSINIT
     00001109  s__GSFINAL
     00008000  s__DATA
     00008004  s__INITIALIZED
     00008006  s__BSEG
     00008006  s__BSS
     00008006  s__HEAP
     00008405  s__HEAP_END
```

La structure mémoire d'un programme en C se divise en sections : *text* pour le code, *data* pour les données initialisées, *BSS* pour les données non initialisées, le tas et la pile. Le compilateur classe les éléments dans les différentes sections en fonction du contenu de votre programme et crée différents symboles correspondant à ces informations. Une variable globale non initialisée explicitement n'est rien d'autre qu'un emplacement mémoire désigné par un nom. Notre tâche sera de l'initialiser à zéro. Par ailleurs, une variable initialisée est également un emplacement mémoire (en RAM, ou en ROM si `const`), mais qui doit être « chargé » avec une donnée précise avant utilisation. Lorsque vous déclarez une variable globale avec, par exemple, `uint16_t toto = 0x42`, `toto` est un emplacement en RAM, mais `0x42` est une valeur qui doit forcément se trouver quelque part, persistante à une coupure d'alimentation. Là, le travail du crt0 n'est pas de mettre l'emplacement à zéro, mais de copier la donnée d'un endroit (en ROM) à un autre (en RAM) **avant** l'exécution du code provenant du code source C.

Les emplacements à initialiser sont désignés par `INITIALIZED` et les



valeurs correspondantes sont les `INITIALIZER`, lorsqu'il s'agit d'initialisations explicites. Lorsqu'on parle, en revanche, de variables globales initialisées à zéro par défaut, c'est `DATA` dont il est question. On retrouve cela dans la table des symboles, accompagné d'un préfixe `s__` pour l'emplacement de départ (`s` pour *start*) et `l__` pour la taille (`l` pour *length*).

Dans notre exemple de sortie, nous comprenons alors que les données initialisées explicitement débutent à 0x8004 (`s__INITIALIZED`), celles initialisées par défaut sont à 0x8000 (`s__DATA`), les données non initialisées sont à 0x8006 (`s__BSS`) et le tas à la même adresse (`s__HEAP`). Pourquoi ? C'est simple, il suffit de regarder `l__BSS`, qui est à zéro. Nous n'avons pas de variables non initialisées. Nous avons par contre une variable globale initialisée et `l__INITIALIZED` comme `l__INITIALIZER` indique `2`, correspondant à deux octets d'un `uint16_t` (`toto`). Plus intéressant encore, `s__INITIALIZER` est à 0x10df, c'est une adresse en ROM et c'est là où se trouve notre 0x42 (ou plus exactement `0x0042`).

Sur cette base, nous pouvons alors comprendre ce que fait le code d'initialisation (similaire à celui que fournit SDCC dans le fichier `share/sdcc/lib/src/z80n/crt0.s`). À commencer par l'initialisation à zéro des variables globales :

```
ld  bc, #l__DATA  ; charge la taille de _DATA
ld  a, b
or  a, c          ; techniquement, c'est un B or C
jr  Z, init_data  ; B et C à zéro ? on arrête
ld  hl, #s__DATA  ; adresse de _DATA dans HL
ld  (hl), #0x00   ; on met 0 à l'adresse de _DATA
dec bc            ; taille de _DATA moins 1
ld  a, b
or  a, c          ; BC à zéro (même astuce)
jr  Z, init_data  ; oui, c'est fini
ld  e, l
ld  d, h
inc de
ldir
init_data:
```

La fin de la routine est astucieuse, car elle repose sur l'instruction `ldir`, qui nécessite davantage d'explications que je ne pourrais en donner en commentaire. `ldir` transfère un octet de l'adresse pointée par HL vers celle pointée par DE, puis incrémente HL et DE tout en décrémentant BC. Si le contenu de BC n'est pas zéro, l'opération est automatiquement répétée. Il s'agit donc littéralement d'un `memcpy()` qu'il suffit d'initialiser via HL, DE et BC, et la copie s'opère toute seule. Ici, le contenu pointé par HL est le premier emplacement, que nous venons d'initialiser à 0x00, et DE est l'adresse juste après. Nous copions donc le contenu d'une adresse vers celle immédiatement après et ainsi de suite jusqu'à arriver au bout du nombre d'octets à initialiser. On peut voir cela comme un `memset()` par copie récurrente.

Le vrai `memcpy()` est plutôt la phase suivante, l'initialisation avec des données, de `s__INITIALIZER` vers `s__INITIALIZED` :

```
ld  bc, #l__INITIALIZER
ld  a, b
or  a, c
jr  Z, gsinit_next
ld  de, #s__INITIALIZED
ld  hl, #s__INITIALIZER
ldir
gsinit_next:
.area   _GSFINAL
ret
```

Ici, BC contient la taille des données d'initialisation, DE l'adresse de destination et HL celle d'origine pour la copie. Et, encore une fois, c'est `ldir` qui fait tout le travail à notre place. L'assembleur est un langage merveilleux...

Ceci constitue l'ensemble de notre crt0 et nous pouvons à présent ne nous intéresser qu'à la partie en C. Mais ça, c'est pour la prochaine fois...

4. C'EST TOUT POUR LE MOMENT

Finalement, nous n'avons pas fait ce C, aujourd'hui... Ne soyez pas déçu, tout le travail préliminaire, qui était assez conséquent, est fait, et la suite ne sera que dans ce langage (sauf exception, ou plus exactement « interruptions »). Il est très agréable de pouvoir toucher ainsi du doigt des mécanismes qui sont, par ailleurs, masqués même en travaillant avec des microcontrôleurs comme les ARM ou même avec de simples Atmel AVR. Ne vous y trompez pas cependant, tout ceci, crt0, initialisation de variables, petits bouts d'assembleur, sont bel et bien présents. Tout autant qu'avec des plateformes plus complexes reposant sur un système d'exploitation. Il n'y a pas de magie et tout finit par s'expliquer, dès lors que l'on creuse suffisamment. C'est juste qu'avec un Z80, il est plus facile de creuser...

La prochaine fois, nous mettrons un peu d'ordre dans tout cela en facilitant la compilation via un `Makefile` et, bien sûr, nous écrirons notre premier code en C (enfin !), qui sera bien plus interactif que le simple affichage de `AAAAAA...`. Et pour les petits curieux (impatients ?), oui, le fait d'avoir un `putchar()` sous-entend forcément celui de pouvoir utiliser `printf()`. **DB**

RÉFÉRENCE

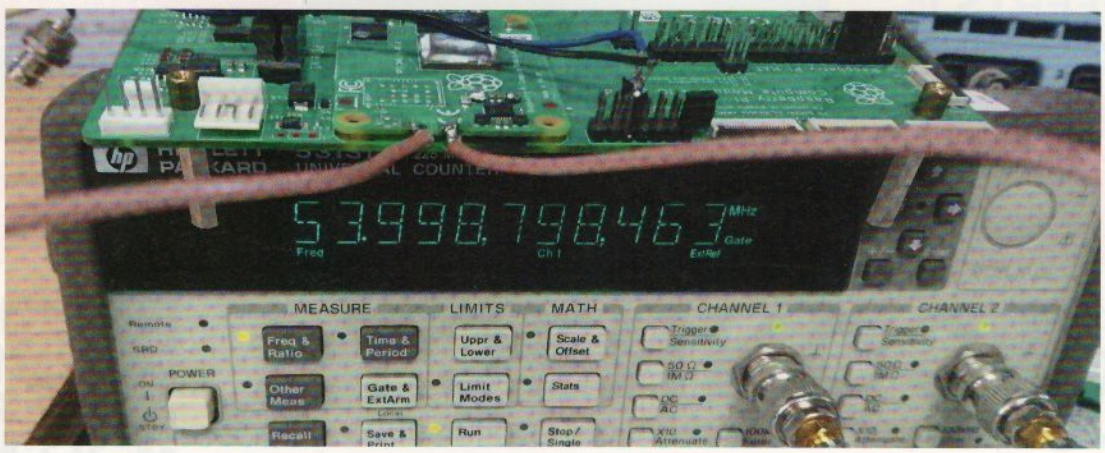
[1] **https://sdcc.sourceforge.net/doc/sdccman.pdf**

SYNCHRONISATION D'ORDINATEURS PAR RÉSEAU INFORMATIQUE POUR LA DATATION SOUS GNU/LINUX :

NTP, PTP ET GPS SUR RASPBERRY PI COMPUTE MODULE 4

Jean-Michel Friedt, FEMTO-ST temps-fréquence

Nombre d'outils, à commencer par make, s'appuient sur la date d'accès aux fichiers pour décider de leur obsolescence. Dans le cadre d'intercomparaisons d'horloges, nous effectuons des acquisitions par radio logicielle sur divers sites géographiquement distincts et nous nous interrogeons sur la date d'acquisition avec une résolution aussi élevée que possible. Que veut dire « élevée » et quel niveau de synchronisation pouvons-nous espérer entre deux ordinateurs exécutant GNU/Linux ? Nous conclurons avec la nécessité de corriger l'erreur de l'oscillateur qui cadence le processeur et démontrerons comment quelques composants passifs sur Compute Module 4 permettent d'atteindre ce résultat.



1. INTRODUCTION

Le transfert de temps est un enjeu dans nombre d'applications technologiques : même si une société moderne ne saurait survivre sans se mettre d'accord sur l'heure (« je suis en retard à mon rendez-vous »), les contraintes sont réduites – un étudiant ne se considère pas en retard jusqu'à 15 minutes après le début du cours – par rapport aux enjeux technologiques de la datation des transactions boursières (qui a obtenu le titre boursier en premier ? Fig. 1), de la production distribuée d'énergie (les pertes en injectant un signal alternatif sur un réseau électrique sont en $\sin(\phi)$ avec $\phi = 2\pi f \tau$ la phase du signal injecté donc son retard τ à fréquence f), de la synchronisation des réseaux de communications téléphoniques ou informatiques **[1, 2]** notamment pour leur sécurité quand une clé cryptographique doit devenir obsolète avant de pouvoir être dupliquée, à la localisation ou la synthèse de diagramme de rayonnement d'antennes distribuées. Rappelons par exemple qu'aux 143,05 MHz du RADAR GRAVES, la période du signal est 7 ns donc tout retard aléatoire de plus de 60 ps se traduit par un déphasage aléatoire de plus de 3°, avec un impact du retard sur le déphasage d'autant plus important que la fréquence est élevée. Formellement, P. Boven rappelle comment les fluctuations des oscillateurs locaux cadençant des récepteurs de radio-télescopes agencés en réseaux d'antennes dégradent leur capacité de corrélation et donc d'extraction du signal du bruit **[3]**. Nous viserons donc à asservir le temps entre deux ordinateurs distants de plus de 300 m à mieux que la microseconde pour les transactions boursières **[4]** ou la nanoseconde pour les applications de RADAR distribué.

Nombre de protocoles de synchronisation existent : NTP (*Network Time Protocol* décrit dans RFC5905 à **https://datatracker.ietf.org/doc/html/rfc5905**), PTP (*Precise Time Protocol* décrit dans RFC8173 à **https://datatracker.ietf.org/doc/html/rfc8173**), White Rabbit (**https://ohwr.org/project/white-rabbit/wikis/home**), ou signaux satellitaires de navigation qui sont avant tout du transfert de temps pour permettre la trilatération du récepteur et la correction de son horloge locale au temps commun de la constellation de satellites. Nous allons expliciter lors de leur mise en œuvre ces techniques de synchronisation de résolution croissante, mais aussi de complexité croissante. Cette discussion n'aurait aucun intérêt si elle se limitait aux propriétaires d'horloges atomiques ou de masers à hydrogène, sources de fréquences ultra-stables inaccessibles au commun

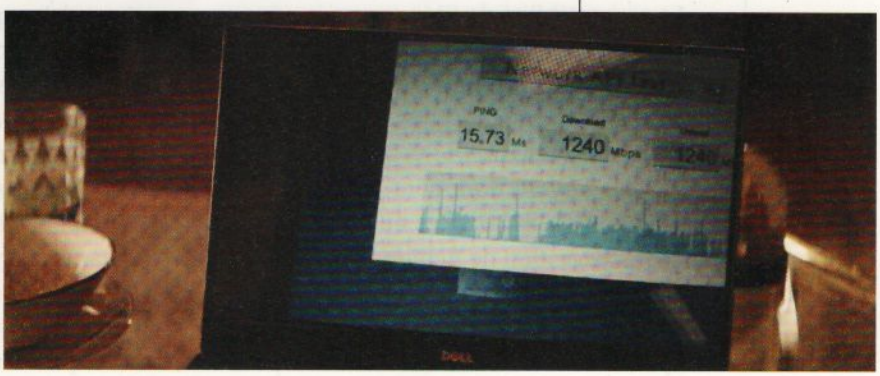


Figure 1 : Extrait du film The Hummingbird Project qui illustre la débauche de moyens pour réduire les délais de communication dans les transactions boursières. Vincent @ 5820 s : « it's not the destination that's important, it's the people we meet and the lessons we learn ». Et pourtant, ils ont réussi à passer sous les 16 ms de temps de propagation d'un signal sur réseau informatique, l'enjeu du film. Évidemment pour une production grand public, les auteurs du film n'ont pu s'empêcher de faire une erreur en utilisant « M » pour le préfixe de milli.

des mortels. En particulier, alors que PTP nécessite des interfaces Ethernet physiques dédiées capables d'horodater les paquets échangés, il s'avère que la Raspberry Pi Compute Module 4 (CM4), et non pas la Raspberry Pi 4, est équipée des périphériques matériels nécessaires à sa mise en œuvre tel qu'expliqué dans **[5]** : nous décrirons donc comment asservir une CM4 sur un signal issu de récepteurs GPS.

Tous ces concepts sont strictement inutiles si je ne peux les utiliser en pratique sur mon système d'exploitation favori : les signaux électriques d'une impulsion par seconde (1-PPS) et l'oscillation à 10 MHz sont inexploitables lorsque je tape la commande `echo toto > fichier` et que je veux savoir la date de sauvegarde du fichier. Que sont ces protocoles et quels sont leurs impacts sur un système informatique distribué en termes de synchronisation ? Comment garantir quel est le fichier le plus récent lorsque de multiples copies sont disponibles ?

La compatibilité de CM4 avec PTP rend donc cette discussion accessible à tout amateur, et change le scénario d'utilisation par son accessibilité et sa simplicité. Voyons donc comment qualifier les latences du système de gestion de fichiers par le noyau Linux et son asservissement sur une source de temps externe.

Disponibilité d'interfaces Ethernet compatibles PTP

D. Bodor informe que les cartes mères munies d'interfaces Ethernet compatibles PTP ne sont pas si rares, puisque les Minisforum Ryzen 9 et 5 ainsi que ASUS Z8NA-D6 bi-Xeon supportent ce protocole. Cependant, une carte mère de PC comporte une multitude de sources de cadencement (résonateurs et oscillateurs), des périphériques dont le rôle est difficile à identifier tant le nombre de contrôleurs est important, alors que cette analyse est limpide sur une CM4. Par ailleurs, le risque financier de détruire une CM4 lors de la manipulation des oscillateurs est considérablement réduit par rapport à la carte mère de PC.

NDLR : La toute récente Raspberry Pi 5 supporte également PTP sur son interface Gigabit Ethernet

2. NTP : ASSERVISSEMENT LOGICIEL PAR ÉCHANGE DE TEMPS

Le transfert de fréquence est un problème trivial : une onde continue dont toutes les périodes sont aussi similaires que possible à leurs voisines est transmise, et le récepteur compare sa copie locale de l'oscillateur avec ce signal reçu, par exemple au travers d'un mélangeur – un composant qui multiplie deux signaux nommés oscillateur local LO et signal radiofréquence RF pour fournir :

$$A_{RF}\cos(\omega_{RF}t) \cdot A_{LO}\cos(\omega_{LO}t) \propto A_{RF}A_{LO}\cos((\omega_{RF}-\omega_{LO})t)$$

après un filtre passe-bas pour éliminer la somme des composantes spectrales – et cherche à annuler cette différence sous hypothèse que les amplitudes A_{RF} et A_{LO} sont constantes, quitte à saturer un comparateur pour s'en assurer. Cette opération s'appelle la **syntonisation**, c.-à-d. ajuster une **fréquence** par rapport à une autre. Nous verrons que technologiquement, cette opération n'est pas si simple, car nombre de systèmes numériques sont cadencés avec un oscillateur de fréquence fixe et le contrôle de cet oscillateur par ledit système informatique n'est pas possible, sauf au travers du réglage grossier qu'est `/sys/devices/system/cpu/cpu0/cpufreq/scaling_governor` pour décider si le processeur réduit sa cadence pour

baisser la consommation ou augmente sa cadence pour traiter plus d'informations. Un écart de fréquence entre les oscillateurs cadençant deux ordinateurs distants, inévitable compte tenu de la technologie des oscillateurs contraints par un résonateur piézoélectrique soumis aux aléas géométriques, de température ou de vieillissement, se traduit rapidement par une dérive du temps : un écart de 1 ppm entre deux oscillateurs (donc une fréquence relative de 10^{-6} qui est aussi l'écart relatif de temps puisque $f = 1/t \Rightarrow df/f = -dt/t$) se traduit au bout de 5 secondes par un écart de 5 µs ou au bout d'une journée de 260 ms. Cela peut sembler faible, mais 3 secondes au bout d'une semaine commencent à être visibles.

Le transfert de temps est bien plus complexe et donc intéressant : une onde électromagnétique se propage dans une ligne de transmission ou une fibre optique à environ 66 % de sa vitesse dans le vide ou v=200 m/µs. Ainsi, se synchroniser à Besançon sur l'émetteur des horloges atomiques de Mainflingen qui cadencent DCF77 à environ 400 km induit immédiatement un retard de $400 \cdot 10^3/v \approx 2000$ µs ou quelques millisecondes, le délai ionosphérique étant un cas intermédiaire entre le guide d'onde et la propagation en espace libre (v=300 µm/s). Afin d'atteindre une synchronisation submicroseconde pour deux interlocuteurs distants de plus de quelques centaines de mètres, la seule solution est de jouer au ping-pong, en se relançant des messages dont la date d'émission et réception est connue dans le référentiel de chaque horloge des deux interlocuteurs, et sous hypothèse de symétrie du canal de communication (le temps de communication de l'interlocuteur 1 vers 2 est le même que de 2 vers 1), le double temps de vol est mesuré et donc retranché des mesures. Aligner ainsi le temps des interlocuteurs s'appelle la **synchronisation**, et sa mise en œuvre par un protocole de ping-pong s'appelle le *two-way* puisqu'il impose que les deux interlocuteurs soient à la fois émetteurs et récepteurs **[6]**.

One-way ou two-way ?

On pourrait se demander à ce point de l'exposé comment GPS peut marcher pour transmettre le temps de chaque satellite à un récepteur uniquement, qui par ailleurs n'émet jamais vers les véhicules spatiaux GPS pour une mesure *two-way*. La réponse tient en la diversité spatiale : tous les satellites sont synchronisés entre eux (via les horloges de l'observatoire naval américain USNO à Washington) et transmettent leur copie du temps GPS (en réalité, leur temps propre et leur écart à GPS, mais c'est un détail sauf si on veut leurrer le temps **[7]**). Ainsi, en trouvant la solution au moindre carré avec au minimum 4 satellites de la position et de l'écart de temps entre l'horloge locale et l'horloge GPS, nous pouvons nous affranchir des quelque 67 ms (au zénith, plus à l'horizon) que met l'onde électromagnétique à parcourir les quelque 20000 km, le vrai problème étant la variation de vitesse avec la densité d'électrons dans l'ionosphère ou l'indice optique de la troposphère qui font varier la vitesse de sa valeur nominale de 300 m/µs. Cela semble facile, mais le passage des pseudoranges (temps de vol satellite-sol) vers une solution de position et de temps (PVT) reste un problème que nous ne maîtrisons pas sans l'aide d'une bibliothèque telle que RTKLib.

L'implémentation la plus simple de ces concepts est NTP, dans laquelle toutes ces opérations de synchronisation sont effectuées au niveau logiciel. Comme peu d'ordinateurs généralistes sont capables d'ajuster finement leur fréquence de cadencement, il n'y a pas de syntonisation, juste l'observation et la correction du retard entre les deux interlocuteurs, retard qui ne saurait jamais s'annuler si les oscillateurs cadençant les

processeurs ne sont pas exactement à la même fréquence. Même si ces ordinateurs étaient cadencés par d'excellentes horloges atomiques, il suffirait qu'ils soient à des altitudes différentes pour que leur temps local dérive selon les lois enseignées par Einstein. Le problème est désespéré sans un protocole de synchronisation.

Cette compréhension de NTP a deux implications :

1. Les performances en termes de synchronisation sont dépendantes de la stabilité des latences du réseau informatique propageant les messages, et en particulier sa symétrie – par exemple une liaison tantôt terrestre, tantôt satellitaire ne sera pas du tout appropriée.
2. Entre deux corrections, l'oscillateur local est libre et dérive.

Afin de vérifier ces affirmations, nous avons assemblé l'expérience suivante qui sera reprise par la suite pour qualifier tous les mécanismes de synchronisation envisagés : un serveur de temps et fréquence supposé parfait (voir plus loin « *White Rabbit* » en section 4) cadence un récepteur de radio logicielle Ettus Research X310 muni d'une interface BasicRX qui se contente de transmettre le signal d'entrée sur les convertisseurs analogiques-numériques configurés pour échantillonner à 5 Méchantillons/s (en réalité, les convertisseurs échantillonnent à 200 Méchantillons/s et déciment le flux, sans importance ici), soit une résolution temporelle de 200 ns. Nous connectons la référence de temps issue de White Rabbit (1-PPS) sur l'entrée du récepteur X310 que nous configurons pour se cadencer sur 10 MHz et 1-PPS externe. La X310 est configurée en `Time Source: External` et `Frequency Source: External` avec une date de départ dans le futur, par exemple `Start Time: 0.2 s` afin de démarrer l'acquisition sur le prochain front montant du PPS issu du *switch* White Rabbit. Une CM4 déclenche la mesure en lançant le script Python d'acquisition qui ne commence effectivement que sur le prochain PPS, acquiert quelques secondes au rythme d'une mesure toutes les 200 ns par définition de la fréquence d'échantillonnage, et stocke le fichier résultant. À la fin de l'acquisition, nous demandons au système d'exploitation GNU/Linux la date de stockage du fichier par `stat -c %y`. Par ailleurs, le flux de données IQ acquis par le convertisseur analogique-numérique doit permettre de dater le PPS et donc de vérifier que le déclenchement de la mesure a été initié par un tel signal.

La synchronisation par NTP nécessite de compiler le paquet `ntp` de Buildroot en plus du système d'exploitation GNU/Linux sélectionné par `make raspberrypicm4io_64_defconfig`, et le *daemon* est lancé par `/etc/init.d/S49ntp*` au démarrage du système. Sous réserve que le routage des paquets vers le serveur NTP fonctionne, nous devrions voir dans `/tmp/messages` apparaître :

```
Jan  1 00:00:34 buildroot daemon.info ntpd[163]: DNS: dns_probe:
zeus.ens2m.fr, cast_flags:8, flags:101
Jan  1 00:00:34 buildroot daemon.info ntpd[163]: DNS: dns_check:
processing zeus.ens2m.fr, 8, 101
Jan  1 00:00:34 buildroot daemon.info ntpd[163]: DNS: dns_check:
DNS error: -3, Temporary failure in name resolution
Jan  1 00:00:34 buildroot daemon.info ntpd[163]: DNS: dns_take_
status: zeus.ens2m.fr=>temp, 3
Jan  1 00:00:42 buildroot daemon.info ntpd[163]: DNS: dns_probe:
zeus.ens2m.fr, cast_flags:8, flags:101
```

Résolution temporelle des divers systèmes de stockage de fichiers

Nous avons été très surpris la première fois que nous avons lancé cette commande de voir 9 décimales après la seconde. En effet, EXT4 propose une datation des fichiers à la nanoseconde de résolution. Ces décimales n'ont évidemment aucun sens sur un système d'exploitation multitâches généraliste et tout l'enjeu de cette étude est d'identifier le nombre de décimales pertinentes. Chez Microsoft, la question ne se pose pas puisque la date de clôture du fichier est stockée avec deux décimales derrière la seconde, ou 10 ms comme nous l'avons constaté en relisant des fichiers stockés sur un disque mobile formaté en NTFS.

On notera que ce n'est pas parce qu'un système de fichiers permet de stocker 9 décimales qu'elles le sont. Ainsi, nous avons découvert que les opérations sur les fichiers (`cp`, `mv`...) dans Busybox mettent simplement la partie fractionnaire des secondes à 0, perdant ainsi la résolution recherchée. Tel que documenté à **https://bugs.busybox.net/show_bug.cgi?id=15622**, lors de la création du fichier sur une Raspberry Pi 4 exécutant un système GNU/Linux issu de Buildroot :

```
TWSTFT:13:44# touch toto
TWSTFT:13:44# stat toto
  File: toto
  Size: 0               Blocks: 0          IO Block: 4096   regular empty file
Device: 13h/19d         Inode: 10525       Links: 1
Access: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 2023-06-06 13:44:44.804197892 +0000
Modify: 2023-06-06 13:44:44.804197892 +0000
Change: 2023-06-06 13:44:44.804197892 +0000
```

Cela fournit bien les 9 décimales, mais lors de toute manipulation du fichier :

```
TWSTFT:13:44# mv /tmp/toto .
TWSTFT:13:45# stat toto
  File: toto
  Size: 0               Blocks: 0          IO Block: 4096   regular empty file
Device: 801h/2049d      Inode: 303         Links: 1
Access: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 2023-06-06 13:44:44.000000000 +0000
Modify: 2023-06-06 13:44:44.000000000 +0000
Change: 2023-06-06 13:45:02.224085953 +0000
```

On note que les dates d'accès et de modifications sont tronquées à la partie entière de la seconde, et que c'est bien la manipulation du fichier qui induit ce problème.

D'après le commentaire dans le code source, le problème est connu et non résolu, tel que nous le constatons à la lecture de **https://github.com/brgl/busybox/blob/master/libbb/copy_file.c#L406** :

```
times[1].tv_sec = times[0].tv_sec = source_stat.st_mtime;
times[1].tv_usec = times[0].tv_usec = 0;
/* BTW, utimes sets usec-precision time - just FYI */
```

En imposant la partie fractionnaire de la date à 0... juste pour votre information.

```
Jan  1 00:00:42 buildroot daemon.info ntpd[163]: DNS: dns_check: processing
zeus.ens2m.fr, 8, 101
Jan  1 00:00:42 buildroot daemon.info ntpd[163]: DNS: Pool taking: 172.16.11.1
Jan  1 00:00:42 buildroot daemon.info ntpd[163]: DNS: dns_take_status:
zeus.ens2m.fr=>good, 8
Jul 26 12:52:58 buildroot daemon.warn ntpd[163]: CLOCK: time stepped by
1690375930.263849
Jul 26 12:52:58 buildroot daemon.warn ntpd[163]: CLOCK: time changed from
1970-01-01 to 2023-07-26
Jul 26 12:52:58 buildroot daemon.info ntpd[163]: INIT: MRU 10922 entries,
13 hash bits, 65536 bytes
```

indiquant que la synchronisation a eu lieu avec le passage de la date de 1970 à 2023. Par ailleurs, `ntpq -p` confirme l'adresse du serveur et son statut dont le `poll` qui indique tous les combien de secondes NTP met à jour l'horloge, intervalle qui croît de 64 à 1024 lorsque la stabilité augmente afin de réduire le volume de transactions sur le réseau.

La séquence de tests décrite ci-dessus s'appuie sur la chaîne de traitement Python produite par GNU Radio Companion et exécutée sur CM4 pour laquelle le support GNU Radio, Python3 et UHD ont été ajoutés dans Buildroot. Enfin, le script suivant :

```
sysctl -w net.core.wmem_max=2453333
sysctl -w net.core.rmem_max=2453333
echo "performance" > /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
while true; do
  python3 ./cm4_x310.py
  stat /tmp/data.bin >> res.txt
  python3 traite.py >> res.txt
done
```

configure l'interface Ethernet pour optimiser les échanges avec la X310, passe le processeur en mode performance à 1,5 GHz pour limiter les risques de pertes de paquets, et boucle sur une requête de 3 secondes de mesures par la X310 au rythme de 5 MS/s, la sauvegarde de l'horodatage du fichier créé lors de l'acquisition (Fig. 2), et l'analyse du fichier pour identifier la position des fronts montants de l'impulsion produite une fois par seconde 1-PPS et alimentant l'entrée de la X310. Le traitement `traite.py` se réduit à un simple seuillage :

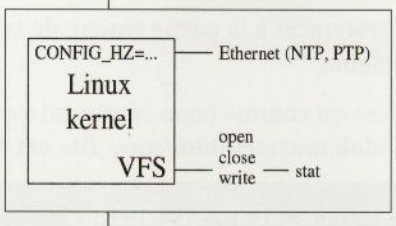


Figure 2 : Couches d'abstraction intervenant dans le problème avec les divers oscillateurs physiques cadençant les périphériques et le processeur.

```
import numpy
x=numpy.fromfile(open('/tmp/data.bin','rb'),dtype=numpy.float32)
s=numpy.nonzero(x[2:-1:4]>max(x[2:-1:4])*0.9)
print(s)
```

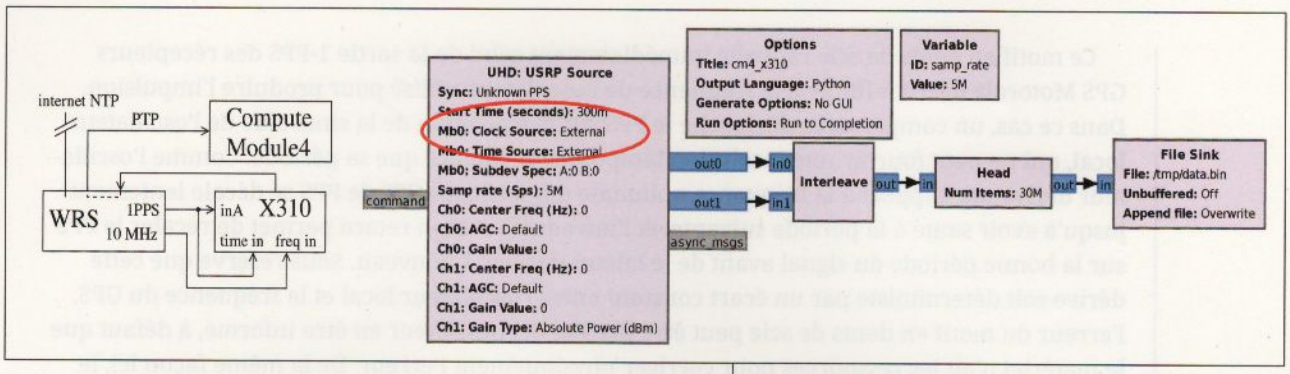


Figure 3 : Gauche : principe de la mesure, avec un récepteur de radio logicielle X310 connectée par Ethernet à une Raspberry Pi Compute Module 4, aussi connectée au travers de cette interface via un switch GbE à un serveur de temps PTP White Rabbit asservi sur maser à hydrogène, et à Internet via une passerelle. La X310 est asservie en temps (1-PPS) et en fréquence (10 MHz) sur la sortie du serveur White Rabbit, tandis que son entrée est connectée à la sortie PPS pour s'assurer que la mesure est bien déclenchée par le PPS externe. Droite : mise en œuvre dans GNU Radio Companion, dans lequel on pensera bien à activer les sources externes de temps et fréquence de la source USRP (ellipse rouge).

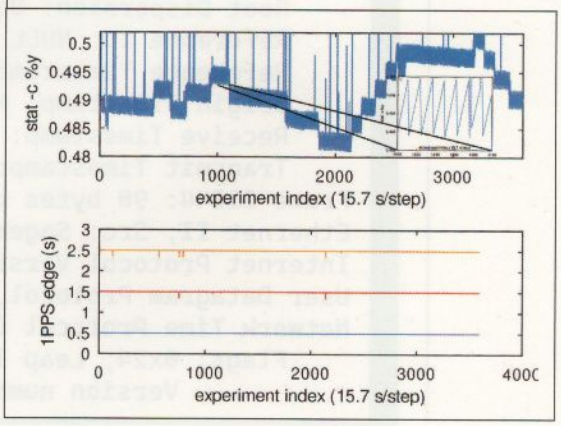
qui tient en quelques lignes de Python et permet à la mesure de se répéter toutes les 15,7 secondes qui définit la graduation selon l'axe des abscisses de toutes les courbes qui vont suivre.

Lors de la première mesure de ce type, le graphique de la Fig. 4 est produit et fournit bien des enseignements. Les 3600 points de mesure ont nécessité 15h45 de mesures au cours d'une journée. Des fluctuations de l'ordre de ± 10 ms sont observables avec la partie fractionnaire de la date de sauvegarde des fichiers – pour s'affranchir des multiples de la seconde au cours de l'avancement du temps lors de l'évolution de l'expérience – allant de 0,480 à 0,50 s. Cette courbe n'est pas continue, mais elle-même formée d'un motif en dents de scie (insert en bas à droite) dont l'excursion est de 3 ms. Ce motif a déjà été observé par le passé dans d'autres études portant sur la gestion du temps par des noyaux Unix autres que Linux, en particulier à **https://frenchfries.net/paul/dfly/nanosleep.html** qui date tout de même de 2004, sans que nous puissions identifier de référence plus récente.

L'analyse de cette courbe, qui sera confortée dans la suite par les mesures additionnelles, est la suivante :

- les fluctuations de temps au cours de la journée sont liées à la capacité de correction de NTP, pourtant reliée à un ordinateur local proche de l'observatoire de Besançon qui sert de serveur primaire ;
- la pente des dents de scie est déterminée par l'écart de fréquence entre l'oscillateur local qui cadence le processeur et sa valeur nominale de 54 MHz ;
- l'excursion du motif en dents de scie est déterminée par la granularité du compteur qui cadence le noyau Linux, paramètre configurable à la compilation et qui a été sélectionné au cours de cette expérience à 300 Hz, ou un pas de temps de 3 ms.

Figure 4 : Haut : date de création des fichiers acquis de la X310, démontrant les fluctuations long terme de fréquence que NTP peine à corriger. En insert, zoom sur une centaine d'acquisitions illustrant le motif en dents de scie induit par la granularité du compteur cadencé dans le noyau Linux pour fournir le temps système. Bas : date des fronts montants des 3 impulsions 1 PPS acquises par la X310, garantissant la cohérence des mesures et le déclenchement de l'acquisition sur le premier PPS.



Ce motif en dents de scie rappelle immédiatement celui de la sortie 1-PPS des récepteurs GPS Motorola décrit à **[8]** lié à la fréquence de l'oscillateur utilisé pour produire l'impulsion. Dans ce cas, un comparateur déclenche le PPS sur la transition de la sinusoïde de l'oscillateur local, qui ne peut fournir une résolution temporelle meilleure que sa période. Comme l'oscillateur dérive par rapport à la fréquence nominale des horloges GPS, le PPS se décale lentement jusqu'à avoir sauté à la période suivante, et l'introduction d'un retard permet de recaler le PPS sur la bonne période du signal avant de le laisser dériver à nouveau. Sous réserve que cette dérive soit déterministe par un écart constant entre l'oscillateur local et la fréquence du GPS, l'erreur du motif en dents de scie peut être prédite et l'utilisateur en être informé, à défaut que le matériel n'ait les ressources pour corriger physiquement l'erreur. De la même façon ici, le noyau Linux avec sa granularité grossière voit le temps dériver, mais ne peut rien y faire tant qu'une période n'a pas été dépassée, délai après lequel le compteur peut sauter un cran pour s'aligner à nouveau sur le temps NTP. Cette procédure est caractéristique de **synchronisation** (ajout de retard) en l'absence de **syntonisation** (écart de fréquence) quand le matériel ne permet pas d'ajuster la fréquence de l'oscillateur cadençant le processeur tel que c'est le cas sur la CM4 avec son oscillateur fixe.

Compte tenu des implications de sécurité de l'échange de temps entre services informatiques **[9]**, on peut s'interroger de la nature des informations transmises. Une observation par Wireshark informe que le client (gauche) et le serveur (droite) communiquent des informations encapsulées dans des paquets UDP (port 123) de la forme :

```
Frame 20949: 90 bytes on wire (720 bits)
Ethernet II, Src: IntelCor_12:ea:3c ...
Internet Protocol Version 4
User Datagram Protocol, Src Port: 123, Dst Port: 123
Network Time Protocol (NTP Version 4, client)
    Flags: 0x23, Leap Indicator: no warning,
           Version number: NTP Version 4, Mode: client
    [Response In: 21044]
    Peer Clock Stratum: unspecified or invalid (0)
    Peer Polling Interval: invalid (0)
    Peer Clock Precision: 4294967296.000000 seconds
    Root Delay: 0.000000 seconds
    Root Dispersion: 0.000000 seconds
    Reference ID: NULL
    Reference Timestamp: NULL
    Origin Timestamp: NULL
    Receive Timestamp: NULL
    Transmit Timestamp: Aug 24, 2093 20:29:59.522956608 UTC
Frame 21044: 90 bytes on wire (720 bits)
Ethernet II, Src: Sagemcom_35:20:16 ...
Internet Protocol Version 4
User Datagram Protocol, Src Port: 123, Dst Port: 123
Network Time Protocol (NTP Version 4, server)
    Flags: 0x24, Leap Indicator: no warning,
           Version number: NTP Version 4, Mode: server
```

```
    [Request In: 20949]
    [Delta Time: 0.079401896 seconds]
    Peer Clock Stratum: secondary reference (2)
    Peer Polling Interval: invalid (3)
    Peer Clock Precision: 0.000000 seconds
    Root Delay: 0.013824 seconds
    Root Dispersion: 0.019348 seconds
    Reference ID: 134.64.19.180
    Reference Timestamp: Aug 10, 2023 08:28:29.131806291 UTC
    Origin Timestamp: Aug 24, 2093 20:29:59.522956608 UTC
    Receive Timestamp: Aug 10, 2023 08:29:18.212364999 UTC
    Transmit Timestamp: Aug 10, 2023 08:29:18.212398515 UTC
```

Nous ne sommes malheureusement pas en 2093, mais seulement en 2023, et comme préconisé dans **[10]** le champ *Transmit Timestamp* du client est rendu aléatoire pour rendre un peu plus difficile l'injection de paquets erronés, mais la manipulation de données erronées notamment en retardant la réponse par une attaque de type *Man in the middle* y est largement décrite et référencée.

3. PTP : ASSERVISSEMENT MATÉRIEL PAR ÉCHANGE DE TEMPS

L'implémentation logicielle du jeu de ping-pong, affectée par les délais matériels et de traitement logiciel, induit donc des fluctuations de quelques millisecondes à court terme à cause de la synchronisation de l'horloge grossière du système sur l'horloge NTP (dents de scie) et présente des fluctuations long terme de quelques millisecondes.

Afin d'éliminer les fluctuations logicielles, une implémentation matérielle de ce protocole se charge de dater les paquets au niveau de l'interface physique Ethernet au lieu d'effectuer cette opération lors de l'émission des trames. Le retard logiciel est donc éliminé et seul le retard matériel subsiste. Cependant, les interfaces physiques PTP sont munies d'oscillateurs à fréquence fixe (TCXO – *Temperature Compensated Crystal Oscillator*) et seul le retard peut être corrigé, pas la fréquence. Même si cette fonctionnalité n'est pas disponible sur toute interface Ethernet, il se trouve que la CM4 est munie d'une telle interface (mais **pas la Raspberry Pi 4**) et peut donc s'asservir sur PTP au lieu de NTP, tel qu'en atteste la commande `ethtool -T eth0` qui indique :

```
# ethtool -T eth0
Time stamping parameters for eth0:
Capabilities:
        hardware-transmit     (SOF_TIMESTAMPING_TX_HARDWARE)
        hardware-receive      (SOF_TIMESTAMPING_RX_HARDWARE)
        hardware-raw-clock    (SOF_TIMESTAMPING_RAW_HARDWARE)
...
```

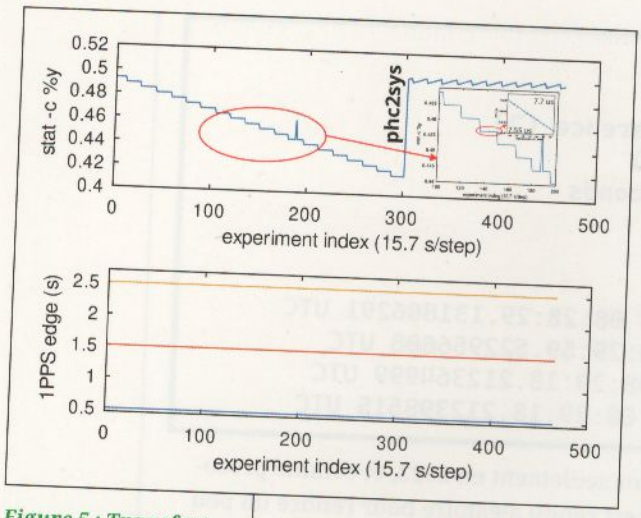


Figure 5 : Transfert du temps PTP observé par l'interface Ethernet vers le système en activant le daemon phc2sys.

Il s'avère par ailleurs que le *switch* White Rabbit que nous utilisons pour ces expériences propage, depuis sa mise à jour avec la version 6 de son *firmware*, des trames PTP, donc nous pouvons tester les performances (Fig. 5).

Cette figure illustre l'utilité de la synchronisation, avec dans un premier temps une dérive lorsque l'horloge système reste libre bien que l'interface physique soit asservie sur PTP, avant de lancer le *daemon* chargé de transmettre le temps PTP au temps système nommé `phc2sys`. Nous voyons l'impact à la mesure 300 avec le rattrapage du retard accumulé et une compensation périodique de la dérive pour nous recaler sur le temps transmis par PTP, toujours limité par la granularité du compteur qui cadence le noyau. Ainsi, nous retrouvons les motifs en

linuxptp v.4 et l'incompatibilité du serveur White Rabbit avec la CM4

Alors que nous ajoutions le paquet PTP dans Buildroot pour le compiler, il s'est avéré que nombre d'options manquaient dans la version 3.1 proposée et que la version 4 éliminerait les *patchs* en plus de fournir les fonctionnalités manquantes. Évidemment ce faisant, la synchronisation fonctionnelle entre le serveur PTP White Rabbit et la CM4 avec l'ancienne version est devenue défectueuse avec la nouvelle !

L'erreur a pu être remontée à un patch majeur de `linuxptp` sous le *hash 2a2532d66121d0060b042c5bd6020a62153f1e0a* qui implémente la version IEEE 1588-2019 de PTP. Bien que le dysfonctionnement soit documenté dans les commentaires à ce patch sur **https://github.com/richardcochran/linuxptp/commit/2a2532d66121d0060b042c5bd6020a62153f1e0a** et que le correctif puisse y être identifié par « *The reason [is] that this particular [GrandMasters] do not reply to Delay_Req messages with minor version set to 1. Below find a fragment of the Delay_Req message from wireshark dump, which is not being replied to. As soon as I change the minorVersionPTP to 0, the appropriate Delay_Resp is being sent.* », sans que nous comprenions la cause du problème que les auteurs du logiciel attribuent sur la liste de diffusion au matériel, le fait est que manuellement modifier `PTP_MINOR_VERSION` de 1 à 0 dans `msg.h` permet de retrouver le comportement fonctionnel d'origine. L'absence de palliatif logiciel à un problème connu, même si d'origine matérielle, semble aussi incompréhensible qu'improductif et nécessite pour le moment la modification manuelle pour les utilisateurs de CM4 au moins.

dents de scie que nous avions observés avec NTP, mais avons cette fois éliminé les fluctuations long terme avec une courbe fluctuant peu au cours du temps autour du motif en dents de scie. Le bon fonctionnement de PTP se valide en sondant le statut de l'interface physique par la commande *PTP Management Client* `pmc` qui prend en argument la nature des informations requises, en limitant aux interfaces locales seules (`-b 0`), de la forme :

```
pmc -u -b 0 'GET CURRENT_DATA_SET'
pmc -u -b 0 'GET TIME_STATUS_NP'
pmc -u -b 0 'GET TIME_PROPERTIES_DATA_SET'
```

tandis que le transfert de temps de l'interface physique vers le temps système par `phc2sys` est validé lorsque `date` ne fournit plus le premier janvier 1970, mais la date actuelle fournie par le *Grandmaster* PTP supposé être à l'heure (par NTP ou PTP, par exemple). Nous avons constaté que nous devions relancer le *daemon* `phc2sys` par `/etc/init.d/S66phc2sys restart` une fois l'asservissement PTP par `ptp4l` fonctionnel pour effectivement engager le transfert de temps de PTP vers le système. Nous prendrons soin par ailleurs de désactiver NTP pour ne pas utiliser par erreur le mauvais mode de synchronisation, tout cela étant automatisé au lancement de la CM4 par :

```
/etc/init.d/S49ntp stop
/etc/init.d/S65ptp4l stop
/etc/init.d/S49ntpd stop
/etc/init.d/S65ptpd2 stop
echo "REMEMBER TO RESTART  /etc/init.d/S66phc2sys restart"
./testptp -d /dev/ptp0 -L0,2
./testptp -d /dev/ptp0 -p 1000000000
ptp4l -s -m -i eth0 -E -2 --tx_timestamp_timeout 200
```

où les arguments de `ptp4l` indiquent que nous sommes esclaves (`-s` pour *SlaveOnly*), en affichant le statut sur la sortie standard (`-m`) et en nous appuyant sur la couche Ethernet (-2) au lieu de la couche IP. Par ailleurs, le programme `testptp` disponible dans les sources du noyau Linux dans `tools/testing/selftests/ptp` (dans Buildroot dans les sources du noyau qui se trouvent dans `output/build/linux-custom`) fournit bien des fonctionnalités, par exemple ici la production d'un signal 1-PPS visualisable sur oscilloscope sur la broche 9 de J2 de la Compute Module 4 IO Board **[5]** placée en sortie. Nous verrons plus tard que cette même broche en entrée permettra de s'asservir sur le signal de cadencement issu d'un récepteur GPS.

Nous constatons donc que PTP permet un asservissement fin de l'horloge système qui sert à dater les fichiers, mais qu'un motif en dents de scie subsiste. Sachant que ce motif en dents de scie est habituellement associé à un compteur de granularité médiocre, nous étudions quel est l'impact de la fréquence du *timer* du noyau puisque les fluctuations long terme ont disparu, et seul le motif en dents de scie subsiste, d'amplitude 4 ms... qui est justement l'inverse de la configuration du *timer* dans le noyau Linux, tel que nous le vérifions par exemple sous Debian/GNU Linux par :

```
$ grep _HZ /boot/config-6.1.0-3-amd64
CONFIG_NO_HZ_COMMON=y
# CONFIG_HZ_100 is not set
CONFIG_HZ_250=y
# CONFIG_HZ_300 is not set
# CONFIG_HZ_1000 is not set
CONFIG_HZ=250
```

qui indique que par défaut, le noyau est cadencé au rythme de 250 observations par seconde. L'aide du noyau Linux sur ces paramètres indique que le choix de ce cadencement est un compromis entre le nombre d'interruptions gérées par le système égal au nombre de cœurs de processeur multiplié par cette fréquence, et que pour un système répondant rapidement aux sollicitations d'un utilisateur, il est judicieux de régler à `CONFIG_HZ_1000`... alors que Debian maintient la valeur par défaut de 250 Hz.

En étudiant dans Buildroot les options du noyau Linux par `make linux-menuconfig`, nous constatons que le *timer* peut prendre des valeurs de 100, 250, 300 et 1000 Hz. Une fois la nouvelle fréquence sélectionnée, `make linux`

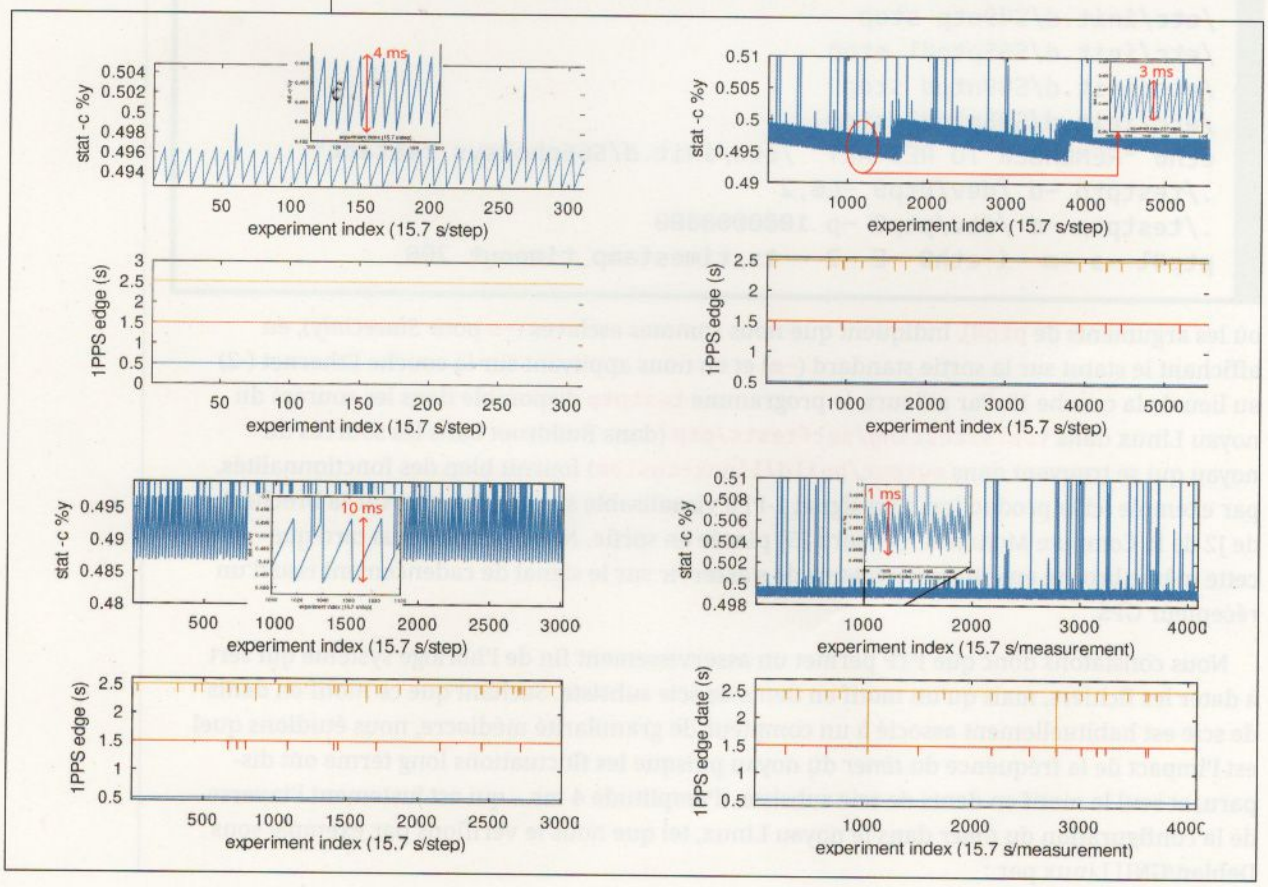


Figure 6 : Quatre mesures successives de la datation des fichiers acquis par la X310 en fonction de la configuration du noyau en sélectionnant un timer à 250 Hz, la valeur par défaut dans le noyau Linux (haut gauche), 300 Hz (haut droite), 100 Hz (bas gauche) et 1 kHz (bas droite). Pour chaque courbe, en bas la date des fronts montants des 3 impulsions 1 PPS acquises par la X310, garantissant la cohérence des mesures et le déclenchement de l'acquisition sur le premier PPS.

recompile le noyau et `make` place `output/images/Image` de l'hôte que nous pourrons copier dans la première partition de la carte SD de la cible pour *rebooter* sur le nouveau noyau. Nous observons les résultats en Fig. 6 : les dents de scie s'atténuent avec une fréquence croissante du *timer* du noyau, mais il semblerait que de temps en temps le noyau oublie sa mise à jour et laisse l'horloge dériver excessivement. Cet effet est peut-être induit par la puissance de calcul modeste de la CM4, pourtant configurée en gouverneur « performance » pour forcer les 4 cœurs de calcul à 1,5 GHz.

4. WHITE RABBIT : ASSERVISSEMENT MATÉRIEL PAR ÉCHANGE DE TEMPS ET FRÉQUENCE

Nous avons explicité les limitations de synchronisation des interlocuteurs par l'action exclusivement sur le retard en laissant les oscillateurs des interlocuteurs libres de dériver. Ce problème est résolu par l'extension de PTP développée par le CERN nommée White Rabbit qui implémente une solution haute performance en permettant la correction de l'horloge locale cadençant la base de temps. Ce faisant, l'oscillateur contrôlé en tension (VCO) peut se caler sur son maître et ainsi annuler sa dérive. Les performances sont drastiquement améliorées, avec des fluctuations sur la *sortie* 1-PPS de White Rabbit fluctuant de quelques dizaines de picosecondes contre quelques nanosecondes pour PTP tel que nous l'avons démontré à **https://forums.ohwr.org/t/synchronizing-wr-master-and-a-non-wr-node-using-ptp/848417/31**. Visuellement, cela se traduit sur un oscilloscope par des fronts d'impulsions de synchronisation qui ne varient non plus de quelques dizaines de nanosecondes, mais seulement de quelques dizaines de picosecondes, à peine perceptible sur une échelle de temps commune (Fig. 7).

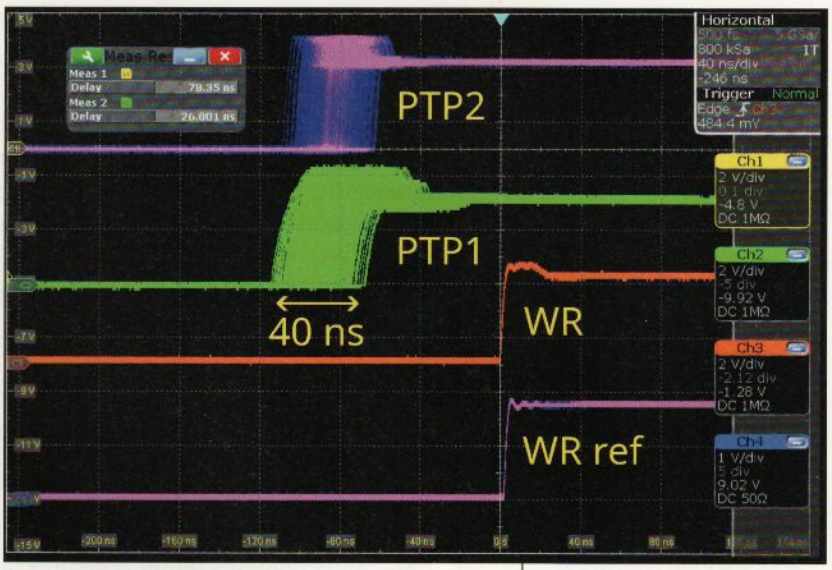


Figure 7 : Capture d'écran des fronts montants des signaux 1-PPS déclenchés sur un signal issu d'un switch White Rabbit (WR ref) de référence (violet en bas). Un second switch White Rabbit (WR) séparé par environ 2 km de fibres optiques du premier présente une gigue d'une soixantaine de ps (courbe orange). Par contre, deux CM4 asservies par PTP (PTP1 et PRP2) au travers d'un lien RJ45 sur câble électrique (et non fibre optique) à ce même switch White Rabbit présentent toutes deux une gigue de quelques dizaines de picosecondes qui se retrouve aussi entre les deux CM4.

Malheureusement, White Rabbit n'est pas accessible au commun des mortels : un *switch* White Rabbit coûte 2780 euros et les cartes PCI qui transmettent le temps au noyau Linux ont bien du mal à être maintenues fonctionnelles avec les noyaux actuels **[11]**. Le CERN publiant tous les codes, il est envisageable de porter White Rabbit à tout FPGA muni d'une interface optique (SFP), condition nécessaire à garantir la maîtrise des délais, mais la

multiplicité des dépôts interdépendants, la multiplicité des branches incompatibles au sein de chaque dépôt, et la multiplicité des couches d'abstraction (*gateware*, *softcore*, modules noyau Linux, applicatifs) rend la tâche ardue sans un lien étroit avec les développeurs White Rabbit, dont la tâche première est de cadencer un accélérateur de particules.

4.1 Analyse de l'impact des fréquences d'oscillateurs sur les performances de datation

Ayant commencé cette étude sur la CM4, nous pouvons tenter d'ajouter la syntonisation – asservissement de la fréquence – à la synchronisation déjà fournie par PTP. L'étude du circuit imprimé indique que seuls deux oscillateurs sont présents, et leur emplacement laisse présager un usage facile à identifier avec un résonateur 25 MHz placé à proximité de l'interface Ethernet de référence BCM5421 et un résonateur 54 MHz placé proche du processeur BCM5711 (Fig. 8). Il sera donc aisé d'analyser l'impact de chaque oscillateur formé du résonateur assemblé en montage Pierce **[12, section 2.1]** autour d'une porte inverseuse dans le circuit numérique et les deux condensateurs de pieds de quelques pF pour respecter la condition de phase de Barkhausen, voire de le modifier pour permettre l'ajustement dynamique de la fréquence. Cependant, avant d'attaquer les modifications du circuit, nous désirons nous convaincre de la pertinence de la démarche.

Nous commençons ainsi dans un premier temps par une expérience de laboratoire consistant à remplacer les deux résonateurs – 25 et 54 MHz – par deux synthétiseurs programmables émettant une puissance de +4 dBm à leurs fréquences respectives (Fig. 9, droite).

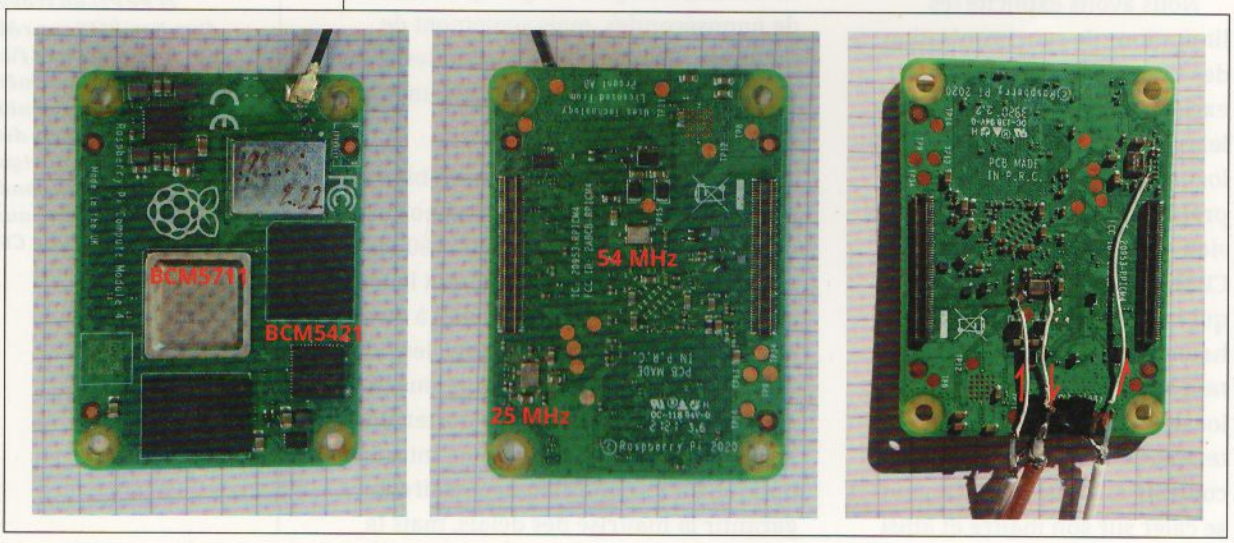
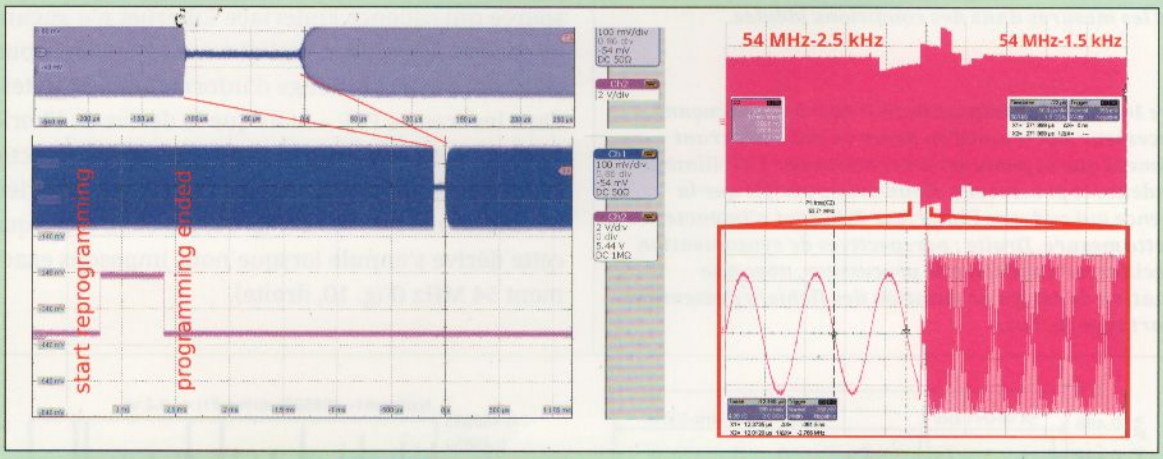


Figure 8 : La CM4 (gauche) est une plateforme idéale pour l'analyse du transfert de temps entre ordinateurs, car elle n'est munie que de deux sources de cadencement (milieu), un résonateur à quartz à 54 MHz qui cadence le processeur BCM5711 et un résonateur à quartz à 25 MHz pour cadencer l'interface physique Ethernet BCM5421. Droite : quelques points de test pour observer ou imposer la fréquence des oscillateurs. Notez que sur l'oscillateur à 54 MHz, nous avons constaté que l'injection du signal se fait efficacement en bas à gauche au travers d'une capacité et l'observation s'effectue en haut à droite.

Cadencement d'un système numérique sur synthétiseur de fréquence

L'analyse de l'implication de la fréquence de cadencement du processeur ou de l'interface physique Ethernet est grandement facilitée par l'utilisation de synthétiseurs de signaux radiofréquences commerciaux dont l'interface utilisateur permet de facilement définir les paramètres que sont la puissance ou la fréquence. Il est donc tentant de se servir de ces instruments pour prototyper un asservissement de fréquence et valider les performances d'ajustement de la fréquence en fonction des paramètres lus par les *deamons* PTP et associés. Sachant qu'un synthétiseur Rohde & Schwarz SMA100A coupe son émission radiofréquence au cours de la reprogrammation de la fréquence (figure ci-dessous, gauche) pendant une centaine de microsecondes – rendant le système numérique amnésique pendant cette durée – nous avons tenté cette fois un synthétiseur AIM-TTi TGR2053. Le résultat n'est pas bien meilleur, avec une Raspberry Pi Compute Module 4 qui cesse de fonctionner chaque fois que la fréquence est reprogrammée. Ce comportement est peu surprenant quand on arrive à déclencher un oscilloscope sur le transitoire : alors que la fréquence est reprogrammée de 54 MHz-2,5 kHz à 54 MHz-1,5 kHz, une analyse fine permet de constater des transitoires à quelques MHz suivis de périodes à 200 MHz avant de se stabiliser sur la valeur finale. Le processeur est incapable de s'accommoder de telles fluctuations de fréquences pendant plusieurs dizaines de microsecondes, rendant son comportement aléatoire pendant ces transitions de fréquences. On ne saura prendre trop garde à ces instruments dont l'utilisation est d'apparence aisée, mais dont le comportement dans la pratique peut réserver des surprises : dans la même veine, un synthétiseur numérique affichera sur son écran 20 MHz avec 8 décimales à 0, mais en pratique produit un signal décalé de quelques microhertz, compte tenu des arrondis numériques.



Gauche : transitoire en fréquence du SMA100A. Droite : transitoire du TGR2053.

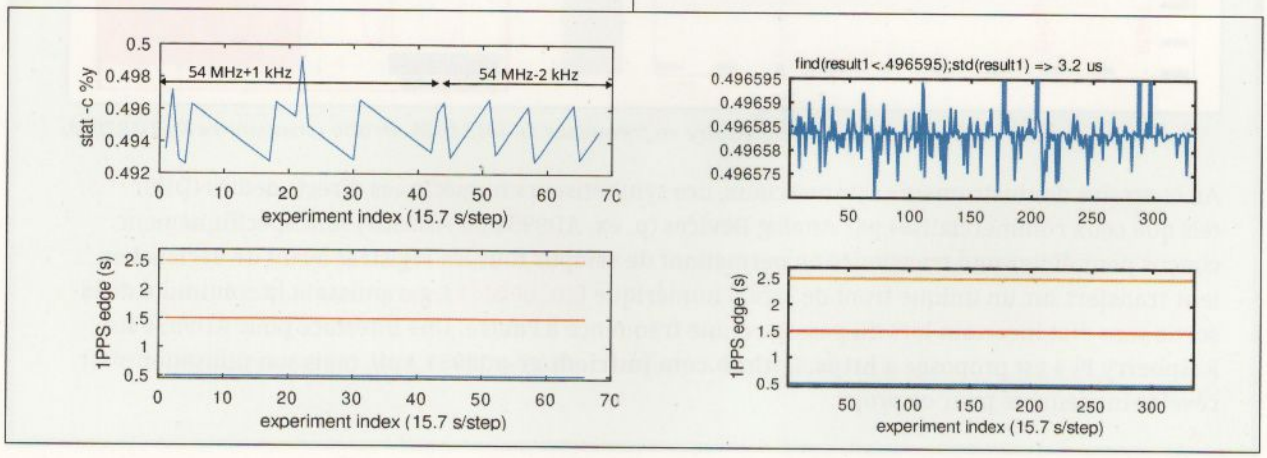
Au contraire des instruments commerciaux, des synthétiseurs numériques directs dédiés (DDS) tels que ceux commercialisés par Analog Devices (p. ex. AD9954 ou AD9832) sont spécifiquement conçus pour éviter tout transitoire en permettant de remplir tous les registres avant de déclencher leur transfert sur un unique front de signal numérique (`IO_UPDATE`), garantissant la continuité de la sortie sans état incertain lors du passage d'une fréquence à l'autre. Une interface pour AD9954 sur Raspberry Pi 4 est proposée à **https://github.com/jmfriedt/gr-ad9954_rpi/**, mais son utilisation s'est révélée inadéquate pour ce projet.

Figure 9 : Gauche : mesure de la fréquence de l'oscillateur cadençant le processeur, de fréquence nominale de 54 MHz, mais en pratique décalée de -1,3 kHz environ. Droite : les fréquences cadençant le processeur et l'interface Ethernet sont imposées par des synthétiseurs contrôlés par la même source de fréquence d'excellente qualité qui cadence le compteur de fréquence et le grand maître PTP, garantissant la cohérence de toutes les mesures dans des conditions idéales.

Figure 10 : Gauche : impact de la fréquence cadençant le processeur sur le motif en dents de scie, montrant clairement que la pente est déterminée par l'oscillateur qui cadence le processeur. Nous avons vérifié que la fréquence qui cadence l'interface Ethernet n'impacte pas cette mesure. Droite : perspectives de syntonisation de l'oscillateur cadençant le processeur, avec une fluctuation du temps de datation des fichiers présentant un écart type de 3 µs.

Nous constatons dans ce cas que l'observation par un compteur de fréquence (HP53131A) remplace la fréquence naturelle de l'oscillateur (gauche), trop basse, par la fréquence forcée.

Dans ces conditions, nous observons que la source qui cadence l'interface Ethernet n'a aucun impact sur le résultat – tel que nous pouvons nous y attendre avec l'échange d'informations de dates dans les trames PTP – mais que la dérive de l'horloge système peut être ajustée pour changer de direction selon que nous imposions une fréquence supérieure ou inférieure à 54 MHz (Fig. 10, gauche), voire que cette dérive s'annule lorsque nous imposons exactement 54 MHz (Fig. 10, droite).



Dans cette dernière configuration, nous observons un écart type sur la date de sauvegarde des fichiers acquis de la X310 – après avoir éliminé les points évidemment erronés – de l'ordre de 3 µs que nous attribuons aux fluctuations de temps d'exécution des appels système par le noyau multitâche. Le gain est donc de l'ordre de 1000 par rapport à un *timer* noyau configuré pour être cadencé à 300 Hz (dents de scie de 3 ms d'amplitude). La syntonisation, puisqu'il s'agit ici de l'effet atteint, présente donc un gain significatif que nous désirons automatiser.

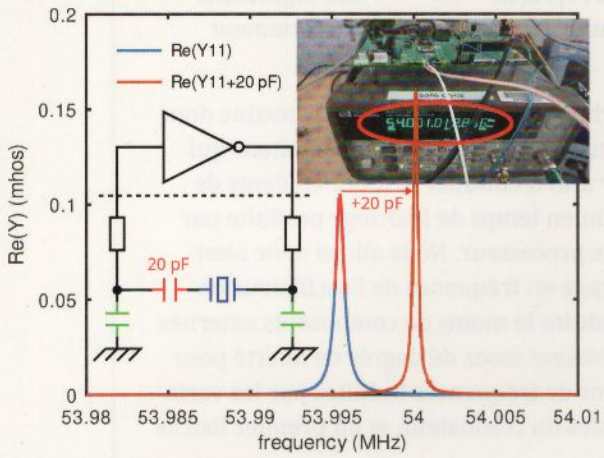
La liste de diffusion *linuxptp* informe par ailleurs d'une option qui pourrait dégrader la vitesse de réponse du noyau à **https://linuxptp-users.narkive.com/479lcYvn/status-file-of-achieved-time-synchronization** en mentionnant que « *you should know that the Linux kernel option CONFIG_NO_HZ_IDLE is harmful to your use case. I recommend to either disable this at compile time or to add «nohz=off» to your kernel command line. Only by using the nohz=off option in the kernel command line, the delay in phc2sys dropped from several microseconds to a hundred nanoseconds.* » et en effet, l'option `CONFIG_NO_HZ_IDLE` est active par défaut, mais l'ajout de `nohz=off` aux arguments de démarrage du noyau n'a pas modifié le comportement observé.

Cette dernière courbe, Fig. 10 (droite), nous convainc donc de la nécessité de corriger la fréquence de l'oscillateur qui cadence le processeur afin d'éliminer le motif en dents de scie de l'asservissement en temps de l'horloge produite par le noyau exécuté sur le processeur. Nous allons donc aborder le problème du tirage en fréquence de l'oscillateur en nous efforçant d'introduire le moins de composants externes possible, tout en fournissant assez de degrés de liberté pour corriger les fluctuations de fréquences induites par les variations environnementales du résonateur, et en premier lieu sa température.

4.2 Mise en pratique : correction de la fréquence d'un oscillateur à quartz

Il existe de nombreuses façons de corriger la fréquence d'un oscillateur à quartz, la bande passante du résonateur résultant de conditions mécaniques liées à la géométrie du substrat de quartz et de la célérité de l'onde élastique qui s'y propage. Comme la célérité de l'onde élastique est liée au ratio de la constante mécanique (l'équivalent en matériau isotrope du module de Young) à la densité, tout changement de ces paramètres se traduit par une variation de la fréquence. La documentation technique indique que la tolérance de fabrication de ces résonateurs faible coût est ± 50 ppm, signifiant une variation relative de la fréquence à la valeur nominale de $50 \cdot 10^{-6}$. À 54 MHz, le résonateur peut donc présenter une fréquence effective de $54 \cdot 10^{6} \pm 2700$ Hz, écart à la valeur nominale dont une partie est fixe (géométrie) et une partie variable, notamment en fonction de la température. L'asservissement vise donc à corriger ces deux effets pour amener la fréquence de l'oscillateur à sa valeur nominale.

Les conditions d'oscillations de la boucle contenant l'amplificateur et le résonateur sont qualifiées de Barkhausen, avec la première condition grossière qui dicte que le gain de l'amplificateur compense les pertes, et la seconde fine qui affirme que la somme des phases dans la boucle est 0 [2π] afin que l'onde qui entre dans l'oscillateur se superpose à celle qui sort de l'amplificateur et que la puissance s'accumule de façon cohérente.



ELM8

SMD Crystal
3.2mm x 2.5mm

AEL CRYSTALS
Powered By ABRACON

Features

- SMD Quartz Crystal Resonator
- Tolerance: ±10ppm Available
- Operating Temperature -10°C to +60°C to -40°C to +125°C

STANDARD SPECIFICATIONS	
PARAMETERS	**MAX (Unless otherwise noted)**
Frequency Range	8.000 to 125.000 MHz
Operation Mode 8MHz -54MHz 64MHz - 125MHz (Contact Abracon for available frequencies)	 Fundamental 3rd Overtone (Standard Frequencies: 64, 66, 66.6666, 75, 80, 98.304, 100, 125MHz)
Operating Temperature	-10°C to +60°C, see options
Storage Temperature	-55°C to +125°C
Frequency Tolerance @ 25°C	± 50PPM, see options
Frequency Stability over the Operating Temperature (ref. to +25°C)	± 50PPM, see options
Equivalent Series Resistance (ESR)	See Table below
Shunt Capacitance (C0)	3.0 pF
Load Capacitance (CL)	18 pF, see options
Drive Level	10 µW Typ 100 µW
Aging (@ 25°C, First Year)	±2ppm

Figure 11 : Gauche : caractérisation à l'analyseur de réseau vectoriel du résonateur AEL sur son mode fondamental à 54 MHz (bleu) et largeur à mi-hauteur permettant d'identifier son facteur de qualité, et simulation d'une capacité de 20 pF en série pour amener la résonance à sa valeur nominale de 54 MHz. En insert, le circuit d'oscillation de Pierce, avec en haut la porte inverseuse intégrée dans le processeur de la CM4 (au-dessus de la ligne pointillée) qui sert d'amplificateur du signal filtré par le résonateur. La condition de phase d'oscillation de Barkhausen est vérifiée grâce aux condensateurs de pieds en vert, tandis que le condensateur rouge en série du résonateur peut servir à augmenter la fréquence de l'oscillateur. Deux résistances de 10 Ω sont incluses entre le processeur de la CM4 et le résonateur pour des raisons que nous ne savons pas identifier. Droite : documentation technique du résonateur indiquant la tolérance de ± 50 MHz par défaut, ou 2700 Hz dans ce cas.

Ainsi, l'approche classique d'ajustement de fréquence consiste à jouer sur la condition de Barkhausen de phase en ajustant les condensateurs de pieds dans le montage d'oscillateur de Pierce équipant tout système numérique, par exemple en complétant un des condensateurs de pieds par une varicap, un condensateur variable commandable en tension. Cependant, nous observons que la fréquence naturelle du circuit (Fig. 9) fourni sur le modèle de CM4 que nous exploitons est déjà **en deçà** de la fréquence nominale de 1,5 kHz (Fig. 11), soit une valeur dans la tolérance de fabrication, mais que nous ne pourrons rattraper par une capacité de pieds additionnelle qui ne peut faire que descendre encore plus la fréquence. Dans ces conditions, il nous faut amener la fréquence de l'oscillateur **au-dessus** de la valeur nominale de 54 MHz pour que cette approche soit effective. Diverses références **[12, section 1.3]** indiquent que la seule façon *d'augmenter* la fréquence de l'oscillateur est d'amener une capacité C en *série* avec le résonateur. En effet, en insérant une vingtaine de pF entre le résonateur et un des condensateurs de pieds, nous constatons que la fréquence croît pour dépasser les 54 MHz nominaux, nous permettant d'envisager le contrôle par une capacité variable de pieds programmable en tension qu'est une varicap (Fig. 12). Nous proposons en fin de ce document une courte annexe qui développe la simulation du résonateur à onde de volume sous `ngspice` intégré dans KiCad (voir annexe A) et sous Qucs (voir annexe B).

Finalement, nous nous interrogeons sur la capacité de tirage du résonateur par rapport à la variation de fréquence du résonateur à quartz

avec son environnement, et en particulier la température. En effet, les paramètres mécaniques de propagation de l'onde élastique vont varier avec la température, induisant une accélération ou un ralentissement observé comme variation de la fréquence à géométrie fixe **[13]**. Expérimentalement, il est facile de chauffer (du courant dans une résistance ou un transistor) alors que refroidir est compliqué (un module Peltier est encombrant et nécessite un radiateur sur sa face chaude) : nous constatons que l'orientation cristalline sélectionnée par le fabricant du résonateur induit une baisse de fréquence quand la température augmente (Fig. 13), impliquant qu'il faut garder un peu de marge si le circuit chauffe lors de son fonctionnement. Par ailleurs, cette mesure explique pourquoi on ne coupe jamais un oscillateur, son temps de stabilisation pouvant se compter en heures avant que la température ne soit homogène dans le très mauvais conducteur thermique qu'est le quartz.

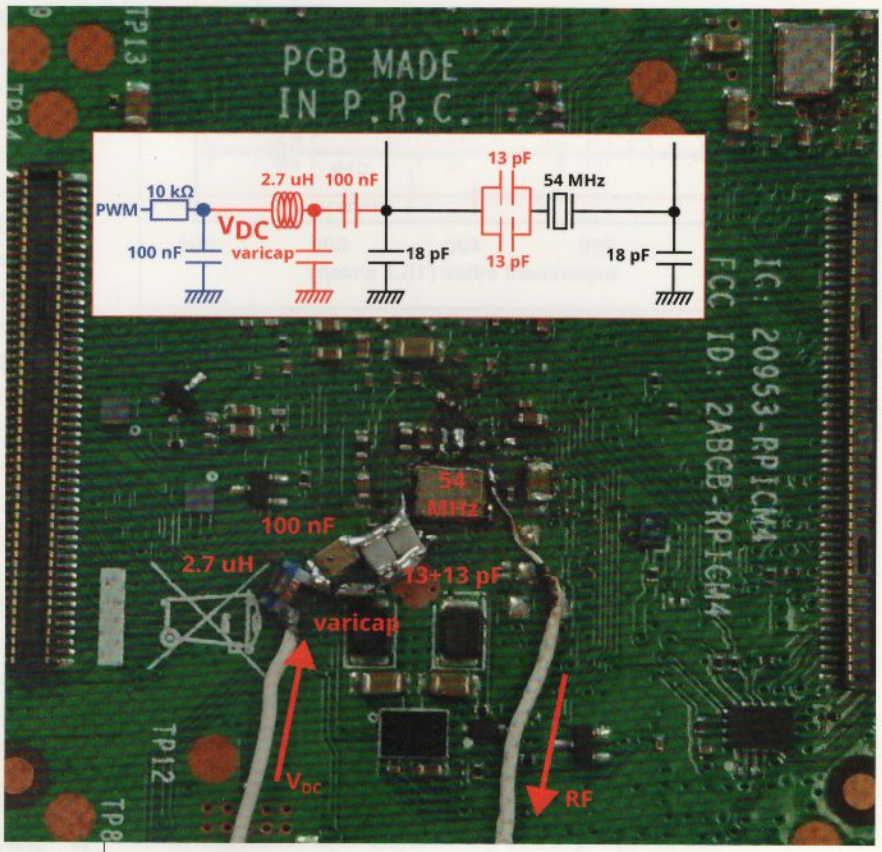


Figure 12 : Ajout de la varicap (rouge) entre le résonateur 54 MHz et la capacité de pieds de valeur nominale 18 pF (noir). La varicap est polarisée par une tension DC issue de la PWM filtrée (bleu), mais afin de protéger la porte inverseuse, une capacité de grande valeur (100 nF ou une impédance de 0,03 Ω à 54 MHz) coupe la composante DC vers l'oscillateur tandis qu'une inductance de grande valeur (2,7 µH ou une impédance de 1 kΩ à 54 MHz) bloque la fuite du signal radiofréquence (RF) vers la tension de commande qui sinon empêcherait l'oscillation de se maintenir. La fréquence est observée sur la broche opposée à celle servant au tirage du résonateur.

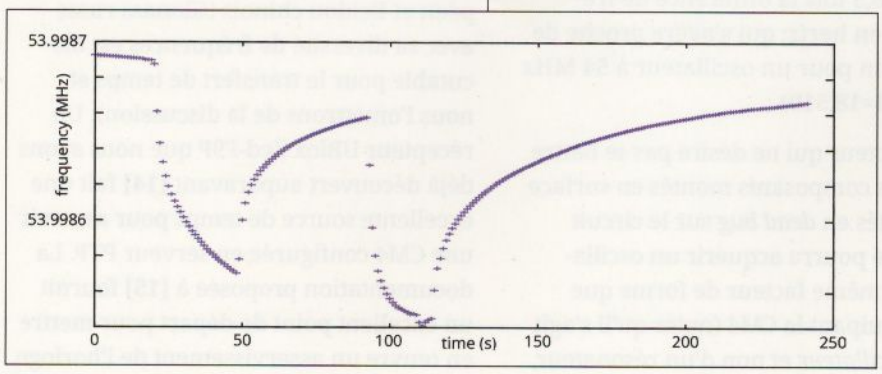


Figure 13 : Impact de l'échauffement du résonateur à quartz à 54 MHz sur sa fréquence en plaçant à deux reprises (dates 15 et 90) un fer à souder sur la face supérieure de la CM4 au niveau de la feuille de la framboise la plus proche du processeur : la chaleur diffuse à travers le circuit imprimé et échauffe le substrat de quartz, induisant une variation de fréquence.

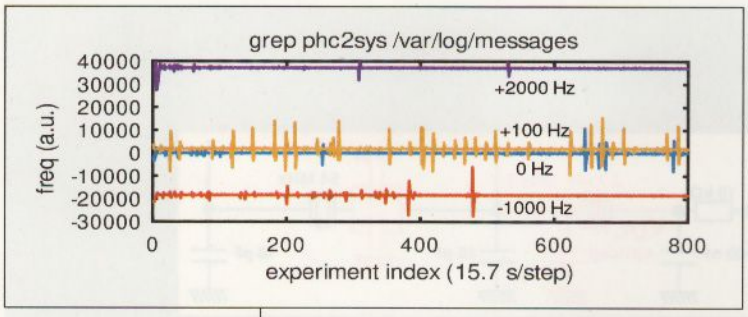


Figure 14 : Information de fréquence fournie dans le fichier de messages dans les logs en fonction de l'écart de fréquence de cadencement du processeur à sa valeur nominale de 54 MHz.

Un dernier problème se pose : nous sommes capables d'observer la dérive du temps système par rapport au temps de référence PTP par la sauvegarde de fichiers contenant les informations acquises par la X310, mais comment établir l'écart de fréquence PTP-système en l'absence d'un tel montage de test ? La solution ne peut pas venir de `ptp4l` qui se charge exclusivement d'asservir l'horloge PTP de l'interface physique et ne s'intéresse pas au système : son champ `freq` n'est pas affecté par la fréquence qui cadence le processeur. Au contraire, `phc2sys` qui fait le lien entre PTP et le noyau contient l'information pertinente dans son champ `freq` tel que nous le vérifions sur le montage de test dans la Fig. 14. Nous avons ainsi volontairement décalé la fréquence de cadencement du processeur de sa valeur nominale et constatons que `phc2sys` indique un écart de fréquence égal à 18,5 fois la différence de fréquence en hertz, qui s'avère proche de 1000 ppm pour un oscillateur à 54 MHz (1000/54=18,519).

Le lecteur qui ne désire pas se battre avec des composants montés en surface assemblés en *dead bug* sur le circuit imprimé pourra acquérir un oscillateur du même facteur de forme que celui équipant la CM4 (noter qu'il s'agit d'un *oscillateur* et non d'un résonateur, cette fois), mais commandable en tension dans une gamme de ± 100 ppm qui doit permettre de compenser la tolérance de fabrication de ± 50 ppm, sous la forme de la référence ECS-2532VXO-540B-2.8 disponible pour 7,60 euros chez Digikey sous la nomenclature XC1488CT-ND. Le remplacement du résonateur par un oscillateur a son importance, car nous allons injecter une fréquence forcée au processeur de la CM4 au lieu de la laisser produire sa propre oscillation par le montage de Pierce. Il faut donc distinguer entrée et sortie de la porte inverseuse pour injecter le signal du bon côté (l'entrée de la porte). Empiriquement, nous avons constaté que c'est la broche la plus proche du symbole de la poubelle sur le circuit imprimé qui permet une telle injection de signal, l'autre (sortie de la porte logique) étant plus propice à l'observation de la fréquence par un compteur.

5. SERVEUR DE TEMPS PTP SUR CM4 CONTRÔLÉ PAR GNSS (GPS)

Sans nécessairement en être conscient, le commun des mortels a bien accès à une centaine d'horloges atomiques : il s'agit des constellations de navigation par satellite GPS américain, Galileo européen et Beidou chinois (Glonass russe avec sa diversité de fréquences est discutable pour le transfert de temps et nous l'omettrons de la discussion). Un récepteur UBlox Zed-F9P que nous avons déjà découvert auparavant **[14]** fait une excellente source de temps pour asservir une CM4 configurée en serveur PTP. La documentation proposée à **[15]** fournit un excellent point de départ pour mettre en œuvre un asservissement de l'horloge

du noyau Linux sur l'information temporelle fournie par ce récepteur GPS sous la forme de son impulsion 1-PPS. En effet, la broche qui nous a servis auparavant à qualifier l'asservissement de l'interface physique sur PTP en la configurant comme une sortie 1-PPS peut aussi être configurée en entrée pour fournir le signal de datation de l'interface physique.

Cependant, il faut pouvoir non seulement recevoir une information de datation sous forme du signal 1-PPS, mais aussi lire les trames NMEA transmises par le récepteur GPS et contenant la date et l'heure. Dans le circuit MikroElektronika MIKROE-4456, le port UART compatible RS232 émet les trames NMEA au débit de 38400 bauds, cause de notre mise à jour de la version de `linuxptp` à la version 4 avec l'ajout de la configuration du *baudrate* dans le fichier de configuration. La latence induite par ce débit de communication relativement lent n'a pas d'importance sur la datation qui est définie uniquement sur le front montant du 1-PPS alors que le message transmis par RS232 n'a vocation qu'à identifier la date et l'heure associées à ce signal. Cependant, nous désirons conserver la console UART de la CM4 nommée `ttyAMA0` pour déverminer

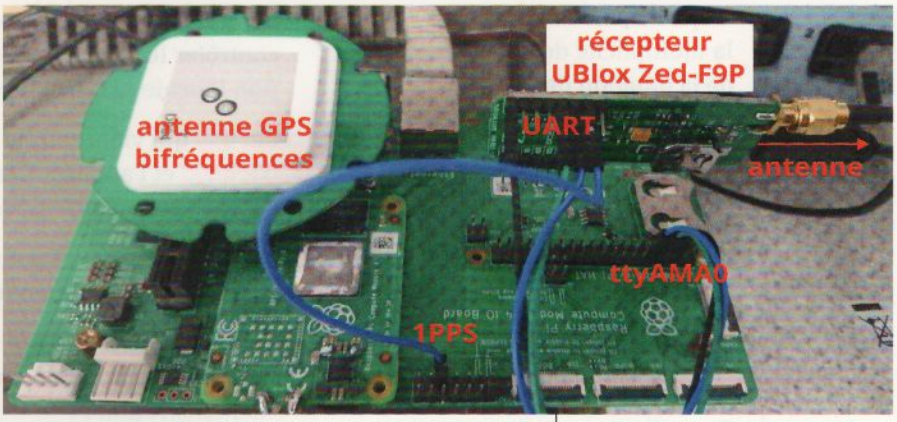


Figure 15 : Montage final, un récepteur UBlox Zed-F9P sur carte MIKROE-4456 est connecté par UART pour transmettre date et heure à la CM4 tandis que l'impulsion horaire est transmise au PHC dont la broche est cette fois configurée en entrée.

les cas de dysfonctionnement de l'interface Ethernet, et utilisons donc un convertisseur série/USB qui se connecte sur le port USB-A de la carte Compute Module 4 IO Board. Il s'avère que nous devons manuellement activer le support du port USB de la CM4, sans quoi le régulateur de tension qui alimentera les périphériques ne sera pas activé :

```
CHEMIN=/lib/modules/5.15.61-v8/kernel/drivers/usb/
insmod ${CHEMIN}/roles/roles.ko
insmod ${CHEMIN}/dwc2/dwc2.ko
insmod ${CHEMIN}/serial/usbserial.ko
insmod ${CHEMIN}/serial/ch341.ko
```

Une fois l'interface de communication créée dans `/dev/ttyUSB0`, nous pouvons vérifier que les trames NMEA circulent et sont lisibles depuis la CM4 :

```
stty -F /dev/ttyUSB0 38400 && cat < /dev/ttyUSB0
```

et nous attendons de voir des messages de position et de datation remplis, notamment GNGGA de la forme `$GNGGA,050252.00,...`. Il faut absolument que la solution de position soit fournie, sans quoi l'impulsion 1-PPS ne peut être produite.

L'asservissement d'un ordinateur sur GPS n'est pas complètement trivial, malgré les efforts de documentation à **[15]** sans lesquels ce résultat n'aurait pu être obtenu.

La principale difficulté est la multiplicité des étapes, de la datation des trames de l'interface physique par le signal 1-PPS produit par le récepteur GPS, à la propagation de ce signal vers le noyau Linux puis vers l'horloge système. Pour le moment, nous restons sur une procédure manuelle qui mériterait d'être automatisée. Travaillant sur CM4 exécutant un système GNU Linux issu de Buildroot, nous ne nous appuierons pas sur `systemd` tel que préconisé dans la documentation, mais nous efforcerons de comprendre chaque étape lancée manuellement. Les quatre outils impliqués, tous disponibles dans Buildroot, seront :

- `ts2phc` utilisé pour synchroniser les horloges matérielles supportant PTP (*PTP Hardware Clocks* ou PHC) à un signal externe tel que l'impulsion 1-PPS venant d'un récepteur GPS proposant cette fonctionnalité (le Zed-F9P la fournit, bien entendu) ;
- `phc2sys` que nous avons déjà rencontré pour synchroniser le temps système du noyau avec le temps des PHC, mais cette fois en plaçant l'information de datation dans une mémoire partagée (SHM) avec un serveur NTP et en particulier `chrony` que nous mentionnons ci-dessous ;
- `ptp4l` se charge des boucles d'asservissement pour contrôler le temps de l'interface physique sur l'information fournie par le maître ;
- `chrony` qui vient sous la forme d'une paire *daemon* `chronyd` et l'applicatif pour sa configuration depuis l'espace utilisateur `chronyc`.

Découpons les étapes pour passer une CM4 en *Grandmaster* PTP commandé par GPS :

1. Nous commençons par tuer tout mécanisme de synchronisation qui pourrait interférer avec notre démonstration :

```
/etc/init.d/S49ntp stop
/etc/init.d/S65ptp4l stop
/etc/init.d/S49ntpd stop
/etc/init.d/S65ptpd2 stop
killall ntpd
killall ntp
killall phc2sys
killall ts2phc
killall pmc
killall ptp4l
```

C'est un peu violent, mais ainsi nous savons où nous en sommes au départ, et en particulier avons volontairement retiré toute référence à un service NTP qui pourrait interférer avec nos mesures.

2. Nous passons la broche connectée au signal 1-PPS en entrée pour lire l'impulsion venant du récepteur GPS :

```
./testptp -d /dev/ptp0 -L0,1
./testptp -d /dev/ptp0 -e 5
```

qui doit répondre quelque chose comme :

```
external time stamp request okay
event index 0 at 199.438950576
event index 0 at 200.438953296
event index 0 at 201.438956024
event index 0 at 202.438958760
event index 0 at 203.438961488
```

pour prouver que le signal 1-PPS est reçu. La date de l'évènement s'incrémente bien de 1 s.

3. Nous lançons l'asservissement de l'interface PTP sur le signal 1-PPS :

```
ts2phc -f ptp_gps.config -s nmea -m -q -l 7 >ts2phc_gps.log &
```

avec `ptp_gps.config` qui contient le port de communication avec le récepteur GPS et son débit de communication, ici pour une communication par convertisseur RS232-USB `/dev/ttyUSB0`, qu'on remplacera par `ttyAMA0` si l'UART de la CM4 est utilisé après avoir désactivé la console dans `cmdline.txt` au démarrage :

```
[global]
ts2phc.nmea_serialport /dev/ttyUSB0
ts2phc.nmea_baudrate 38400
leapfile /usr/share/zoneinfo/leap-seconds.list
tx_timestamp_timeout 100
[eth0]
```

Si tout se passe bien, le fichier `ts2phc_gps.log` nous informe, grâce au niveau de log 7 (tout afficher) dans un premier temps, que les trames NMEA sont bien lues et décodées pour être comprises :

```
ts2phc[450.950]: nmea sentence: GNRMC,095001.00,A,4715.09446,N,00559.58923,
E,0.053,,200723,,,A,V
ts2phc[450.975]: nmea sentence: GNGGA,095001.00,4715.09446,N,00559.58923,E,
1,09,1.62,309.3,M,47.1,M,,
```

Ensuite, nous vérifions que la réception des impulsions 1 PPS fonctionne bien et commande l'interface physique. Au cours de cette acquisition, `grep offset ts2phc_gps.log` doit indiquer un retard qui diminue pour tendre vers quelques nanosecondes de la forme :

```
ts2phc[451.970]: /dev/ptp0 offset -1689846181041193840 s0 freq      -0
ts2phc[453.037]: /dev/ptp0 offset -1689846181041195248 s1 freq   -1408
ts2phc[454.103]: /dev/ptp0 offset    -875736 s2 freq -877144
ts2phc[454.903]: /dev/ptp0 offset    -191884 s2 freq -456013
ts2phc[455.970]: /dev/ptp0 offset     270614 s2 freq  -51080
ts2phc[457.037]: /dev/ptp0 offset     354877 s2 freq +114367
...
ts2phc[470.103]: /dev/ptp0 offset        -20 s2 freq   -1417
ts2phc[470.903]: /dev/ptp0 offset        -32 s2 freq   -1435
```

Autant dans ce fichier de log que lorsque nous lisons les statuts dans `/tmp/message*`, **[16]** nous informe de l'état de l'asservissement selon *s0=unlocked*, *s1=clock step* et *s2=locked*.

Si lors de son lancement, `ts2phc` se plaint que `ts2phc[3760.610]: nmea: utc time is past leap second table expiry date` alors soit le fichier qui informe du nombre de secondes intercalaires a expiré avant la date actuelle, soit il est simplement inexistant (ce fut le cas dans Buildroot). Dans les deux cas, on pourra chercher à **https://www.ietf.org/timezones/data/leap-seconds.list** sa dernière version et placer le fichier dans `/usr/share/zoneinfo` en s'assurant que la date d'expiration :

```
# File expires on:  28 December 2023
#@ 3912710400
```

est postérieure à la date courante.

Cette étape d'asservissement de PTP sur GPS validée, le processus `ts2phc` est tué puisque nous le relancerons ultérieurement dans une configuration différente.

Maintenant que nous sommes persuadés que le 1-PPS GPS est vu par l'interface PTP, nous devons configurer la CM4 comme grand maître PTP asservi sur GPS.

1. Après avoir tué le processus `ts2phc`, lancé manuellement auparavant, par `killall ts2phc`, nous consultons les variables d'environnement proposées à **https://github.com/jclark/rpi-cm4-ptp-guide/blob/main/files/ptp4l-gm** qui règlent les paramètres du grand maître pour encourager ses esclaves à écouter ses consignes.
2. Nous lançons `ptp4l` en exploitant l'interface Ethernet (-2 pour IEEE802.3, en opposition à IP) :

```
ptp4l --serverOnly 1 -i eth0 -l 6 --tx_timestamp_timeout 100 -f ptp_gps.config &
```

qui nous indiquera dans `/var/log/messages` que :

```
ptp4l: [449.008] selected /dev/ptp0 as PTP clock
...
ptp4l: [455.266] selected local clock dca632.fffe.e8e477 as best master
```

3. Nous utilisons le *PTP Management Client* `pmc` pour configurer le grand maître :

```
pmc -u -b 0 \
"set GRANDMASTER_SETTINGS_NP
   clockClass              6
   clockAccuracy           0x23
   offsetScaledLogVariance 0xffff
   currentUtcOffset        37
   leap61                  0
   leap59                  0
   currentUtcOffsetValid   1
   ptpTimescale            1
   timeTraceable           1
   frequencyTraceable      0
   timeSource              0x20
"
```

avec la `clockAccuracy` qui définit la performance de la synchronisation, ici 1 µs pour 0x23 (allant de 0x20 pour 25 ns à 0x25 pour 10 µs). Nous pourrons à tout moment vérifier la validité de cette configuration par `pmc -u -b 0 'get GRANDMASTER_SETTING_NP'`.

4. Nous lançons `phc2sys` qui fait le lien entre horloge PTP et système avec :

```
phc2sys -s eth0 -E ntpshm -O -37 -N 1 -R 0.07 -l 7 &
ts2phc -f ptp_gps.config -s nmea -c eth0 -l 6 --clock_servo pi --step_threshold 0.1 --tx_
timestamp_timeout 100 \
  --ts2phc.nmea_serialport /dev/ttyUSB0 --leapfile /usr/share/zoneinfo/leap-seconds.list &
```

Ici, `ts2phc` se contente d'un niveau de log de 6 puisque nous avons déjà validé auparavant son bon fonctionnement et voulons éviter de remplir inutilement `/tmp/messages` de toutes les trames NMEA lues, tout en reprenant la configuration du port série de communication dans `ptp_gps.config`.

Si `/var/log/messages` indique `ts2phc: [XXX.YYY] source ts not valid`, nous avons constaté qu'un `stty -F /dev/ttyUSB0 38400 && cat < /dev/ttyUSB0` permet de le réveiller.

Finalement, le *daemon* `chronyd` a normalement été lancé au démarrage du système, afin que son outil associé `chronyc` en espace utilisateur qui prend en argument `sources` puisse valider que seul le GPS est utilisé pour asservir le temps et lire les trames NMEA pour connaître la date, et non pas un autre serveur PTP ou NTP. Ainsi, une fois toutes ces étapes complétées, la commande `chronyc sources` doit donner une unique source de la forme :

```
MS Name/IP address         Stratum Poll Reach LastRx Last sample
===============================================================================
#? GPS                           0   4     0     -     +0ns[   +0ns] +/-    0ns
```

avec un nombre de mesures lues initialement nul, mais qui finiront pas s'incrémenter avec un délai. Avec un peu de patience, nous finirons par obtenir :

```
MS Name/IP address         Stratum Poll Reach LastRx Last sample
===============================================================================
#? GPS                           0   4    10    85  -7048us[-7048us] +/- 1000us
```

Le fichier `/etc/chrony.conf` contient un grand nombre d'options, mais la plus importante semble être :

```
refclock SHM 0 refid GPS poll 4 precision 1e-3 offset 0
```

parmi :

```
#Connect to GPS Atomic Clock
refclock SHM 0 refid GPS poll 4 precision 1e-3 offset 0
# Allow the system clock to be stepped in the first three updates
# if its offset is larger than 1 second.
makestep 1.0 3
# Enable hardware timestamping on all interfaces that support it.
hwtimestamp *
# Get TAI-UTC offset and leap seconds from the system tz database.
leapsectz right/UTC
```

Une fois que tous les asservissements fonctionnent, `/tmp/messages` montrera :

```
phc2sys: [1264.140] CLOCK_REALTIME phc offset    244552 s0 freq  +0 delay  64019
...
ts2phc: [1275.768] /dev/ptp0 offset         13 s2 freq   +3951
ts2phc: [1276.776] /dev/ptp0 offset          8 s2 freq   +3950
ts2phc: [1277.784] /dev/ptp0 offset          2 s2 freq   +3946
phc2sys: [1278.427] CLOCK_REALTIME phc offset    249129 s0 freq  +0 delay  66167
ts2phc: [1278.792] /dev/ptp0 offset         16 s2 freq   +3961
ts2phc: [1279.800] /dev/ptp0 offset          9 s2 freq   +3959
ts2phc: [1280.808] /dev/ptp0 offset         18 s2 freq   +3970
```

Nous discuterons du statut `s0` de `phc2sys` ci-dessous, mais `chrony` qui prend en charge la distribution du temps compte tenu de l'option `-E ntpshm` (*NTP Shared Memory* décrit à **https://www.ntp.org/documentation/drivers/driver28/**) indique par `chronyc tracking` que :

```
Reference ID    : 47505300 (GPS)
Stratum         : 1
Ref time (UTC)  : Mon Jul 31 05:48:18 2023
System time     : 0.000001597 seconds fast of NTP time
Last offset     : +0.000001611 seconds
RMS offset      : 0.000148294 seconds
Frequency       : 5.687 ppm slow
Residual freq   : +0.001 ppm
Skew            : 0.158 ppm
Root delay      : 0.000000001 seconds
Root dispersion : 0.001048921 seconds
Update interval : 64.1 seconds
Leap status     : Normal
```

qui n'est pas stupide quand le compteur de fréquence indique 53999625 ou un écart de $(53999625 - 54 \cdot 10^6)/54 = -6.9$ ppm. Au contraire :

```
Reference ID    : 47505300 (GPS)
Stratum         : 1
Ref time (UTC)  : Mon Jul 31 06:08:54 2023
System time     : 0.000002803 seconds fast of NTP time
Last offset     : +0.000006581 seconds
RMS offset      : 0.000180112 seconds
Frequency       : 2.128 ppm fast
Residual freq   : +0.005 ppm
Skew            : 0.276 ppm
Root delay      : 0.000000001 seconds
Root dispersion : 0.001100202 seconds
Update interval : 64.1 seconds
Leap status     : Normal
```

indique que l'horloge est trop rapide de 2,128 ppm, en accord avec l'observation d'une fréquence d'oscillateur de 54 MHz+46 Hz.

Nous avions auparavant confirmé que l'argument `freq` de `phc2sys` est représentatif de l'écart de fréquence à la valeur nominale (Fig. 14), mais en lisant les logs, nous constatons qu'avec la configuration qui s'appuie sur `chrony`, le statut de `phc2sys` reste à `s0` (*unlocked*). Même si J. Clark affirme préférer `chrony` pour sa souplesse de configuration, le fait est que son information d'erreur de fréquence en ppm (partie par million) est peu précise et difficilement exploitable. Nous pouvons revenir au fonctionnement précédent en retirant le partage de mémoire avec `chrony` tel qu'il était configuré par `-E ntpshm` pour le remplacer par `-E pi` choisi par défaut, qui active le contrôleur proportionnel intégral qui une fois accroché indique à nouveau :

```
phc2sys: [7157.917] CLOCK_REALTIME phc offset 82251 s2 freq +11476 delay 62983
```

avec `s2` pour indiquer que l'asservissement fonctionne. C'est cette information de fréquence `freq` que nous allons utiliser pour asservir la fréquence d'oscillation en ajustant la capacité de tirage. Cependant, la CM4 ne possède pas de convertisseur analogique-numérique qui pourrait polariser la varicap : nous allons user d'un autre stratagème pour produire le signal de commande. Notez dès à présent que l'asservissement proportionnel intégral de `-E pi` sera source d'ennuis et sera remplacé finalement par `-E linreg`.

6. SYNTONISATION DE LA CM4 SUR L'IMPULSION 1-PPS GPS

Le processeur de la CM4 propose un certain nombre de broches supportant la modulation en largeur d'impulsion (*Pulse Width Modulation* PWM) commandée non pas par logiciel, mais par un compteur matériel donc immune aux fluctuations de charge du processeur généraliste. Une PWM produit, après passage du signal de période fixe, mais de rapport cyclique variable dans un filtre passe-bas une tension constante entre 0 et sa tension maximale, égale au rapport de la tension maximale au rapport cyclique.

La configuration de la PWM est supportée par `python-pigpio` disponible comme paquet dans Buildroot en complément du *daemon* `pigpio` que nous devrons aussi compiler pour la cible. Une fois ces outils disponibles :

```
import pigpio
pi=pigpio.pi()
pi.hardware_PWM(18, 1000000,750000)
```

configure la broche 18 pour générer une PWM de fréquence 1 MHz et de rapport cyclique de 75 %. Ainsi, bien que la CM4 ne possède pas de convertisseur analogique-numérique, le VCO est contrôlé par la PWM dont le rapport cyclique détermine la tension de polarisation une fois filtrée par un circuit RC de fréquence de coupure bien inférieur à la fréquence de la PWM. Afin de ne pas s'encombrer de Python, il semble plus facile, en shell, de commander :

```
pigs HP 18 2000000 200000
```

pour définir la fréquence de la PWM de la broche 18 à 2 MHz et de rapport cyclique 20 %. Cependant, le choix de la fréquence de la PWM fPWM est un compromis entre la capacité à filtrer la sortie oscillante pour produire un signal de commande continu, et la résolution de la commande

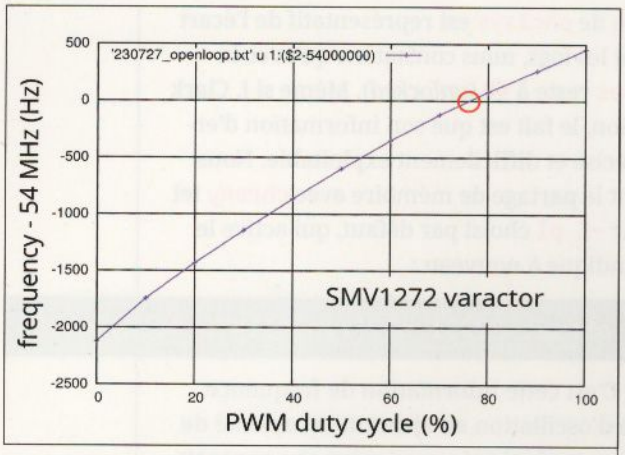
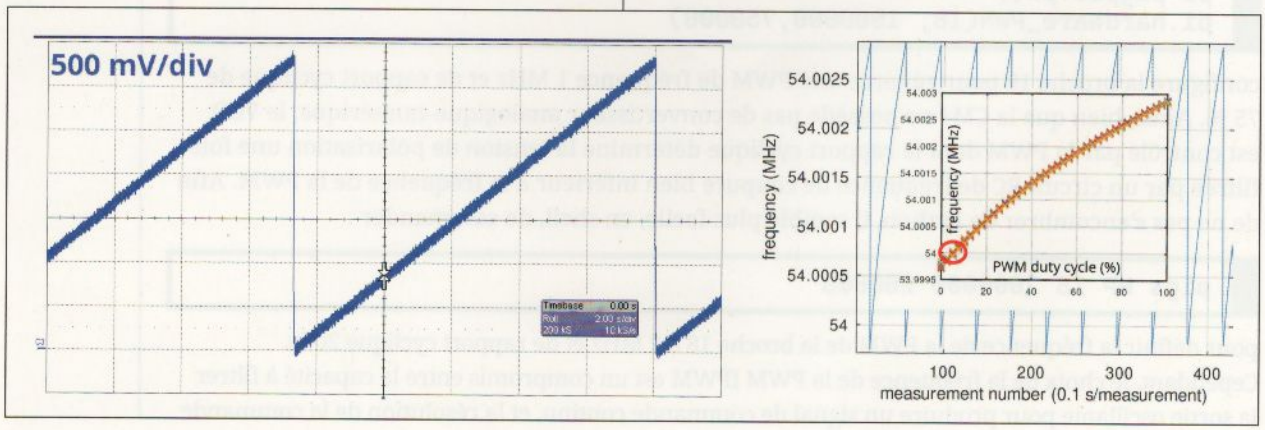


Figure 16 : Relation fréquence-tension pour un rapport cyclique de la PWM variant de 0 % (0 V) à 100 % (3,3 V) en utilisant une varicap Skyworks SMV1272. Le point de fonctionnement recherché est mis en évidence par le cercle rouge lorsque la fréquence de l'oscillateur est 54 MHz. La fréquence de l'oscillateur est mesurée par un compteur HP53131A.

Figure 17 : Gauche : capture d'écran d'oscilloscope représentant les rampes de tension en sortie de PWM configurée avec une période de 10 µs (fréquence 100 kHz) et un rapport cyclique variable par le script bash while true; do for i in 'seq 0 10000 1000000'; do pigs HP 18 100000 $i;sleep 0.1;done;done lorsque la broche 18 est connectée à un circuit RC formé d'une résistance en série de 10 kΩ et une capacité de 1,1 µF en parallèle avec la masse. Droite : mesure de fréquence dans le domaine temporel (arrière-plan) et réinterprétée comme la fréquence en fonction de la commande du rapport cyclique de la PWM (insert) en superposant toutes les mesures acquises dans les mêmes conditions. Pour cette configuration, la varicap est une vénérable BB833 et la capacité en série du résonateur vaut 13+15=28 pF.

puisque le nombre de rapports cycliques est égal à $375 \cdot 10^6 / f_{PWM}$ **[17]**. En effet, choisir une fréquence élevée rend le filtrage plus aisé avec un simple filtre passe-bas RC, mais réduit la granularité de la commande. Ainsi, pour une commande sur 12 bits, la fréquence de la PWM sera limitée à un peu moins de 100 kHz et le filtre passe-bas devra couper à une fraction de cette fréquence, laissant tout de même assez de marge pour mettre à jour la tension de polarisation de la varicap et ajuster la fréquence selon l'information fournie périodiquement par phc2sys (Fig. 14).

Avant tout asservissement, nous validons la capacité de tirage de l'oscillateur en boucle ouverte. Pour ce faire, la tension de commande étant limitée à 0-3,3 V par la PWM filtrée, nous devons identifier une varicap capable de faire varier la capacité dans cette gamme de tensions. Le composant SMV1272 de Skyworks semble parfaitement adapté avec une capacité qui varie de 16 à 6 pF entre 1 et 3,3 V pour monter à près de 30 pF à 0 V de polarisation, suffisamment proche des 18 pF des condensateurs de pieds du montage Pierce seul pour ne pas faire décrocher l'oscillateur. Nous avons cependant constaté qu'il est peu judicieux de trop baisser la tension de commande, la CM4 cessant de fonctionner si son oscillateur devient instable. La caractéristique en boucle ouverte de la fréquence d'oscillation en fonction de la tension est fournie en Fig. 16 ou 17 selon les modèles, mais devra être reévaluée par chaque CM4, la gamme de tirage étant relativement réduite cependant. Néanmoins en accord avec les attentes (annexe A),



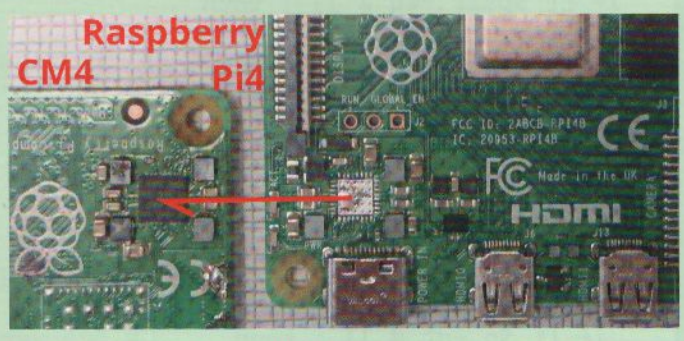
augmenter la tension de polarisation de la varicap abaisse la capacité en parallèle du condensateur de pieds et fait monter la fréquence de l'oscillateur.

D'ailleurs, on peut même se demander pourquoi se fatiguer à commander la fréquence de l'oscillateur et ne pas le laisser vivre tranquillement sa vie. La Fig. 18 illustre l'impact du changement de température du résonateur, que nous avions mentionné auparavant en chauffant par un fer à souder, mais cette fois simplement lors de la mise sous tension de la CM4 avec son processeur à basse fréquence par la configuration par défaut de Buildroot (`powersave`) puis le passage au cadencement le plus rapide (`performance`) avec l'augmentation de fréquence qui se traduit immédiatement par une consommation accrue et donc une élévation de température. Lors du passage de la fréquence de cadencement du processeur de 600 à 1500 MHz, la consommation globale de la carte Compute Module 4 IO passe de 230 mA à 300 mA sous 5 V, soit un passage de la consommation globale de 1,15 W à 1,5 W. Ce résultat est en accord avec celui présenté à **http://www.ntp.org/ntpfaq/NTP-s-sw-clocks-quality/** où « *the frequency of the oscillator's correction value increases by about 11 PPM after powering up the system. This is quite likely due to the increase of temperature.* »

Vie et mort ... et vie d'une Compute Module 4

L'histoire faillit se finir à ce stade lorsque par maladresse, une sonde de multimètre court-circuita les broches 1 et 2 du bornier d'extension de la carte mère CM4-IO. Il ne faut **jamais** court-circuiter 1 (5 V) et 2 (3,3 V) sous peine d'irrémédiablement détruire le PMIC – *Power Management Integrated Circuit* qui alimente la CM4 (**https://forums.raspberrypi.com/viewtopic.php?t=323594**). Je sais enfin pourquoi les étudiants me rendent des Raspberry Pi 4 détruites en fin de projet.

Mais si la Raspberry Pi4 est sujette aux mêmes déboires, peut-être pourrons-nous sauver l'unique modèle de CM4 à notre disposition avec une des multiples Raspberry Pi 4 utilisées en enseignement. Après un prélèvement d'organe quelque peu forcé d'une Raspberry Pi 4 qui passait à proximité et une transplantation de cœur^wPMIC vers la CM4, cette dernière ressuscita et permet de continuer l'histoire. Ce n'est pas très déontologique, mais la raison du plus fort est toujours la meilleure.



Le PMIC de la Raspberry Pi 4 (droite) est compatible avec celui de la CM4 (gauche) et permit de ressusciter cette dernière après une manipulation malheureuse.

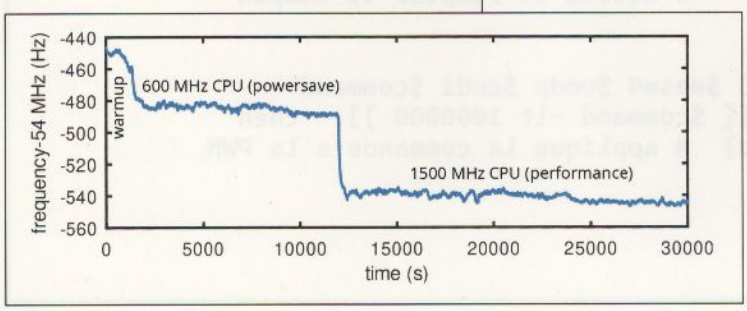


Figure 18 : Échauffement du circuit imprimé de la CM4 lors de sa mise sous tension (warmup) induisant la dérive de l'oscillateur, puis échauffement accru lors de l'augmentation de la fréquence de cadencement du processeur configuré autour de l'abscisse 12000 pour passer du mode powersave à 600 MHz à performance à 1500 MHz.

La finalité de cette présentation tient donc en un script bash qui périodiquement sonde la fréquence de l'oscillateur, par exemple en recherchant (`tail /var/log/messages`) les dernières informations de `phc2sys` dans les logs du système, pour en extraire l'information du champ `freq` et commander par `pigs` la PWM en conséquent. La principale subtilité tient aux protections contre les dysfonctionnements (messages autres que ceux de `phc2sys`), contre les valeurs aberrantes, et contre les commandes en dehors des plages autorisées. Le script s'allonge donc petit à petit avec les conditions de protection `if () ;then ... fi` pour se conclure en :

```
command=89770  # initial setpoint
pigs HP 18 100000 ${command}
mesm4=0;mesm3=0;mesm2=0;mesm1=0
pair=1;sumpair=0
n=0
Kp=0.1;Ki=0.1
moy=0;cmdi=0
while true; do
 d=`date +%s` ; echo -n $d " "
 df=`tail -30 /var/log/messages | grep phc2sys | tail -1 | cut -dq -f2 | cut
-dd -f1`
 if [[ $df -ne $mesm1 ]] ; then
  moy=$((moy-(mesm4)));moy=$((moy+(df)))  # proportional x[n+1]-x[n]
  mesm4=$mesm3;mesm3=$mesm2;mesm2=$mesm1;mesm1=$df
  sumpair=$((sumpair+df))
  if [[ $pair -eq 2 ]] ; then
   pair=0
   if [[ $sumpair -lt 50000 ]] &&  [[ $sumpair -gt -50000 ]]; then
    cmdi=`echo "${sumpair}*16.632/4*${Ki}" | bc | cut -d\. -f1`
    sumpair=0
   fi
  fi
  pair=$((pair+1))
  cmdp=`echo "${moy}*16.632/4*${Kp}" | bc | cut -d\. -f1`
  if [ -n "$cmdp" ] && [ "$cmdp" -eq "$cmdp" ] && [ -n "$cmdi" ] && [
"$cmdi" -eq "$cmdi" ] ; then
   if [[ $n -gt 4 ]] ; then      # ^^^ verifie que cmdp et cmdi sont bien
                                 # des nombres
    command=$((command-(cmdp)-(cmdi)))  # PI controller
    n=5
   else
    n=$((n+1))                   # attend de remplir le tampon
   fi
  fi
  echo $n $mesm1 $mesm2 $mesm3 $mesm4 $cmdp $cmdi $command
  if [[ $command -gt 0 ]] && [[ $command -lt 1000000 ]];  then
   pigs HP 18 100000 ${command}  # applique la commande a la PWM
  fi
 fi
 sleep 5
done
```

Le script commence par initialiser des variables et notamment la commande `command`, partant d'une valeur qui place l'oscillateur pas trop loin de la fréquence visée de 54 MHz, c.-à-d. l'argument de `pigs` qui produit un rapport cyclique qui polarise la varicap afin d'ajuster l'oscillateur convenablement. Nous verrons ci-dessous pourquoi nous aurons besoin d'une moyenne glissante sur un nombre pair de valeurs `mesmi` avec $i \in [1:4]$ les valeurs passées de la mesure, et un compteur de parité `pair`. Nous implémentons un correcteur de type proportionnel intégral tel que $command_n = K_p \cdot erreur_n + K_i \int_1^n erreur_n$ avec erreur l'erreur de fréquence fournie par `phc2sys` que nous cherchons à annuler. Il est classique dans ce contexte de calculer $command_{n+1} - command_n = K_p(erreur_{n+1} - erreur_n) + K_i\ erreur_n$ afin d'éviter d'accumuler une intégrale qui pourrait diverger, et ainsi ne calculer que l'*évolution* de la commande de la forme $command_{n+1} = command_n + K_p(erreur_{n+1} - erreur_n) + K_i\ erreur_n$. Une fois la commande calculée, nous vérifions que la valeur de la PWM est dans la gamme autorisée entre 0 et 10^6, avant de commander la PWM. La majorité des tests vise à vérifier que les arguments sont des nombres et ne contiennent pas de chaîne de caractères qui feraient planter les opérations arithmétiques, notamment lors de la mise à jour de la commande qui fait intervenir `cmdp` et `cmdi` les évolutions par le terme proportionnel et le terme intégral. Par ailleurs, on notera que `bc` de Buildroot, qui effectue toujours des opérations avec des décimales que la variable `scale` ne permet pas d'ajuster, ne se comporte pas comme celui de notre système Debian/GNU Linux qui par défaut, en l'absence de l'option `-l`, fournit des résultats entiers tel que nous les recherchions, forçant un appel à `sed` pour éliminer la partie fractionnaire.

Nous avons constaté que l'oscillateur varie sur une plage de 3250 Hz entre une tension de polarisation de la varicap de 0 V à 3,3 V. Nous savons que le nombre de pas de commande de la PWM de fréquence f est $375 \cdot 10^6/f$ donc 3750 pas lorsque f = 100 kHz. Ainsi, un pas de fréquence varie la fréquence d'oscillation de 3250/3750 = 0,87 Hz et le meilleur asservissement que nous pouvons espérer est $0{,}87/54 = 0{,}016$ ppm ou $16 \cdot 10^{-9}$. Ce n'est pas terrible, mais mieux que rien.

La surprise vient de l'observation du champ `freq` de `phc2sys` extrait de `/var/log/messages` en garantissant de ne prendre que les valeurs successives, ne connaissant pas le taux de rafraîchissement du message, en rejetant les nouvelles valeurs égales aux anciennes. Nous constatons que les mesures fluctuent énormément d'une estimation à l'autre... mais que la moyenne des valeurs deux par deux donne une bonne estimation de la fréquence de l'oscillateur. La cause de ce fonctionnement n'est pas connue, mais c'est un constat : en moyennant (`conv(...,[1 1])` sous GNU Octave) un nombre pair de mesures, nous éliminons les fluctuations et obtenons un estimateur de l'écart de fréquence de l'oscillateur à la fréquence GPS exploitable (Fig. 19).

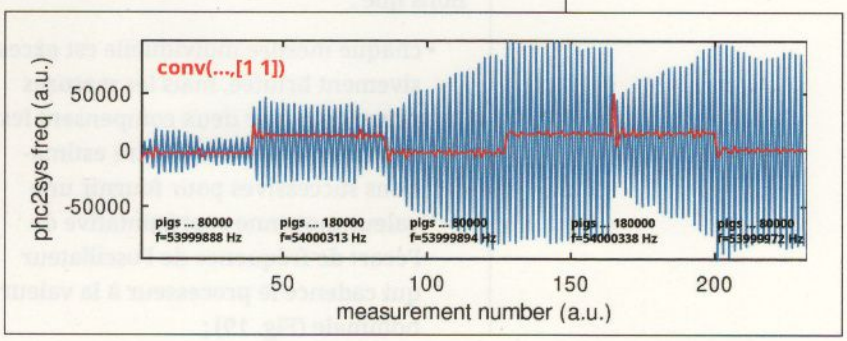


Figure 19 : Évolution du champ freq des messages de phc2sys dans /var/log/messages (bleu) et moyenne des valeurs adjacentes (rouge). Le texte en dessous de la courbe indique la commande de la PWM définissant la tension qui polarise la varicap, et la fréquence résultante de l'oscillateur observée. Noter comment, avec l'asservissement de type -E pi de phc2sys, la mesure de fréquence tend à diverger en prenant des valeurs de plus de 50000, avant de devenir incontrôlable dans un délai de 24 h tout au plus.

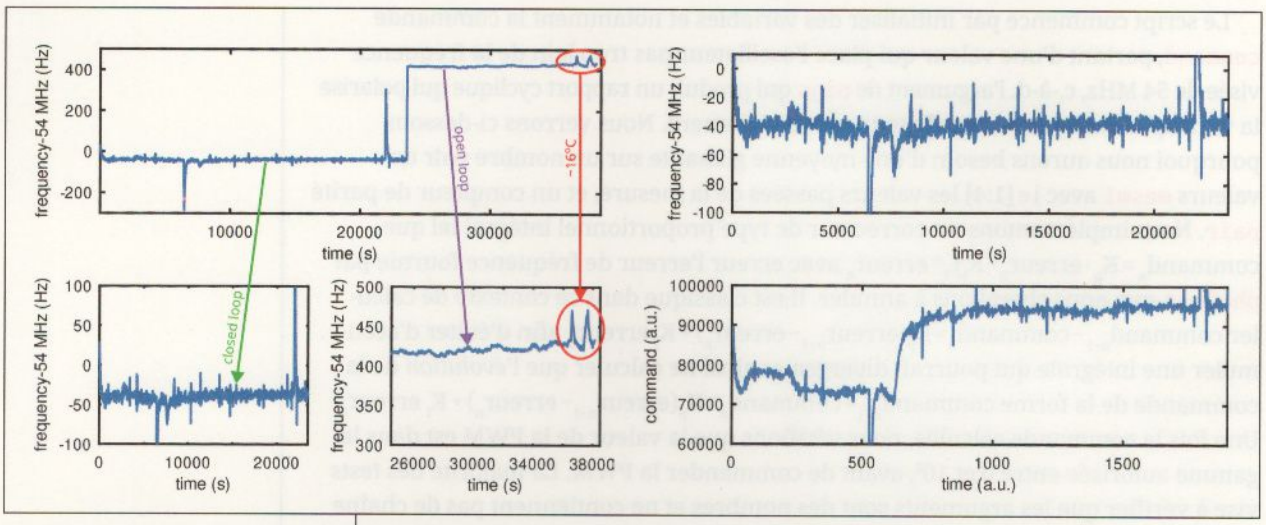


Figure 20 :
Gauche : en haut l'évolution de la fréquence de l'oscillateur qui cadence le processeur, au début en boucle fermée puis en boucle ouverte quand le programme d'asservissement a cessé de fonctionner. Les deux sauts à droite de la courbe sont dus à une fenêtre ouverte exposant la CM4 à une baisse de température, la température extérieure étant annoncée à 16°C.
Droite : pour la zone en boucle fermée, en haut la fréquence et en bas la commande que nous constatons corriger la fréquence pour la maintenir constante alors que les conditions environnementales de la CM4 évoluent.

Nous avons la mesure sous la forme du champ `freq` de `phc2sys` dans `/var/log/messages`, nous avons la commande sous la forme de l'argument de `pigs` pour modifier la tension de polarisation de la varicap, nous avons la caractérisation en boucle ouverte, il ne reste plus qu'à fermer la boucle et asservir l'oscillateur de la CM4 sur GPS.

Dans un premier temps, un asservissement proportionnel avec $K_p = 0.1$ et $K_i = 0$ permet de s'approprier le comportement du système. Nous savons que la solution sera biaisée en l'absence de l'élimination de l'erreur constante par le terme intégral, mais au moins on aura un résultat (Fig. 20).

De cette première mesure, nous apprenons que :

- chaque mesure individuelle est excessivement bruitée, mais les mesures prises deux par deux compensent les fluctuations énormes entre estimations successives pour fournir une valeur moyenne représentative de l'écart de fréquence de l'oscillateur qui cadence le processeur à la valeur nominale (Fig. 19) ;
- néanmoins avec le contrôleur `-E pi` de `phc2sys`, nous avons été incapables d'obtenir un asservissement pendant plus de quelques heures, après lesquelles les valeurs successives des estimations de `freq` divergent vers des valeurs de plus de 105 que l'arithmétique de bash n'arrive plus à correctement appréhender, induisant une divergence de notre propre boucle d'asservissement et l'absence de correction qui a dépassé la gamme des commandes que peut comprendre la PWM. Il est de toute façon compliqué d'obtenir une estimation fiable d'écart de fréquence de quelques unités quand les valeurs successives fournies par `phc2sys` diffèrent de plusieurs centaines de milliers ;

Calcul de variance d'Allan

Nombre d'outils sont disponibles pour calculer une variance d'Allan, mais profitons de cette occasion pour promouvoir la solution locale développée par François Vernotte et François Meyer à l'observatoire de Besançon disponible à **https://gitlab.com/fm-ltfb/SigmaTheta**. Partant d'une série de mesures de fréquences f_n, $n \in N$ issues d'un compteur, nous sauvons un fichier ASCII contenant les écarts normalisés de fréquence $(f_n - \nu_0)/\nu_0$ à la valeur nominale de l'oscillateur ν_0, par exemple par `save -text fichier` de GNU Octave, contenant une unique colonne de la forme :

```
-1.8746481481878585e-06
-1.8725740740737567e-06
-1.8717592593231057e-06
...
```

Alors `1col2col -c < fichier | st_MDev > tmp` sauve dans `tmp` les variances d'Allan modifiées, et `uncertainties tmp` fournit les barres d'erreurs et les coefficients des asymptotes que nous sauvons pour Octave dans une matrice `sf`. Les barres d'erreurs sont tracées dans GNU Octave par :

```
errorbar(sf(:,1),sf(:,2),sf(:,2)-sf(:,4),sf(:,7)-sf(:,2))
set(gca, 'XScale','log', 'YScale','log')
```

puisque la colonne 1 contient le temps d'intégration, la colonne 2 la variance d'Allan modifiée, la colonne 4 la borne inférieure de la barre d'erreurs et la colonne 7 la borne supérieure. Noter que `uncertainties` fournit le script `gnuplot` pour atteindre le même résultat.

- les corrections à **court terme**, une fois toutes les 15 secondes environ, au rythme du rafraîchissement des messages de `phc2sys`, sont plus néfastes que bénéfiques à la stabilité de l'oscillateur, mais qu'à **long terme** ou en présence de perturbations telles qu'une fenêtre ouverte qui refroidit le circuit par un courant d'air, la correction est utile (Fig. 20). Cette observation se formalise par le calcul de la variance d'Allan des mesures de fréquences (Fig. 21) : au-delà de quelques centaines à milliers de secondes (1 h d'intégration), la courbe corrigée bleue passe en dessous de la courbe libre rouge, indiquant que la stabilisation a eu un effet positif. Les barres d'erreurs attestent de la validité de la conclusion.

Le deuxième point est corrigé en remplaçant la correction `-E pi` de `phc2sys` par `-E linreg` qui stabilise l'estimation de `freq` et la maintient stable pendant plusieurs heures, tel qu'inspiré du commentaire à **https://www.mail-archive.com/linuxptp-users@lists.sourceforge.net/msg02297.html** concernant la précision de l'estimation d'écart de fréquence par `phc2sys`. Apparemment, `linreg` est plus gourmand en ressources de calcul que `pi`, mais cela ne semble pas importer pour une CM4, n'incluant que des opérations arithmétiques déterministes (**https://github.com/nxp-archive/openil_linuxptp/blob/master/linreg.c**). De cette façon, les écarts de fréquence estimés par `phc2sys` restent, une fois l'asservissement fonctionnel, de

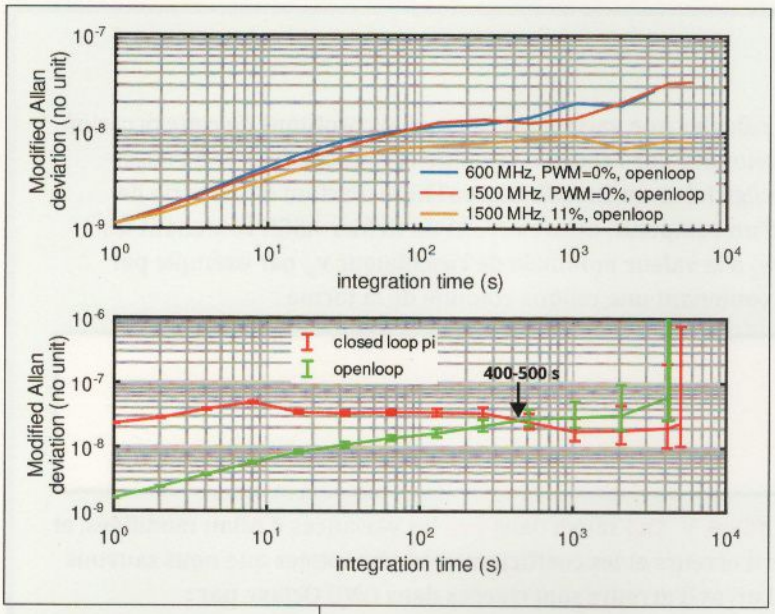


Figure 21 : Variance d'Allan de l'oscillateur en boucle ouverte en haut pour diverses configurations, que le processeur soit cadencé à 600 MHz ou 1500 MHz, avec une commande nulle ou une PWM à 11 % de rapport cyclique, qui ne semble avoir aucun impact significatif sur la stabilité. En bas : comparaison boucle ouverte/boucle fermée, avec l'asservissement de fréquence qui dégrade la stabilité court terme, mais stabilise l'oscillateur sur des durées d'intégrations de plusieurs centaines au millier de secondes (flèche verticale).

quelques unités même 24 h après avoir lancé la mesure, permettant de maintenir la boucle de contrôle de l'oscillateur fermée.

Le correcteur PI converge vers 53,999960 MHz ou un biais de -40 Hz ou -0,9 ppm à la consigne de 54 MHz lors d'une mesure avec un compteur Agilent 53131A cadencé par son pilote interne. Ce pilote a été mesuré a posteriori comme biaisé de -5 Hz à 10 MHz sur une référence métrologique, ou -0,5 ppm. Ces valeurs ne sont pas aberrantes vis-à-vis de la documentation technique *Agilent 53131A/132A 225 MHz Universal Counter Operating Guide* (page 3-4) qui indique un taux de vieillissement annuel de 0,3 ppm/mois de cet oscillateur dans sa version standard, pour être réduit d'un facteur 10 quand un oscillateur thermostaté (OCXO) est utilisé.

En conclusion de cette étude, nous démontrons le bénéfice de corriger, à long terme, l'oscillateur soumis aux aléas de son environnement sur une référence stable qu'est l'impulsion 1-PPS GPS (Fig. 22).

Ce faisant, du fait d'une correction trop rapide polluée par une mesure grossière de la fréquence par `phc2sys` et d'un pas de correction insuffisamment précis, nous avons dégradé la stabilité court terme de l'oscillateur. Fort de Fig. 21, nous constatons que l'intersection entre la courbe libre et la courbe asservie se trouve sur un temps d'intégration de quelques centaines de secondes, qui indique que la constante de temps de l'asservissement devrait être de quelques dizaines d'échantillons acquis dans `/var/log/messages` toutes les 15 secondes environ. La résolution de la correction égale à 0,87 Hz ne devrait pas handicaper le contrôleur en l'état, mais un « vrai » DAC sur bus SPI ou I²C ne pourrait qu'être bénéfique pour réduire le pas de correction. Il est fort probable qu'un lecteur compétent en automatique sache mieux régler les coefficients du correcteur proportionnel et intégral à la vue de la réponse indicielle de la Fig. 22, puisque les oscillations résiduelles lors du retour à la consigne sont clairement visibles dans les encadrés en haut à gauche.

CONCLUSION

Nous avons abordé l'utilisation de l'information temporelle disponible auprès d'un noyau Linux et en pratique la datation des fichiers, et ce dans un contexte de synchronisation d'un réseau distribuant les ressources. Nous avons décrit les deux modes de synchronisation par logiciel NTP et par matériel PTP dans des

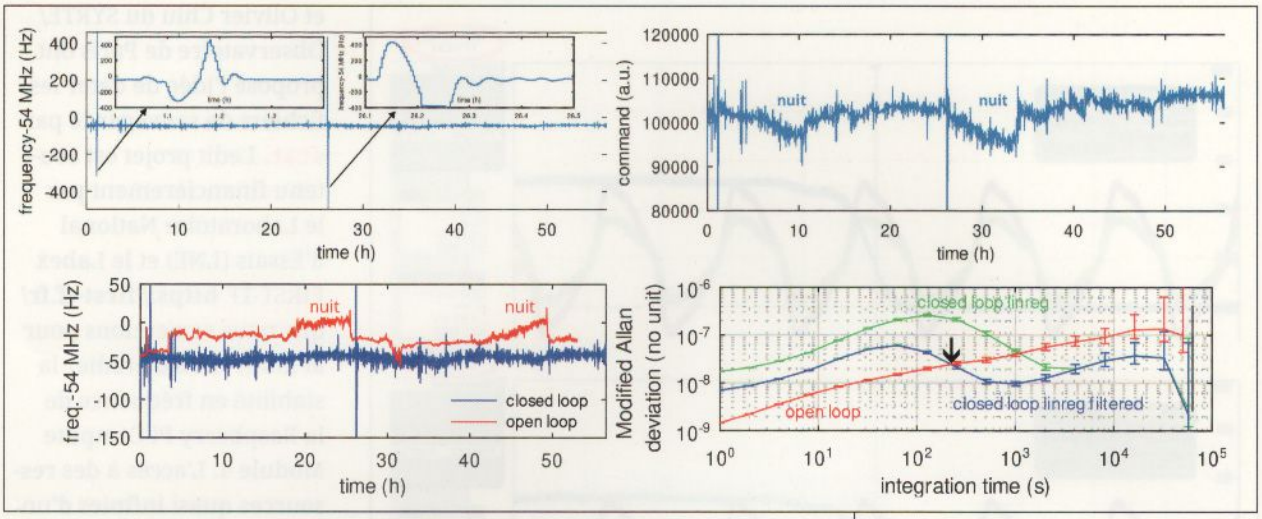


Figure 22 : Gauche : en haut, la mesure en boucle fermée de l'évolution temporelle de la fréquence de l'oscillateur au cours de plus de 48 h pour mettre en évidence le bénéfice de l'asservissement, et en bas la commande appliquée à la PWM pour polariser la varicap. Les deux dérives soudaines de fréquence inexpliquées (encadré en haut) – la seconde en pleine nuit sans interférence volontaire possible – ont été éliminées lors de l'analyse statistique en variance d'Allan. Au cours de cette mesure, phc2sys est asservi par linreg. Droite : mesure pendant plus de 48 h en boucle ouverte (haut), et analyse en variance d'Allan modifiée (bas) mettant en évidence la dégradation de la stabilité court terme avec l'asservissement de fréquence, mais le bénéfice sur le long terme pour des temps d'intégration dépassant les quelques centaines de secondes. Ici aussi, la constante de temps nécessaire pour un asservissement optimal est indiquée par la flèche noire verticale.

conditions réelles et idéales en contrôlant tous les instruments sur une même source de fréquence supposée parfaite. Ce faisant, nous avons découvert l'impact de la fréquence du compteur qui définit le temps dans le noyau, et identifié la nécessité de syntoniser l'oscillateur qui cadence le processeur, en plus de le synchroniser. Finalement, nous avons étendu cette solution à tout possesseur d'un récepteur GPS capable de fournir l'horodatage qu'est l'impulsion 1-PPS qui fournit alors la source de synchronisation PTP. Une solution d'asservissement de l'oscillateur qui cadence le processeur de la Raspberry Pi Compute Module 4 a été proposée. Sans prétendre atteindre les performances de White Rabbit – en l'état tout au moins – la syntonisation du processeur sur le signal PTP semble amener la perspective d'exploiter le signal de datation précis au noyau et donc à l'espace utilisateur (Figure 23, page suivante).

Au-delà de la solution matérielle de syntonisation proposée, nos explorations de la gestion du temps par le noyau Linux nous ont amenés à découvrir l'appel système `adjtimex` et l'outil en espace utilisateur qui lui est associé dans Busybox de Buildroot dans *misc → adjtimex*, mais qui se contente d'afficher l'erreur relative de fréquence sans proposer de valeur absolue. Ainsi, **http://www.ep.ph.bham.ac.uk/general/support/adjtimex.html** explore la gestion du temps par cet outil, mais nous n'avons pas persévéré dans cette voie qui reste ouverte de modifier non pas physiquement l'oscillateur qui cadence le processeur, mais la perception du temps qu'a le noyau cadencé par un oscillateur faux.

Il semble donc peu probable que l'information temporelle fine fournie par PTP soit exploitable lors des appels système du noyau Linux.

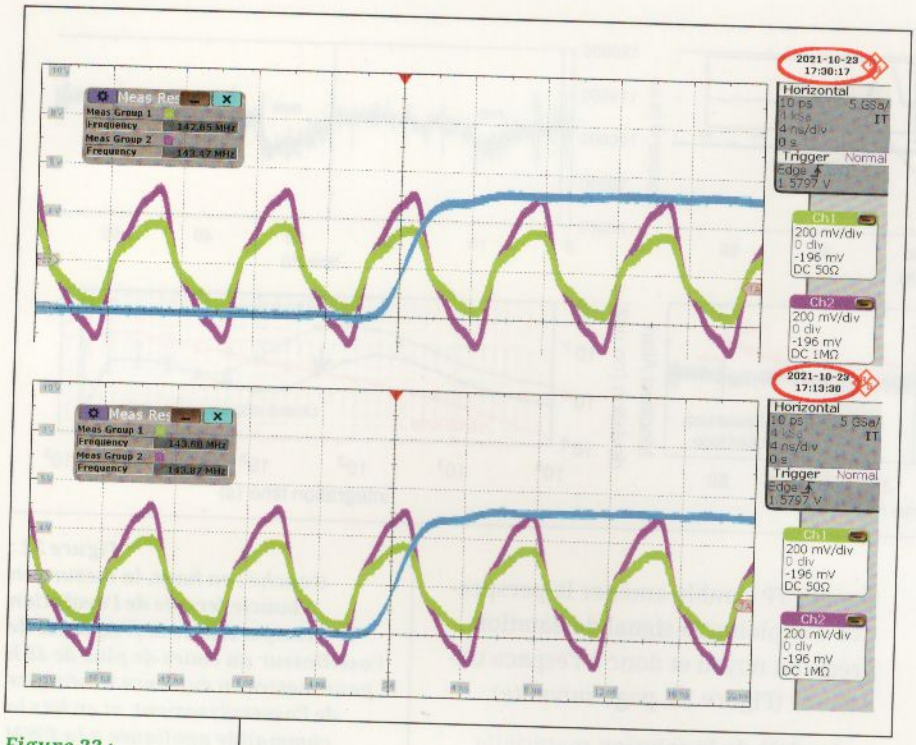


Figure 23 : Deux signaux 143 MHz produits par deux AD9548 (vert, rouge) synchronisés sur une même impulsion 1-PPS (bleu) distribuée par White Rabbit.

La seule utilité de la synchronisation des interfaces physiques Ethernet reliées au réseau informatique reste donc la production de l'impulsion 1-PPS en vue de générer un signal radiofréquence physique cohérent, par exemple au moyen du « *Network Clock Generator/Synchronizer* » AD9548 de Analog Devices qui produit un signal jusqu'à 450 MHz asservi sur une entrée 1-PPS (Fig. 23).

REMERCIEMENTS

Cette étude est un effet de bord imprévu d'un projet concernant le transfert de temps par satellite géostationnaire qui nécessite de synchroniser la date de sauvegarde de fichiers à Paris et Besançon en s'appuyant sur une base de temps commune échangée par NTP : Baptiste Chupin et Olivier Chiu du SYRTE/Observatoire de Paris ont proposé l'idée de dater les fichiers de sauvegarde par `stat`. Ledit projet est soutenu financièrement par le Laboratoire National d'Essais (LNE) et le Labex FIRST-TF **https://first-tf.fr/** que nous remercions pour la motivation d'étudier la stabilité en fréquence de la Raspberry Pi Compute Module 4. L'accès à des ressources quasi infinies d'un laboratoire de recherche public, et en particulier les références de temps et fréquences métrologiques, facilite l'obtention rapide de résultats, mais nous ne pouvons qu'encourager tout amateur éclairé à reproduire ces résultats. Malgré la volonté de faire disparaître la piézoélectricité de FEMTO-ST, Serge Galliou (ENSMM, Besançon) a encore pu partager son expérience de conception des oscillateurs à quartz et fourni de précieuses informations sur le tirage capacitif pour élever la fréquence d'oscillation.

A. MODÉLISATION DU RÉSONATEUR À QUARTZ DANS NGSPICE SOUS KICAD

Le simulateur de circuits électroniques `ngspice` est désormais intégré dans KiCad. Afin de simuler les conditions de Barkhausen du résonateur muni de ses deux condensateurs de pieds et éventuellement d'une capacité de tirage en série,

nous devons savoir comment modéliser le résonateur. Nous omettrons la porte logique supposée idéale, de gain infini et de rotation de phase de 180° comme toute porte inverseuse qui se respecte.

Comme tout circuit résonnant, un résonateur à quartz se modélise par une équation différentielle du second ordre, donc un circuit RLC en électronique. Dans cette branche dite motionnelle, la résistance R_1 représente les pertes (acoustiques pour le système physique), l'inductance L_1 la masse équivalente, et la capacité C_1 la raideur du ressort équivalent. Connaissant le facteur de qualité $Q = 1/R_1\sqrt{L_1/C_1}$ et la fréquence de résonance $f = 1/(2\pi\sqrt{L_1C_1})$, nous avons deux équations et trois variables à déterminer. Cependant, une originalité du résonateur à quartz par rapport au circuit RLC est d'être formé d'un morceau de diélectrique piézoélectrique – ici le quartz – couvert de deux électrodes. Deux électrodes de part et d'autre d'un diélectrique forment un condensateur $C_0 = \varepsilon_0\varepsilon_r S/d$ avec S la surface des électrodes, d l'épaisseur du substrat de quartz de permittivité relative ε_r qui détermine l'espacement entre les électrodes. On montre **[18]** que le ratio C_0/C_1 est déterminé par le coefficient de couplage électromécanique du substrat piézoélectrique, une propriété intrinsèque du matériau pour une orientation cristalline donnée. Si C_0 est donné par la géométrie et C1 par la nature du matériau, il ne reste plus de degré de liberté pour trouver R_1 et C_1. En pratique C_0 et C_1 sont donnés par le constructeur dans sa documentation technique (Fig. 11, droite avec le champ C_0) et nous en déduisons les autres paramètres utiles.

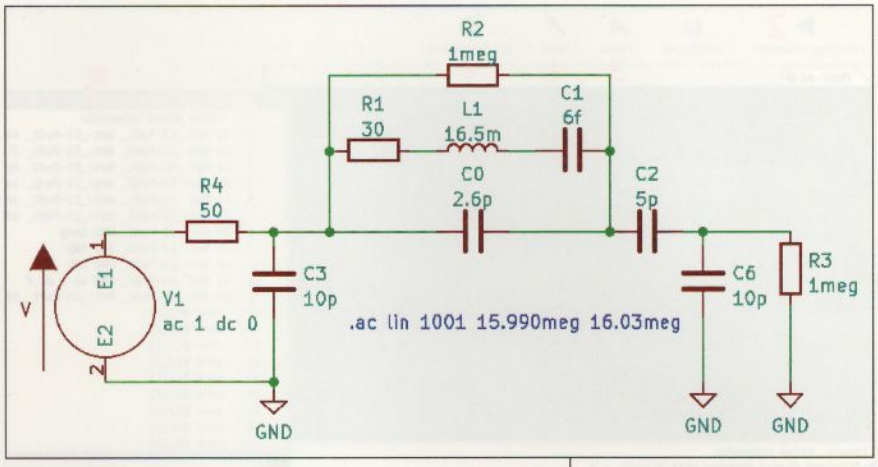


Figure 24 : Simulation d'un modèle de résonateur à quartz de type Butterworth-van Dyke RLC//C dans un circuit d'oscillation comprenant les deux capacités de pieds C_3 et C_6 et un condensateur de tirage en série avec le résonateur C_2. La résistance R_2 est incluse, comme souvent dans le vrai oscillateur, pour aider SPICE à converger, mais son utilisation pratique est la polarisation de la porte logique.

Nous dessinons le circuit RLC série avec sa capacité parallèle – dit modèle de Butterworh-van Dyke – avec les valeurs de composants qui permettent de simuler ici un résonateur à 16 MHz. Nous ajoutons la capacité C_2 en série et les deux capacités de pieds C_3 et C_6 pour tenter d'atteindre la condition de Barkhausen en phase d'une rotation globale de 180°. Afin de mesurer une fonction de transfert S_{21}, nous excitons un côté avec une source de tension en série avec une impédance limitant le courant que peut fournir la porte logique, et chargeons par une impédance représentative de l'impédance en entrée de la porte logique. La source de tension s'obtient dans les composants KiCad en effectuant une recherche sur le mot clé « spice » qui propose une liste de symboles non routables mais utiles à la simulation (Fig. 24). Les paramètres de la source sont fournis dans son champ ***Value*** avec une valeur de tension DC nulle et une amplitude de la composante alternative unitaire. La nature de la

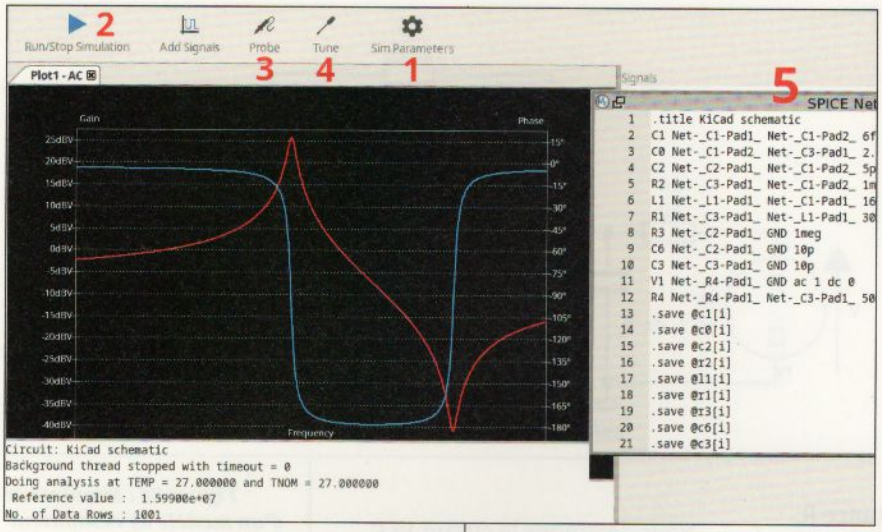
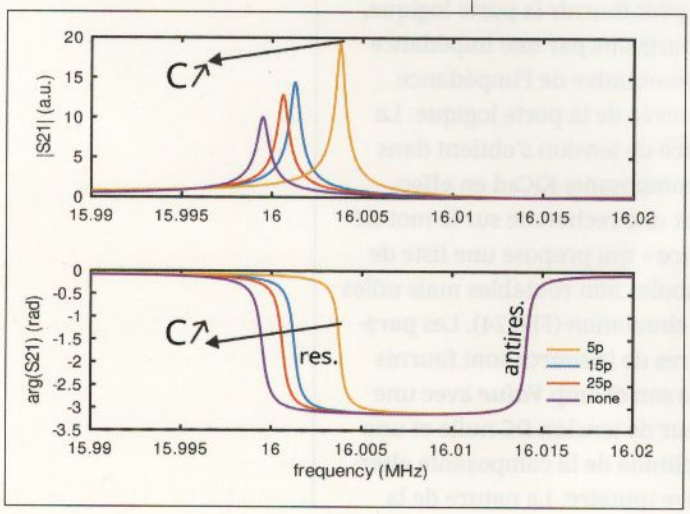


Figure 25 : Résultat de la simulation après avoir chargé les paramètres depuis le texte libre dans le circuit (1), lancé la simulation (2), choisi le signal à sonder (3) au-dessus de R_3, et potentiellement choisi de faire varier C_2 avec le tournevis (4).

Figure 26 : Résultats de la simulation pour diverses valeurs de C_2, avec en haut le module de la tension et en bas sa phase, à gauche la résonance sensible à la valeur de C_2 et à droite l'anti-résonance insensible à C_2. La largeur de la résonance déterminée par R_1 fournit le facteur de qualité du résonateur.



simulation est fournie comme texte libre dans le schéma comme nous le ferions avec `ngspice`, à savoir une fonction de transfert spectrale avec un axe des fréquences qui s'incrémente linéairement autour de la résonance à 16 MHz : `.ac lin 1001 15.99meg 16.02meg` en se rappelant que sous SPICE, « `m` » comme « `M` » représentent le milli (10^{-3}) tandis que le million (10^6) se nomme « meg »... c'est aussi valable pour les résistances.

Le schéma rempli, sous KiCad 7 nous sélectionnons le menu ***Inspect → Simulator...*** et une fois la fenêtre de `ngspice` lancée, nous sélectionnons ***Sim Parameters → Custom → Load directives from schematic***, nous lançons la simulation par la flèche, et sélectionnons les signaux à afficher avec la sonde ***Probe***. Nous pouvons faire varier un composant, par exemple *C2*, en plaçant le tournevis ***Tune*** sur ce composant (Fig. 25).

Cependant, la sortie d'écran de `ngspice` est hideuse, ne permet pas d'accumuler des graphiques ou d'automatiser la simulation : nous revenons donc aux fondamentaux en exportant la *netlist* par ***Simulation → Show SPICE netlist*** que nous sauvons (Fig. 25, étape 5) dans un fichier texte (par convention d'extension `.cir` mais sans importance), et sous `ngspice` dans un terminal `source fichier.cir` suivi de la commande de simulation `ac lin ...` permet de relancer la simulation dont le résultat est sauvé dans un fichier `sauve` par `print V(Net-(C2-Pad1)) > sauve` pour être traité avec son outil favori (GNU Octave ou `gnuplot` par exemple). Le résultat est fourni en Fig. 26, avec la condition asymptotique de l'absence de condensateur en série pour la fréquence de résonance la plus basse, un tirage maximal pour la valeur de condensateur la plus faible, et une évolution continue entre les deux, puisque l'absence de condensateur revient au cas de la capacité infinie ($|Z| = 1/(C\omega) \to 0$ si $C \to \infty$).

Nous comprenons ici pourquoi on dit qu'un oscillateur peut être tiré entre la résonance – minimum d'impédance – et l'anti-résonance, maximum d'impédance que nous constatons être insensible à la capacité série (mais très sensible à toute capacité parallèle, puisque produite par C_0) et que donc les résonateurs en matériau piézoélectrique faiblement couplé tel que le quartz ne permettent qu'une correction modeste de la fréquence d'oscillation.

B. MODÉLISATION DU RÉSONATEUR À QUARTZ DANS QUCS

Modéliser un résonateur sous SPICE impose de fournir un circuit équivalent en composants discrets R, L, C qui peut être plus ou moins exact en fonction des éléments parasites qui entourent le composant, par exemple fils de connectique ou capacités parasites du boîtier. Il est classique en mesure de dispositifs radiofréquences de caractériser la réponse spectrale en module et en phase : un instrument dédié à cette mesure est l'analyseur de réseau vectoriel qui mesure les caractéristiques de l'onde réfléchie (et transmise le cas échéant) en fonction de la fréquence définie par l'utilisateur **[19]**. De cette mesure de la fonction de transfert du dispositif, nous devons pouvoir déduire l'impact des éléments entourant le dispositif analysé, par exemple les condensateurs de pieds dans le montage Pierce ou la capacité série de tirage.

Le format standard d'une mesure par analyseur de réseau vectoriel se nomme le format Touchstone, ou SnP lors de la mesure de *n* ports. Ce format commence avec un entête qui annonce la nature de la mesure, l'impédance de référence, puis autant de colonnes que nécessaire pour contenir d'abord la fréquence, puis les coefficients complexes de réflexion et éventuellement de transmission de l'onde. Pour un résonateur qui ne contient qu'un port (deux broches), seul le coefficient de réflexion importe et la mesure de S_{11} ne contient que trois colonnes, la fréquence suivie de la partie réelle et de la partie imaginaire du coefficient de réflexion. Notre fichier de mesure est disponible à **http://jmfriedt.free.fr/resonateur54MHz.s1p**.

Cependant, SPICE ne sait pas s'accommoder d'un fichier SnP, il nous faut un autre outil pour analyser un tel fichier. L'outil propriétaire classique de Agilent (maintenant Keysight) se nomme ADS pour *Advanced Design System*. Une tentative de version libre de cet outil se nomme Qucs, *Quite Universal Circuit Simulator*. Bien que nombre de fonctionnalités d'ADS manquent dans Qucs, ce dernier sera suffisant pour ce qui nous importe. Malheureusement, Qucs a bien du mal à rester en vie, et tenter de compiler depuis les sources sur une distribution Debian/sid actuelle se solde par un logiciel qui crashe à la moindre sélection de menus. Nous avons dû nous résoudre à utiliser une distribution binaire telle que décrite dans **https://snapcraft.io/qucs-spice**. Une fois ce paquet installé, nous lançons `/snap/qucs-spice/qucs-spice.qucs` pour obtenir une fenêtre de la forme de celle proposée en Fig. 27 (gauche). Nous créons un nouveau projet, choisissons dans le menu ***Components*** à gauche les ***lumped components*** pour une résistance et trois condensateurs, et sous ***file components*** un bloc ***1-port S parameter file***. Ce dernier bloc est renseigné du nom et chemin du fichier S1P et le menu ***Edit*** permet de vérifier que le fichier a été convenablement interprété. Finalement, nous ajoutons la nature de la simulation sous forme de ***simulations → S-parameter***, simulation que nous paramétrons avec la gamme de fréquences de la mesure S1P et le même nombre de points que la mesure – dans notre cas 53,98 MHz à 54,01 MHz en 1201 points.

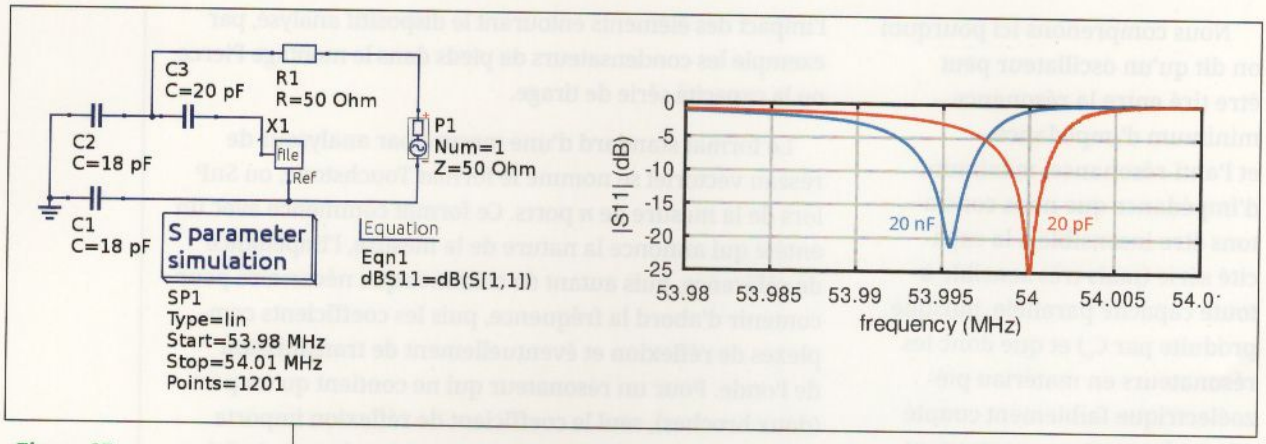


Figure 27 : Gauche : schéma de la modélisation par Qucs du tirage capacitif du résonateur dont la réponse a été mesurée et sauvegardée en fichier S1P. Droite : résultat de la simulation pour une capacité de tirage en série avec le résonateur de 20 pF et de 20 nF s'apparentant à une capacité infinie et fournissant donc presque la réponse libre du résonateur.

Il est inutile, voire faux, d'étendre l'analyse au-delà de la gamme de mesure du fichier S1P puisque Qucs n'aura aucune information sur le comportement du composant en dehors de sa gamme de caractérisation. La position du port de mesure est ajoutée par ***sources → power sources*** qui indique l'impédance caractéristique par rapport à laquelle le coefficient de réflexion est calculé.

Enfin, nous définissons la nature de l'affichage, à savoir ***diagrams → Cartesian***. Dans la fenêtre nouvellement créée, double-cliquer sur le cadre permet de sélectionner la nature de l'affichage. Comme nous avons défini (***Insert → Insert Equation***) un affichage en décibels par `dBS11=dB(S(1,1))`, nous avons l'option dans le graphique de double-cliquer sur *dBS11* en fonction de `frequency` pour afficher le résultat de la simulation produite par F2 ou ***Simulation → Simulate***.

Puisque nous désirons appréhender l'impact de varier le condensateur en série C3 et que comme d'habitude, un outil de simulation n'a pas pour rôle de produire une sortie graphique esthétique, nous désirons sauver le résultat de la simulation pour l'importer dans GNU Octave ou `gnuplot`. La seule façon que nous avons trouvée de conserver trace de la simulation est de copier `$HOME/snap/qucs-spice/common/.qucs/projet/projet.dat` et de l'éditer pour le rendre compatible avec Octave. Le résultat de la Fig. 27 est en parfait accord avec la modélisation SPICE proposée auparavant, mais en s'appuyant cette fois sur un vrai fichier de mesures et non un ajustement par un modèle s'approchant du comportement du résonateur.

La nouvelle mouture de Qucs mise à jour à **https://github.com/ra3xdh/qucs_s** ne supporte pas nativement le chargement des fichiers au format Touchstone, mais vise à s'appuyer sur le convertisseur vers un modèle SPICE par **https://github.com/transmitterdan/s2spice** qui une fois fonctionnel affranchira totalement de la dépendance à Qucs pour fournir une solution exclusivement `ngspice` une fois que les sources de tension et de courant dépendantes de la fréquence auront fini d'être implémentées. **JMF**

RÉFÉRENCES

[1] Oleg Obleukhov, Ahmad Byagowi, *How Precision Time Protocol is being deployed at Meta* à : **https://engineering.fb.com/2022/11/21/production-engineering/precision-time-protocol-at-meta/**

[2] Ahmad Byagowi, Oleg Obleukhov, *PTP: Timing accuracy and precision for the future of computing* à : **https://engineering.fb.com/2022/11/21/production-engineering/future-computing-ptp/**

[3] P. Boven, *Synchronization for interferometry through White Rabbit*, European GNU Radio Days (2023) à : **https://www.youtube.com/watch?v=-rt0mQtxn64** à 7'35" formalisé dans A.R. Thompson & al., *Interferometry and Synthesis in Radio Astronomy*, Springer Open (2017) disponible à : **https://link.springer.com/book/10.1007/978-3-319-44431-4** (pp. 434–448).

[4] *Time-stamping and business clocks synchronisation under MiFID II* indique que les horloges datant les transactions rapides doivent diverger du temps universel coordonné UTC de 100 µs au plus et horodater à la microseconde à : **https://emissions-euets.com/time-stamping-and-business-clocks-synchronisation**

[5] Jeff Geerling, *PTP and IEEE-1588 hardware timestamping on the Raspberry Pi CM4* à : **https://www.jeffgeerling.com/blog/2022/ptp-and-ieee-1588-hardware-timestamping-on-raspberry-pi-cm4**

[6] D.L. Mills, *On the chronometry and metrology of computer network timescales and their application to the Network Time Protocol*, Proc. ACM SIGCOMM Computer Communication Review 21(5) 8–17 (1991) à : **https://www.eecis.udel.edu/~mills/database/papers/time.pdf**

[7] G. Goavec-Merou, J.-M Friedt, F. Meyer, *Leurrage du GPS par radio logicielle, MISC Hors-Série n°19*, févr. 2019 : **https://connect.ed-diamond.com/MISC/mischs-019/leurrage-du-gps-par-radio-logicielle**

[8] Leapsecond, *Motorola GPS M12+ Sawtooth* à : **http://www.leapsecond.com/pages/m12/sawtooth.htm**

[9] *Security Requirements of Time Protocols in Packet Switched Networks* (2014) à : **https://www.rfc-editor.org/rfc/rfc7384.txt**

[10] A. Malhotra & al., *The Security of NTP's Datagram Protocol*, Cryptology ePrint Archive, Paper 2016/1006 (2016) à : **https://eprint.iacr.org/2016/1006**

[11] Commit **https://ohwr.org/project/white-rabbit/commit/eee7859a331d3b15811cac6e0ef06be26dd63255** de J. Serrano indiquant « *Delegating support to WR commercial providers was part of our scalability strategy but did not really work ; Some WR developers at CERN moved on to other projects ; This combined with the increasing uptake of the technology in many areas raises a sustainability issue.* », 26 juillet 2023.

[12] Geyer Quartz Technologies, *Short Tutorial on Quartz Crystals and Oscillators* (2022) à : **https://www.geyer-electronic.de/wp-content/uploads/2022/11/GEYER-Quarz-Tutorial_e_07_22_V1.0.pdf**

[13] un résonateur à onde de volume tel que ceux qui cadencent les systèmes numériques synchrones est formé d'un substrat de matériau piézoélectrique – généralement le quartz – qui convertit le signal électrique en onde mécanique. Comme dans un interféromètre optique de Fabry-Pérot, l'onde élastique est confinée dans le morceau de quartz d'épaisseur e et la condition de résonance à longueur d'onde λ vérifie $e = N \cdot \lambda/2$, $N \in \mathbb{N}$ impair. Pour un mode fondamental tel que le résonateur 54 MHz qui équipe la CM4, $N = 1$ et $\lambda = c/f$ avec c la célérité de l'onde élastique de l'ordre de 5000 m/s et f=54 MHz indique que $e \simeq 44$ µm. Puisque c dépend de la température ou la contrainte, à e fixe nous observerons f varier.

[14] 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**

[15] James Clark, *Guide to using the hardware PTP support in the Raspberry Pi CM4* (juillet 2023) à : **https://github.com/jclark/rpi-cm4-ptp-guide/**

[16] *GPS Grandmaster with ARM embedded Linux and SNMP* (2022) à : **https://www.embien.com/blog/gps-grandmaster-with-arm-embedded-linux/**

[17] Documentation pigpio et en particulier la section sur les PWM matérielles à : **http://abyz.me.uk/rpi/pigpio/python.html#hardware_PWM**

[18] J. Vig, *Quartz Crystal Resonators and Oscillators – For Frequency Control and Timing Applications – A Tutorial* (2014) à : **https://www.researchgate.net/publication/272177403_Quartz_Crystal_Resonators_and_Oscillators_-_For_Frequency_Control_and_Timing_Applications_-_A_Tutorial**

[19] J.-M. Friedt, *Introduction à l'analyseur de réseau : le NanoVNA pour la caractérisation spectrale de dispositifs radiofréquences*, *Hackable 36* (2021) - **https://connect.ed-diamond.com/Hackable/hk-036/introduction-a-l-analyseur-de-reseau-le-nanovna-pour-la-caracterisation-spectrale-de-dispositifs-radiofrequences**

L'ÈRE DES TRANSISTORS AU GERMANIUM

[Yann Guidon – yg@ygdes.com]

Dans ma quête de « désinformatisation » [1], après avoir touché le fond en jouant avec des relais [2] (et sans en être remonté encore), j'ai sauté l'étape thermoïonique pour m'intéresser aux premiers transistors. Je me posais surtout cette question bête : comment nos aïeux ont-ils bien pu se débrouiller avec une technologie si capricieuse ?

Et quand je dis « capricieuse », je suis généreux, car comment qualifier poliment un composant dont les caractéristiques changent significativement durant leurs mesures, rien que par la chaleur des doigts qui tiennent les pattes, ou simplement l'échauffement minime dû au courant qu'ils commutent ? Ce n'est pas un thermistor que je veux mais bien un transistor.

Encore une fois, il s'agit bien d'une saine curiosité pour combler un manque culturel et technique, et non d'un fétichisme quelconque qui partirait d'une croyance que « le germanium, c'est mieux ». Et puis me pencher sur les limitations des relais m'a bien fait découvrir les ABCE (*Arbres Binaires à Commande Équilibrée* **[3]**), alors qui sait ce qui pourrait émerger de l'exploration d'une autre technologie surannée et exotique...

Je sais pertinemment que les composants au silicium surclassent largement leurs prédécesseurs en termes de bruit, de courants de fuite, de stabilité en température, de fiabilité, d'homogénéité des caractéristiques... Nous avons beaucoup de chance de bénéficier aujourd'hui des développements industriels du siècle dernier ! Donc, comment en sommes-nous arrivés à notre actuelle abondance de qualité ?

Oui, cela fait beaucoup de questions, commençons donc par le début.

1. UN HÉRITAGE DÉJÀ OUBLIÉ

Les composants antiques au germanium ont cette réputation de fuir comme un SR71 au sol, de l'ordre de 20 à 200 µA selon la tension, la température et la phase de la lune, et le résumé de Wikipédia sur les applications électroniques du germanium **[4]** ne les rend pas très excitantes :

L'effet transistor a été observé en 1948 (sic) *dans du germanium. Il a servi de substrat semi-conducteur jusqu'à ce que le silicium prenne sa place, vers les années 1970. Des transistors au germanium sont encore employés dans les années 2020 comme composants principaux de certaines pédales d'effet pour guitare électrique, en particulier les fuzz, pour leur sonorité supposée particulière et qui serait appréciée des amateurs de sons « années 1960 ». Aujourd'hui, il est plus utilisé dans le domaine des hautes fréquences, pour la réalisation de diodes à faible chute (0,3 V environ, application en détection) du poste à diode et dans les cellules photovoltaïques multijonction pour utilisations spatiale et terrestre après concentration. On le trouve également à l'état d'alliage ou de multicouches avec le silicium (SiGe).*

Je ressens une certaine frustration lorsque je vois les transistors au germanium réduits aujourd'hui à des « composants miraculeux pour *shoegazers* » même s'il n'y a aucun mal à fabriquer ses propres pédales d'effets, bien au contraire. Mais en prenant une voie similaire à celle des audiophiles, teintée de subjectivité et de mysticisme, je constate aussi que cela a fait exploser les prix de ces composants qui n'ont rien de magique, et qui peuvent être émulés de façon bien plus fiable par un plug-in VST. Mais passons ce faux débat, puisqu'il y a bien plus intéressant à explorer.

Car dès que l'on se penche un peu dessus, en suivant les innombrables ressources en ligne, l'importance du germanium prend une tout autre dimension scientifique, industrielle, historique et même politique ! Justement, cet élément est l'une des dernières victimes de la guerre économique

entre les USA et la Chine, avec le bannissement des exportations du gallium et du germanium par cette dernière, qui en est aujourd'hui le plus gros producteur (c'est un résidu de la purification de l'aluminium, du cuivre, du zinc ou de l'argent **[5]**).

Si nous vivons aujourd'hui dans l'ère du silicium avec ses avantages, son abondance, et toute une industrie derrière, c'est parce que nous sommes passés par l'étape du germanium. Mais d'ailleurs, pourquoi utiliser cet élément ? En fait, ce n'est pas totalement clair, mais plusieurs facteurs ont joué en sa faveur.

2. UN PEU D'HISTOIRE

La théorie des semi-conducteurs est née au 19e siècle et a accompagné l'essor de la physique quantique et de la chimie industrielle. Karl Ferdinand Braun a découvert en 1874 qu'un contact entre un métal et un minerai cristallin laissait passer un courant dans une direction privilégiée **[6]**, ce qui est bien plus pratique que la diode construite avec un tube à vide (inventée en 1904 par Fleming).

Des diodes de puissance à l'oxyde de cuivre ou au sélénium supportaient de forts courants dès 1920 **[7]**, permettant ainsi une alimentation continue à partir de courant alternatif, bien qu'il y ait de fortes pertes et une tension inverse si faible qu'il fallait empiler de nombreuses jonctions. C'était inefficace, lourd et encombrant (et l'odeur était nauséabonde lors d'une défaillance), mais relativement simple et cela avait le mérite d'exister. C'était même indispensable pour générer les hautes tensions requises par les appareils utilisant les tubes à vide de l'époque.

De l'autre côté du spectre, les pionniers de la radio avaient besoin de détecteurs extrêmement sensibles, le premier brevet pour cela fut déposé en 1901 par Jagadish Chandra Bose. Cela a conduit aux fameux « postes à galène », dont l'élément détecteur est une fine pointe métallique, appelée « cat whisker » (en français : *moustache de chat*) délicatement posée à la surface d'un morceau de sulfure de plomb, le nom chimique de cette fameuse galène ; le carbure de silicium et la pyrite de fer étaient aussi utilisés **[8a]**. Les irrégularités de ces matériaux naturels rendaient le montage difficile à régler, éphémère, sensible aux vibrations et à l'humidité. Leur fonctionnement mystérieux fut élucidé en 1938 par le physicien Walter H. Schottky,

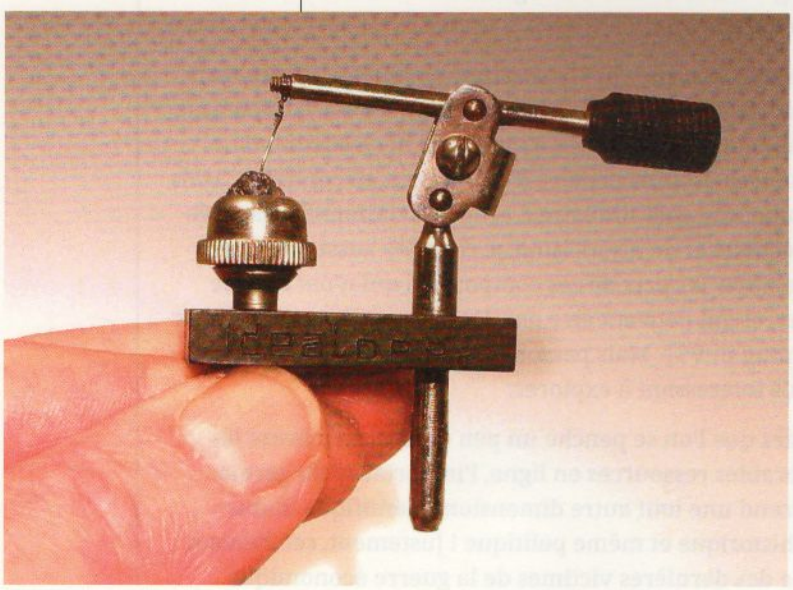


Figure 1 : Une « moustache de chat » (une pointe métallique) appuie sur un minerai (sulfure de plomb, pyrite de fer, carbure de silicium...) pour créer une diode rudimentaire, mais suffisante pour capter les émissions radiophoniques il y a un siècle. Source : Holger. Ellgaard@Wikimedia (2008, CC-SA3), consulter en particulier https://en.wikipedia.org/wiki/Crystal_detector qui est très instructif !

donnant ainsi son nom aux diodes formées par une jonction entre un métal et un semi-conducteur.

Grâce à leur simplicité et leur faible coût, les postes à galène furent très répandus pour capter les émissions en modulation d'amplitude de l'industrie radiophonique naissante (particulièrement en temps de guerre **[8b]**). Ils reposent sur une diode très sensible et fragile, incompatible avec un appareil mobile, par exemple. Les livres de H. Peter Friedrichs montrent qu'ils étaient très faciles à fabriquer par soi-même, avec des éléments simples et beaucoup de soin **[9] [10]**.

Les efforts de la Seconde Guerre mondiale (en particulier pour les radars qui nécessitaient de travailler dans les bandes centimétriques) ont conduit à des recherches plus poussées aux USA et en Allemagne, où la purification du matériau semi-conducteur s'est révélée critique. De plus, grâce à l'encapsulation de ces détecteurs dans des cartouches, où le point de contact est préréglé en usine et immobilisé, les diodes sont aussi protégées de la lumière et de l'humidité (qui oxyde et détériore progressivement le composant).

Lors de la fabrication des diodes, de nombreux paramètres affectent la sensibilité, la répétabilité, les rendements de production donc le coût des diodes industrialisées, en particulier :

- La purification : le matériau doit être purifié à l'extrême, à moins d'un atome d'impureté par milliard, soit un intrus dans un cube de mille atomes de côté. La quantité de matériaux nécessaire implique la création de filières industrielles spécialisées, donc toute infrastructure qui existe déjà est réutilisée autant que possible, et la qualité est optimisée progressivement à partir de la Seconde Guerre mondiale. On voit apparaître des méthodes comme le « zone melting » ou la vapeur de tétrachlorure de silicium (la plus utilisée actuellement).
- La cristallisation : le cristal doit être extrêmement homogène, donc le matériau est idéalement monocristallin et orienté selon un angle très précis. Le procédé de Czochralski **[11]** est la principale méthode utilisée depuis un siècle, tous les paramètres (vitesses de rotation et de tirage, températures à tous les niveaux) doivent être finement contrôlés, mais ce fut un procédé manuel encore dans les années 1970.
- Le dopage : l'implantation contrôlée d'impuretés spécifiques (de polarité N ou P) permet au semi-conducteur de devenir conducteur (selon les conditions désirées) et c'est encore un défi scientifique et industriel, une sauce secrète mise au point par chaque fabricant, selon les outils et matériaux dont ils disposent.

Les contraintes de sensibilité et le point de fusion du germanium plus faible (environ 938 °C) ont d'abord favorisé ce dernier, alors

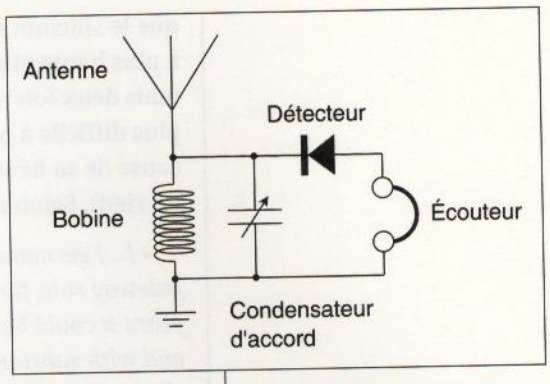


Figure 2 : Un récepteur radiophonique AM minimaliste est très simple à réaliser avec les moyens du siècle dernier. Il suffit de fils (pour l'antenne et la bobine), de plaques métalliques parallèles (pour le condensateur), d'un casque à très haute impédance et d'une diode. La capacité d'accord règle la fréquence de résonance du circuit formé avec la bobine. Le symbole actuel de la diode est une stylisation de la pointe métallique (cat whisker) posée sur un cristal (la barre perpendiculaire). D'autres topologies et de nombreux raffinements existent pour augmenter la sélectivité ou la sensibilité.

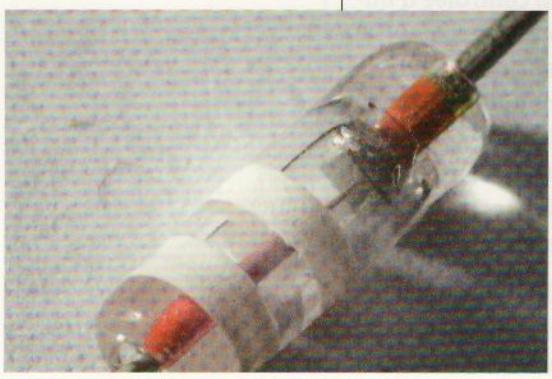


Figure 3 : Les diodes encapsulées dans du verre permettent d'observer la jonction entre le métal et le semi-conducteur. Ici, une diode russe Д9К laisse apercevoir la pointe métallique, tordue pour que le ressort appuie sur le fin carré de germanium. Ce dernier repose sur un point de soudure (brillant) faisant la liaison mécanique et électrique avec la cathode. La surface de contact microscopique explique pourquoi le courant est limité à 20 mA environ. Au-delà, la densité de courant au niveau de la pointe augmenterait dangereusement la température et endommagerait la jonction.

que le silicium était connu pour fonctionner à plus haute vitesse et être moins bruyant, mais deux fois moins sensible et beaucoup plus difficile à purifier (1410 °C) et à usiner à cause de sa haute réactivité (il s'oxyde pour un rien). Selon IEEE Spectrum **[12]** :

« [...] germanium played the crucial gateway role, for in the immediate postwar years it could be refined much more easily and with substantially higher purities than silicon. Such high-purity semiconductor material was absolutely essential for fabricating the first transistors. »

Pour fixer définitivement la pointe métallique à la surface du semi-conducteur, en plus d'utiliser un ressort pour la pointe, la méthode de « formage » a été développée : de brèves impulsions avec une tension et un courant soigneusement limités vont chauffer et fondre localement le semi-conducteur, faisant migrer des atomes de l'aiguille dans le cristal, et ainsi créer une jonction mieux contrôlée, en plus de stabiliser mécaniquement l'ensemble. Cela donne les diodes de type « point contact » (en anglais), que l'on peut retrouver sur certains vieux modèles (G118D, OA154, Д9Д, 1N34...). Les premiers supports en céramique ressemblaient à des fusibles, mais lorsque la diode est encapsulée dans une petite ampoule en verre, nous pouvons observer une telle jonction (figure 3), même s'il est plus facile d'examiner une LED moderne.

La grande sensibilité du germanium vient de sa faible tension de seuil, qui est d'environ 0,2V selon le courant, parfois 0,1V pour les plus sensibles ou lorsque la température augmente. Cette tension est liée à la faible énergie requise par les électrons pour passer de la bande de valence (la couche isolante de l'atome) à la bande de conduction (où les électrons sont libres de passer d'un atome au voisin). Cet écart s'appelle le « bandgap », il est affecté par les impuretés (contrôlées par dopage) et cet aspect est essentiel pour la suite. Cette faible énergie qui sépare l'état isolant de l'état conducteur s'accompagne d'un bruit élevé : les électrons agités par le bruit thermique (des microscopiques vibrations du cristal dépendant de la température) vont plus facilement sauter d'un niveau à l'autre et perturber le courant ou fuiter là où le champ électrique les attire.

3. UN PEU PLUS D'HISTOIRE

Durant la Deuxième Guerre mondiale, des ingénieurs allemands essaient de réduire ce bruit intrinsèque, qui noie les signaux utiles dans les récepteurs de radars : ils créent une diode différentielle appelée *duodiode* en supposant que le signal serait plus propre si l'on place deux électrodes le plus près possible l'une de l'autre (quelques dizaines de microns) et si l'on n'en garde

que la valeur moyenne. Contre toute attente, en variant la distance entre les électrodes, ils observent un phénomène d'« interférence », mais ils n'ont pas le temps de reproduire et étudier ces effets, alors que les armées alliées s'approchent de leurs laboratoires.

Après la guerre, c'est le début de la grande saga de la réelle paternité du transistor, qui avait été déjà envisagé et breveté (sous une autre forme) dès 1925 par Julius Edgar Lilienfeld **[13]**. Les Alliés vainqueurs s'arrachent les recherches, les équipements et les ingénieurs du grand vaincu, en particulier l'Union soviétique, qui opère déjà des réseaux d'espionnage tous azimuts et infiltre aussi les États-Unis (et pas juste le projet Manhattan) **[14]**. Le bloc soviétique revendique beaucoup de découvertes, mais il est impossible de trier avec certitude entre les recherches effectives de ses talentueux (et contraints) chercheurs, les fruits de l'espionnage, le secret paranoïaque (qui a masqué et étouffé beaucoup d'innovations soviétiques) ou simplement la propagande nationaliste exacerbée stalinienne. Il semble en tout cas que la production de semiconducteurs soit industrialisée en Union soviétique dès 1947 grâce à la récupération d'une ligne de production allemande dédiée aux détecteurs au germanium destinés aux radars **[15]**.

Parallèlement, à partir de 1945 à la « Compagnie des Freins et Signaux Westinghouse » à Aulnay-sous-Bois (dans le 93), d'autres ingénieurs d'origine allemande sont embauchés pour créer une ligne de fabrication de diodes, ce qui leur permet de poursuivre leurs recherches sur les interférences. Début 1948, Herbert Mataré **[16]** **[17]** et Heinrich Welker ont observé et caractérisé l'amplification avec leur propre diode à trois broches (voir la figure 4), puis développé et breveté le « Transistron » (déposé le 13 août 1948). Mais quelques mois auparavant, en décembre 1947, le fameux trio Bardeen, Brattain & Shockley **[18]** avait déjà découvert et fait fonctionner un démonstrateur d'amplificateur à semi-conducteur, en se reposant sur les recherches secrètes sur les diodes initiées aux Bell Labs, au MIT et à Purdue University pendant la guerre. Ils cherchent en vain depuis une décennie à créer un transistor à effet de champ (inspiré par Lilienfeld), donc ils partent sur d'autres principes que les Allemands, avec une meilleure base scientifique, mais ils arrivent par hasard à une structure similaire : deux électrodes posées l'une près de l'autre sur un cristal de germanium purifié. C'est surtout l'antériorité de leur prototype fonctionnel, étayé par leurs travaux théoriques, qui leur vaut le Prix Nobel en 1956.

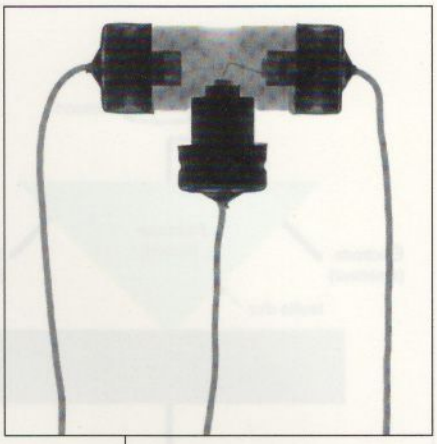


Figure 4 : Radiographie d'un Transistron (transistor français) où l'on distingue bien la structure typique d'une diode avec deux électrodes presque symétriques (il faut distinguer le collecteur de l'émetteur). Source : Andrew Wylie, http://www.wylie.org.uk/technology/semics/westcrel/westcrel.htm.

Ce sont ces travaux des laboratoires Bell Labs, avec la structure en biseau coupé **[19]**, qui ont donné le symbole qui nous est aujourd'hui si familier (voir la figure 5) : la base est représentée alors par le substrat de germanium purifié et dopé, et deux électrodes convergent pour faire entrer et sortir le courant dans le sens indiqué par la flèche. Ensuite,

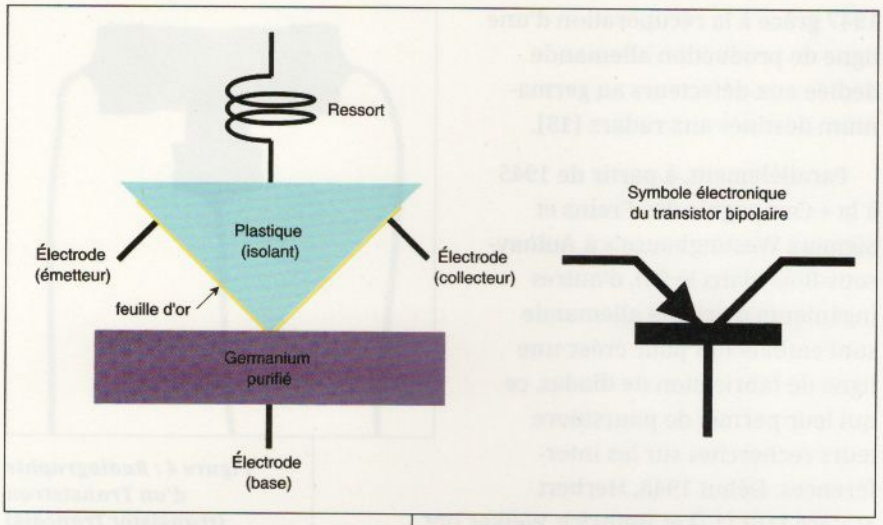


Figure 5 : Pour créer un transistor à la façon Bardeen & Brattain, c'est très simple, il vous faut une surface anodisée de germanium purifié, électriquement reliée à une électrode appelée la base, ainsi qu'un isolant taillé en biseau recouvert d'une fine feuille d'or. Cette feuille est délicatement coupée à l'endroit de la pointe pour créer une très fine séparation (environ 40 µm, la moitié d'un cheveu), et les deux électrodes ainsi formées sont appelées émetteur et collecteur (en fonction du sens du courant qui les traverse). Lorsque l'ensemble du biseau est appuyé contre la base en germanium, cela peut créer un transistor à contact, atteignant un gain de tension de 18 en 1948. Cette structure se retrouve dans le symbole (à droite) adopté depuis, bien que les structures et les performances aient considérablement changé depuis.

avec l'arrivée du transistor à jonction (développé dans la foulée par Shockley), d'autres symboles apparaissent (voir la figure 12), mais l'original s'est déjà imposé.

> Bien que le germanium puisse être dopé facilement, aussi bien positivement que négativement, la plupart des transistors en germanium sont de type PNP, et je n'ai pas encore bien compris pourquoi. Les transistors au silicium sont surtout NPN, car les PNP sont plus lents et/ou moins conducteurs, donc plus chers au final à mettre en œuvre. Il en va de même pour les MOSFET canal N, préférés au canal P.

Le brevet de Bell Labs s'est focalisé sur l'appareil que Bardeen & Brattain ont mis au point alors que Shockley était occupé ailleurs, ce qui a rendu furieux ce dernier. D'autres facteurs ont compliqué la situation, comme l'existence du brevet européen de Westinghouse, déposé juste après l'annonce de Bell Labs du 30 juin 1948, ainsi que l'absence de mention de l'antériorité de Lilienfeld par Schockley. Mais ce n'est que le début du développement de nouvelles générations de structures, plus efficaces, plus performantes, moins chères, plus faciles à fabriquer. Dès 1948, frustré de ne pas avoir la seule paternité du transistor, Shockley invente le transistor à jonction « tirée », reposant sur un procédé de fabrication très différent et capitalisant sur les avancées théoriques et industrielles les plus récentes.

Du côté français, malgré les limites du pays en pleine reconstruction, le Transistron Westinghouse (commercialisé sous le nom *Westcrel*) commence à être utilisé pour amplifier les liaisons téléphoniques. Dès 1950, la haute bande passante du Westcrel permet de relier la France à l'Algérie, ce qui aiguillonne encore plus Shockley. Le premier lot de 1000 unités de Transistrons reste un secret d'État et les composants en forme de T ne se trouvent plus qu'en rares exemplaires dans des musées (aux Arts et Métiers **[20]** ou au Deutsches Museum de Munich **[21]**). On voit apparaître un récepteur radio à

base de 4 Transistrons lors d'une exposition en 1953 (en même temps qu'un récepteur à transistors Intermetall), mais malgré sa vitesse supérieure au prototype d'AT&T, le *Westcrel* est très cher et fragile. Il faudra encore attendre un an pour que le transistor apparaisse dans les poches du public, grâce à d'autres fabricants.

Le CNET (récemment fondé pour assister les PTT) poursuit les recherches **[22]** pour moderniser le réseau téléphonique (comme AT&T aux USA), car le nouveau composant économise beaucoup de place et d'énergie au niveau des répéteurs. Avec le temps, il n'est plus très clair si le mot « Transistron » désigne le mot francisé du transistor (terme qui finit par s'imposer en quelques années) ou s'il s'agit du circuit à contact inventé par Mataré & Welker. Freiné par le Gouvernement dans les années 50, dépassé par les évolutions qu'il n'a ni les moyens ni la prérogative de poursuivre seul, Westinghouse ferme son laboratoire de recherche et d'autres transistors français sont fabriqués par Clarville (CSF) ou le Laboratoire Central de Télécommunications (LCT) créé par le CNET pour ses propres besoins. Radiotechnique s'essaie aussi aux semi-conducteurs, mais ils ne font que suivre, laissant de côté dès 1955 l'innovation de Mataré pour des techniques plus robustes et performantes : la voix c'est bien, la radio c'est mieux !

En 1951, en raison d'accords avec le gouvernement américain pour limiter le monopole de sa maison-mère AT&T **[23]**, sa filiale industrielle Western Electric vend des licences abordables de ses méthodes à d'autres fabricants (25 000 USD pour tout le monde), même à l'étranger (dont le Japon après quelques hésitations **[24] [25]**). Puisque seul le brevet est licencié, les secrets de production doivent être réinventés par chacun. De plus en plus de fabricants américains, petits et grands, saisissent l'occasion, comme Raytheon qui a développé des aides auditives enfin abordables **[26]**, à faible consommation et compactes (par rapport aux encombrants systèmes à tubes). Le transistor à contact original est en sursis.

Cela a encore plus stimulé la compétition, l'évolution, la diversité, les techniques de production, la qualité, les performances, la réduction des coûts... De son côté, Mataré est retourné en Allemagne en 1951 pour y fonder la société Intermetall, concevant et industrialisant ses propres générations de diodes et transistors. Le Royaume-Uni tient bon avec le fabricant Mullard (déjà succursale du Hollandais Philips), puisque les semi-conducteurs sont une extension logique de sa fabrication de tubes. Les années 50 voient une explosion technologique incroyable où chaque nouvelle génération rend obsolètes celles introduites seulement quelques années avant.

Pendant ce temps-là, dans le bloc communiste, c'est *compliqué*. Ce qui suit n'est qu'un résumé des quelques informations que j'ai pu récolter et (comme le reste) ne constitue pas une référence absolue. Les ressources russes sont quasiment toutes écrites en russe (*comme par hasard !*) et certaines comme **https://chipprof.ru/** sont tout simplement bloquées hors de Russie (nous en revenons encore à la politique).

Dans l'URSS d'après-guerre, le secret extrême entre les nombreux laboratoires a mené à la duplication des recherches indépendantes (comme partout, en fait), donc toute attribution est difficile, mais ils ont reproduit la *duodiode* allemande presque en même temps que Westinghouse, certainement grâce aux prisonniers de guerre qui l'avaient imaginée. La paranoïa étatique, la fragilité et les faibles performances (par rapport aux tubes) et les malentendus ont étouffé les transistors au niveau politique, retardant l'industrialisation durant

Figure 6 : Quelques transistors russes « classiques » de la série MII. Ils ont un look vraiment particulier. Merci Jaromir Sukuba pour ces trésors !

Figure 7 : D'autres transistors fournis par Jaromir, principalement de la série GC/GF/GT/NU, construits derrière le « Rideau de fer » par Tesla en Tchécoslovaquie. La variété de types, de performances, de tailles et de boîtiers est plus grande, probablement pour servir plus de clients que juste les militaires.

les années 50. L'espionnage a permis de suivre les évolutions du reste du monde, avec un peu de retard et une grosse différence : le seul vrai client était l'État et surtout, l'Armée rouge. L'économie centralisée et planifiée n'allouait pas forcément les bonnes ressources aux bons projets ; cela limitait la diversité, comme le montre la figure 6 qui présente des références courantes de l'époque. Au final, le volume de production était un dixième de celui des États-Unis.

Le bloc du Pacte de Varsovie distribuait les secteurs d'activité parmi les pays. Parmi eux, la Tchécoslovaquie fabriquait des transistors avec le conglomérat Tesla (rien à voir avec Elon Musk, voir l'image 7) puis des circuits intégrés du style TTL. Toute cette histoire, encore obscure pour les Occidentaux, émerge lentement avec l'arrivée du Web et l'écoulement de vieux stocks de composants sur les sites de vente en ligne, ce qui fait un nouveau mystère à démêler et une histoire parallèle à découvrir. Une chose est claire cependant : l'ère du germanium (le transistor du pauvre, car les meilleurs composants étaient réservés aux militaires) et même des tubes s'est prolongée bien plus longtemps derrière le « Rideau de fer ».

4. L'AUBE DE LA SILICON VALLEY

Le tout premier transistor au silicium a été fabriqué en janvier 1954 aux Bell Labs, mais c'est Texas Instruments qui l'a industrialisé en premier, quelques mois plus

tard. Il prend rapidement à Raytheon le titre de plus gros producteur commercial de transistors **[27]**.

En 1955, Shockley **[17]** est toujours insatisfait de n'être *que* co-inventeur du transistor. Il quitte Bell Labs pour s'installer près de ses parents, dans une région de vergers en Californie, du côté de Palo Alto et Mountain View : c'est l'acte fondateur de la Silicon Valley. Il y crée sa société « Shockley Semiconductor Laboratory » pour développer les procédés basés sur le silicium et vendre ses nouveaux composants, comme les « diodes Shockley » **[28]** (similaires aux diacs). Il s'obstinera à les développer avec l'espoir d'un marché gigantesque, puisque c'est l'époque où se multiplient les ordinateurs, qui ont besoin de toujours plus de mémoire rapide, et les bistables monolithiques de Shockley sont justement rapides et compacts. Malheureusement, ils sont chers à l'unité (à cause d'un fort taux de défauts), complexes à utiliser, gros consommateurs d'énergie même inactifs, alors que les fabricants adoptent en masse les mémoires à tores de ferrite pour la mémoire centrale des ordinateurs.

Le chercheur publie ses recherches théoriques et reçoit le Prix Nobel en 1956, mais devient encore plus caractériel, affectant le moral et la rentabilité de sa compagnie, ce qui pousse huit de ses plus importants employés à démissionner à la fin 1957. Ces « traitorous eight » fondent Fairchild Semiconductor, qui aura un rôle considérable dans l'évolution du marché dans les années suivantes, en parallèle de Motorola et Texas Instruments.

Les transfuges vont essaimer bien au-delà de Fairchild. Par exemple, Robert Noyce et Gordon Moore (qui a énoncé la loi qui porte son nom) co-fonderont ensuite Intel en 1968 avec Andrew Grove, alors qu'AMD sera créé en 1969 par d'autres cadres de Fairchild. Les laboratoires Shockley sont revendus en 1960 et William Shockley enseignera à l'Université de Stanford de 1963 à 1975 (où il affichera ses convictions eugénistes). Le germanium a déjà perdu tout intérêt alors que les premiers hommes posent le pied sur la Lune.

Pourtant, dès le début, le silicium était connu pour avoir de meilleures caractéristiques : sa jonction peut monter à 125 °C au lieu de 70 °C pour le germanium, et le *bandgap* plus élevé réduit les courants de fuite ainsi que le bruit. Un cristal de silicium fut utilisé pour la réception radio dès 1906, ainsi que durant la Deuxième Guerre mondiale pour les fréquences UHF et micro-ondes : grâce à la finesse de sa pointe, la diode de détection 1N23 (silicium dopé au bore, avec une pointe en tungstène) monte jusqu'à 26 GHz ! Mais l'extrême taux de pureté nécessaire a posé des défis industriels énormes, que l'industriel Dupont a progressivement relevés. Les Japonais ont rapidement émulé ces efforts **[25b]** pour fournir du silicium pur à Sony, Toshiba et leurs concurrents.

Au-delà des applications grand public, l'un des plus grands enjeux pour ces développements est la Guerre froide avec sa course à l'armement et à la Lune. Il faut rendre les ordinateurs, les fusées et les missiles plus légers, plus petits, moins gourmands, plus tolérants aux températures élevées, et bien sûr plus fiables ! La guerre cryptographique se poursuit (mais avec le bloc de l'Est, c'est la création de la NSA) et de nombreuses machines dédiées et innovantes sont produites dans l'ombre des ordinateurs commerciaux. Dans l'espace, les liaisons radio (télémétrie, voix, image, données) utilisent des fréquences de plus en plus élevées dans des environnements très hostiles et intransigeants. Les Américains se focalisent sur ces défis, laissant mécaniquement les Japonais pénétrer le marché « grand public » (plus aligné sur le courant pacifiste d'après-guerre) grâce à sa main-d'œuvre moins chère.

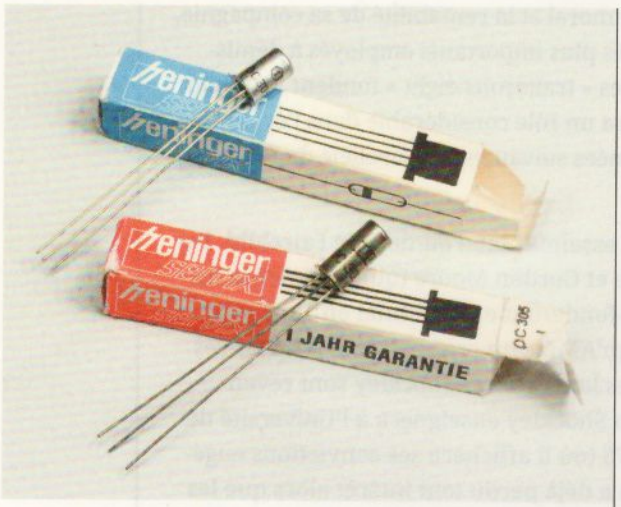


Figure 8 : Deux transistors OC305 emballés individuellement dans leur minuscule boîte en papier. Introduits en 1957 par Intermetall et fabriqués par ITT, 32 V 50 mA, Ft<2 MHz. On ne voit plus un tel soin de nos jours, où les composants ne coûtent que quelques centimes et leurs emballages sont adaptés pour les machines automatisées. Je me demande si la garantie d'un an est encore valable... Origine : https://www.ebay.fr/str/germaniumtransistor.

L'existence de procédés de fabrication de diodes au germanium a fait de ce dernier l'élément idéal pour développer l'industrie, améliorer la compréhension des mécanismes quantiques en jeu, créer une inertie dans le marché permettant de financer les appareils et les développements de la filière silicium. Celle-ci a donc partagé beaucoup d'éléments, comme certaines étapes de purification, de cristallisation ou d'encapsulation, ainsi que les méthodes et nomenclatures (au point qu'il est souvent difficile de savoir quel élément est utilisé pour un modèle donné). Ainsi, la transition a été beaucoup plus fluide pour les clients qu'avec les tubes à vide, aux caractéristiques électriques et mécaniques radicalement différentes. Mais pour les producteurs, le silicium posait quand même des défis particuliers que tous n'ont pas réussi à surmonter.

Au début, les techniques prototypées en laboratoire étaient directement appliquées à la production, donc chaque composant était fabriqué manuellement dans des usines employant des centaines de « petites mains », pour régler, encapsuler, tester et emballer chaque composant individuel (comme sur

Figure 9 : La palme de l'emballage revient à Telefunken pour son sublime écrin en carton, papier ondulé et ouate de cellulose avec de jolis motifs en relief. Confortablement installés à la main, on y trouve 25 exemplaires d'OC602 : c'est un transistor « à alliage » d'amplification audio pour l'étage final (avant un petit haut-parleur) annoncé en 1954. Une languette en métal autour du corps en verre forme un petit radiateur à visser, ce qui permet de dépasser les 50 mW typiques du modèle standard (ici, c'est le modèle « spécial »). Pour le reste, c'est un PNP au germanium, 15 V 175 mW avec un gain d'environ 25 et un courant de fuite d'environ 10 µA (à froid !). Voir aussi https://sites.google.com/site/transistorhistory/Home/european-semiconductor-manufacturers/history-of-telefunken.

les figures 8 et 9). Le coût unitaire élevé était d'abord subventionné par les clients étatiques, en particulier les militaires et les conglomérats de télécommunication. Progressivement, les usines se sont automatisées, les coûts de production ont baissé, le marché a grignoté celui des tubes, les incertitudes et aléas des procédés ont été gommés, ce qui a réduit les taux de déchets (qui pouvaient dépasser 99 % selon les conditions météorologiques) et amélioré les rendements. Les applications ont suivi ou même stimulé les avancées. Et aujourd'hui, un lot de transistors basiques dans le commerce a une dispersion des gains de quelques pour cent au pire.

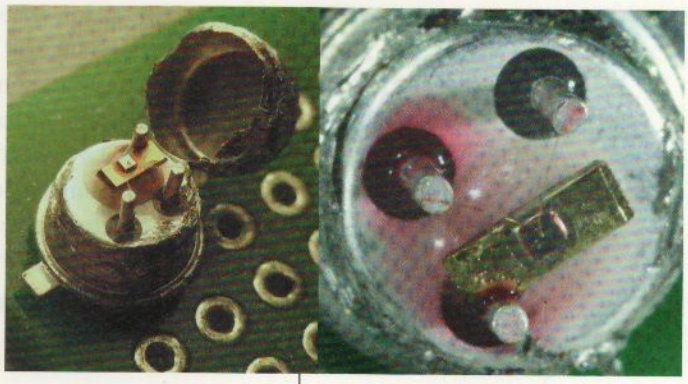


Figure 10 : Vue interne d'un transistor AF240 avec deux fins fils d'or (l'émetteur et le collecteur) reliés à un substrat de germanium lithographié (la base). J'ai essayé d'ouvrir un AF200 (très similaire), mais il est rempli de graisse de silicone qui empêche de tout voir correctement. Ici, la partie sensible est juste recouverte d'un fin vernis. Selon https://www.radiomuseum.org/tubes/tube_af240.html l'AF240 de Siemens aurait été introduit en 1966. C'est un format typique à 4 broches, dont l'une est reliée au boîtier pour réduire les capacités parasites. On ne le distingue pas facilement des nombreux autres modèles avec son boîtier à 4 pattes, son courant de base limité à 1 mA, son courant collecteur de 10 mA, mais il peut monter à 500 MHz, ce qui le destine aux oscillateurs et préamplificateurs UHF. Cependant, à technologie égale, plus la fréquence augmente, plus le gain diminue : celui-ci est d'environ 20.

Dès les années 60, le silicium s'imposait d'autant plus facilement que le marché n'était pas encore totalement saturé par le germanium. À part le premier prototype de circuit intégré de Kilby en 1958 et un prototype japonais en 1962 **[25b]**, je ne trouve aucune trace de circuit intégré sur substrat germanium. Pourtant, les mêmes techniques de lithographie peuvent aussi être employées, comme le prouve le procédé mesa présenté sur la figure 10.

5. GÉOGRAPHIE

Les fabricants se sont organisés en suivant le système inventé pour les tubes : les modèles enregistrés sur des registres (inter)nationaux peuvent être produits par plusieurs fabricants et ces « équivalences » sont importantes pour permettre les « secondes sources », qui garantissent la continuité d'approvisionnement aux assembleurs. En quelques années, ils ont par exemple mis en place des règles de nommage, des boîtiers compatibles et des tables d'équivalences, puisque chacun créait jusqu'alors de nouvelles versions dans son coin. Typiquement, l'industrie électronique a été découpée en 4 grandes régions : les États-Unis, l'Europe, le Japon et le bloc communiste.

5.1 USA

Les USA utilisent typiquement la nomenclature JEDEC, commençant par un nombre désignant le nombre de broches (moins un), suivi d'un **N**.

- Les **1N** correspondent donc aux diodes, vous connaissez certainement les 1N4148 (silicium petits signaux), 1N4004 (silicium haute tension), 1N34 (détection au germanium, introduite en 1946) ou 1N5819 (Schottky 1A).

- Pour les transistors, les 2N2222 (silicium petits signaux), 2N3055 (silicium de puissance) et 2N7000 (MOSFET N petits signaux) sont les plus connus, mais les 2N396 (PNP germanium) sont oubliés (comme 99 % des transistors enregistrés). La transition vers le silicium a été progressive et le numéro de modèle n'indique pas le type de semi-conducteur ou la géométrie, juste l'ordre d'inscription au registre.
- La série 4N correspond aux optocoupleurs, composés d'une LED et d'un phototransistor.

5.2 Europe

En Europe, après la débandade des Français de Westinghouse, les Anglais (dont Mullard, avant son intégration totale à Philips) et les Allemands (dont Intermetall, Telefunken) sont montés au créneau. Pour de nombreux modèles, la nomenclature était simple : le préfixe OA pour les diodes et OC pour les transistors, bien que chacun n'en fasse qu'à sa tête **[29]**. En 1966, un nouveau système appelé *Pro-electron* (aujourd'hui *EECA*), inspiré par la nomenclature des tubes, a été adopté :

- La première lettre désigne le type de semi-conducteur, dont A pour germanium et B pour silicium. D'autres éléments comme l'arséniure de gallium (C) sont prévus, mais il ne semble utilisé que pour les optocoupleurs (CNY17 par exemple).
- La deuxième lettre désigne l'application :
  - → A pour les diodes ;
  - → C pour les basses fréquences et signaux faibles (dont la voix) ;
  - → F pour les radiofréquences ;
  - → D pour les fortes puissances. D'autres catégories existent, mais on s'y perd vite !
- Le nombre suivant est la séquence d'inscription dans le registre mais typiquement, plus il est élevé, plus la performance augmente. Par exemple, la fréquence de transition augmente avec les nouveaux modèles, du moins *dans l'idéal* :

```
AF106  220MHz
AF138  40 MHz
AF178  180MHz
AF200  200MHz
AF240  500MHz
AF280  550MHz
AF439  400MHz
```

- Un suffixe est possible (lettre) pour indiquer le gain ou la tension de fonctionnement.

5.3 Japon

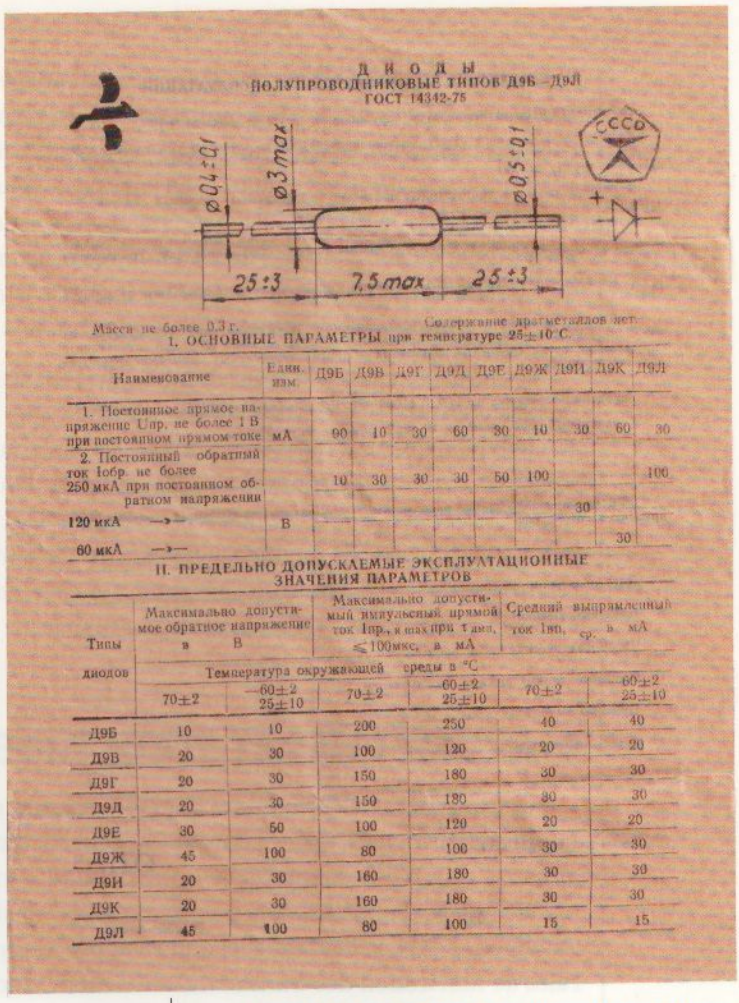
Au Japon, c'est JIS (aujourd'hui JEITA) qui normalise les références, d'une façon un peu hybride :

- Le premier chiffre est 1 pour les diodes et 2 pour les transistors.
- Ensuite vient la lettre **S**.
- En troisième position, une lettre désigne la fonction, par exemple :
  - → A PNP radio ;
  - → B PNP audio ;
  - → C NPN radio ;
  - → D NPN audio ;
  - → E diode ;
  - → J P-FET ;
  - → K N-FET ;
  - → etc.
- Enfin, un numéro d'enregistrement qui n'indique rien d'autre.

Dans le cas de l'Europe et du Japon, la normalisation s'est produite à partir de 1966, donc tout composant vendu sous ces dénominations a été fabriqué après. Auparavant, des références constructeurs étaient souvent utilisées, mais rien n'a obligé à en changer, donc ce n'est pas une garantie de millésime.

5.4 Bloc communiste

La situation en Union soviétique est différente, en partie parce que ce n'était pas une économie de marché et les décisions étaient surtout politiques. Le site **http://www.155la3.ru/aktiv.htm** propose un musée virtuel des modèles, mais la traduction automatique n'est pas claire ou fiable. De plus, comme vu dans la partie 3, l'URSS a sa propre nomenclature, différente des pays satellites comme la Tchécoslovaquie, la Pologne, l'Allemagne... Mais parlons de l'URSS en particulier, car Artem Kashkanov en est un spécialiste et entre la fabrication de deux ordinateurs Brainfuck en technologie semi-artisanale délirante (voir à **dekatronpc.com**), il m'a apporté quelques précisions pertinentes.



Д И О Д Ы
ПОЛУПРОВОДНИКОВЫЕ ТИПОВ Д9Б–Д9Л
ГОСТ 14432-75

Масса не более 0,3 г. Содержание драгметаллов нет.

I. ОСНОВНЫЕ ПАРАМЕТРЫ при температуре 25±10 °С

Наименование	Един. изм.	Д9Б	Д9В	Д9Г	Д9Д	Д9Е	Д9Ж	Д9И	Д9К	Д9Л
1. Постоянное прямое напряжение Uпр, не более 1 В при постоянном прямом токе	мА	90	10	30	60	30	10	30	60	30
2. Постоянный обратный ток Iобр, не более 250 мкА при постоянном обратном напряжении		10	30	30	30	50	100			100
120 мкА →	В							30		
60 мкА →									30	

II. ПРЕДЕЛЬНО ДОПУСКАЕМЫЕ ЭКСПЛУАТАЦИОННЫЕ ЗНАЧЕНИЯ ПАРАМЕТРОВ

Типы диодов	Максимально допустимое обратное напряжение, В		Максимально допустимый импульсный прямой ток Iпр, и max при tимп ≤ 100 мкс, в мА		Средний выпрямленный ток Iвп. ср, в мА	
	Температура окружающей среды в °С					
	70±2	−60±2 25±10	70±2	−60±2 25±10	70±2	−60±2 25±10
Д9Б	10	10	200	250	40	40
Д9В	20	30	100	120	20	20
Д9Г	20	30	150	180	30	30
Д9Д	20	30	150	180	30	30
Д9Е	30	50	100	120	20	20
Д9Ж	45	100	80	100	30	30
Д9И	20	30	160	180	30	30
Д9К	20	30	160	180	30	30
Д9Л	45	100	80	100	15	15

Figure 11 : Les boîtes de composants soviétiques contenaient typiquement une « étiquette » (этикетка en russe) rappelant les principales caractéristiques techniques. Dans le cas des semi-conducteurs, le nombre de variantes peut atteindre la dizaine selon les procédures de « binning » : cette notice liste 9 suffixes différents, faits d'une lettre cyrillique de Б à Л (pourquoi ont-ils ont sauté le A ?). Cette notice d'époque est aussi typique par son papier non blanchi. De toute façon, qui les lit ?

- Les diodes russes sont simplement désignées par le préfixe Д. La D9B (Д9Б) par exemple est équivalente à la 1N34 (détecteur au germanium). Après le numéro du modèle, on trouve souvent tout un abécédaire de suffixes qui peuvent changer complètement les caractéristiques (voir l'этикетка de la figure 11).
- Les premiers transistors russes sont désignés par un préfixe П (telle la série П1 à П3), puis МП (pour l'encapsulation **M**oderne en métal, le boîtier que l'on voit sur la figure 6) jusqu'en 1964. Le préfixe est suivi d'un numéro indiquant le matériau, la puissance, la fréquence et le modèle. Par exemple, lorsque le premier chiffre est pair (ou absent), il s'agit de germanium, sinon c'est en silicium.

```
  1-99  : Germanium basse fréquence, basse puissance (les tout premiers)
100-199 : Silicium basse fréquence, basse puissance
200-299 : Germanium basse fréquence, haute puissance
300-399 : Silicium basse fréquence, haute puissance
400-499 : Germanium haute fréquence/micro-ondes, basse puissance
500-599 : Silicium haute fréquence/micro-ondes, basse puissance
600-699 : Germanium haute fréquence/micro-ondes, haute puissance
700-799 : Silicium haute fréquence/micro-ondes, haute puissance
```

Ce qui m'intrigue est que les transistors à alliage П104 et П106 sont arrivés dès 1956, à temps pour le lancement des premiers Sputniks, mais je ne suis pas sûr qu'ils soient au silicium. Donc, vérifiez bien les références avec plusieurs sources ! Aussi, le bloc de l'Est base ses mesures sur le millimètre donc au lieu d'un pas de 2,54 mm, les écartements sont de 2,5 mm, ce qui pose parfois des soucis d'interchangeabilité avec nos composants habituels (surtout les boîtiers DIP).

Plus tard, des transistors plus économiques encapsulés dans du plastique ont reçu un autre préfixe : je dispose de KT315 (NPN silicium planaire) et KT361 (PNP silicium, similaire au BC250) qui semblent avoir été fabriqués juste après la fin de l'Union soviétique. C'est ce genre de stock que l'on trouve sur eBay de nos jours, trente ans après, montrant que l'ère du germanium, des relais, des tubes et autres curiosités étaient encore « normales » à l'époque. Tant que ça marche, pourquoi changer ?

6. GÉOMÉTRIES ET STRUCTURES

Les transistors au germanium ont été fabriqués selon de nombreux principes, qui se sont rapidement succédé pour apporter plus d'économie d'échelle, de fiabilité, de performance **[30a]**... Contrairement à G. Renard qui considère 5 méthodes **[31]**, je distingue ici quatre approches plus générales de fabrication des transistors, utilisant des techniques fondamentalement différentes, mais reposant sur les mêmes principes physiques. De nos jours, on ne trouve pratiquement que des composants réalisés selon les deux dernières méthodes, en raison de leurs volumes de production élevés et des stocks accumulés, alors que les premiers sont tombés en panne ou ont été démantelés depuis bien longtemps.

6.1 Transistors à contact

Ce sont les premiers types découverts et construits, représentés par les figures 4 et 5.

Nous en avons déjà discuté, et leurs défauts sont nombreux. D'abord, ils sont mécaniquement fragiles, sensibles aux vibrations, à l'humidité et la température qui agissent sur les contacts, malgré les tentatives de fiabilisation. Même noyer le dispositif dans de la résine ne suffit pas. Ensuite, ils sont difficiles à régler manuellement : les électrodes sont posées individuellement par de « petites mains », les taux de rebuts sont très élevés et ils sont donc impossibles à produire en série à bas coût. De plus, ils ne peuvent pas dissiper de puissance, donc ils sont seulement adaptés pour l'amplification de signaux faibles (vocaux ou numériques) en espérant qu'ils ne soient pas noyés dans le bruit. Enfin, malgré les performances honorables des Westcrel, la vitesse n'est pas fulgurante. En 1953, le modèle GC1 de Telefunken a remplacé les ressorts avec des contacts à l'or dopé mais il était déjà surpassé.

Comme H. Peter Friedrichs l'a documenté **[10]**, des hobbyistes ont tenté de reproduire cette structure en bricolant des 1N34, mais les résultats sont très aléatoires, renvoyant aux affres des postes à galène. En plus de leurs défauts, le gain est faible, donc lorsque le composant est utilisable, il est peu utile, et pas longtemps.

6.2 Transistors à jonction

Parlons maintenant du « transistor à jonction bipolaire », plus précisément de « grown-junction » en anglais, mais c'est difficile à traduire, « tiré » ou « tirage » est assez juste mais peu employé.

C'est l'invention de William Shockley, peu après la controverse sur l'attribution du brevet des transistors à contact. Je ne sais pas si parler d'*invention* soit le meilleur terme, mais c'est en tout cas l'évolution logique des travaux de ses collègues durant les années précédentes. Comme le rappelle Brattain durant son discours de réception du Prix Nobel **[32]**, Bell Labs a fait appel aux métallurgistes Ohl, Scaff et Theuerer (entre autres) pour purifier à l'extrême le silicium et le germanium, et ils ont fait des expériences avec le dopage des cristaux. En jouant sur cela, ils ont créé la première jonction P-N monolithique **[31]** et Shockley a simplement étendu le principe à trois zones de dopage alternées, formant la structure PNP (ou NPN au besoin). Cela explique pourquoi cette « invention » a eu lieu si rapidement après l'annonce du transistor à contact : la technique était déjà opérationnelle, il suffisait de changer un paramètre.

Donc, ce procédé modifie la manière de créer le cristal de germanium : au lieu de chercher la pureté ultime, on introduit alternativement les dopants « dans la masse » durant le « tirage » de la « boule » (*rien à voir avec Dragon Balls*). Des substances comme l'indium ou le phosphore sont ajoutées durant la formation du cristal, selon la polarité désirée, et les intervalles d'introduction contrôlent l'épaisseur de chaque couche (la base doit être la plus fine possible).

Cela crée donc un sandwich à l'intérieur du cristal qui sera ensuite découpé, telles des allumettes (ou des frites). Deux électrodes (émetteur et collecteur) tiennent fermement les deux extrémités du minuscule barreau, il ne reste plus qu'une aiguille à régler (manuellement) pour constituer la base (en touchant la fine couche de dopage inverse). Les deux jonctions sont dans la masse du cristal et non plus à la surface, ce qui permet de commuter plus de puissance, et c'est un peu moins fragile que son prédécesseur. De plus, en réglant les dopages asymétriques, on peut mieux distinguer le collecteur de l'émetteur et améliorer le gain. On retrouve la structure de ce type de transistors dans les vieux schémas d'IBM (figure 12).

C'est pourtant encore une voie sans issue pour les mêmes raisons que le transistor à contact : le coût de la main-d'œuvre, la fragilité et la sensibilité à l'environnement, et les performances encore insuffisantes auront raison de cette branche de l'évolution. Dorénavant, les « boules » de cristal vont s'allonger de plus en plus (pour donner des barreaux ou des « lingots » à découper en *carpaccio*) et la pureté sera encore améliorée, le dopage se faisant ensuite dans un four, par diffusion en phase gazeuse ou par épitaxie.

6.3 Transistors à alliage

On va enfin parler sérieusement. C'est encore une autre approche, contemporaine, mais pourtant à l'opposée des transistors à jonction précédemment décrits, utilisant de nouveau une fine lamelle de cristal purifié et dopé. Au lieu de poser des pointes à la surface, un autre métal (tel que de l'indium) est « cuit » sur chaque face et fournit une plus grande surface ainsi qu'une bien meilleure résistance physique. Lors de la cuisson, le métal fond et va créer un *alliage* avec le germanium (d'où son nom, « alloy » en anglais **[33]**) qui constitue une jonction P-N. Grâce au réglage précis (et automatisable) de la distance entre les deux électrodes, il faut moins d'énergie pour affecter le cristal de la base, et la fréquence de fonctionnement augmente. La puissance est aussi plus facile à dissiper, donc de plus forts courants sont possibles. On peut enfin concevoir des régulateurs électroniques d'alimentation ou des amplificateurs qui attaquent directement un

haut-parleur ! En plus, la fabrication est beaucoup moins délicate et requiert moins de finesse manuelle.

General Electric et RCA ont développé cette méthode en 1950, et elle fut vite adoptée par beaucoup d'autres fabricants, qui vont l'améliorer, donnant par exemple les transistors à *base diffusée*. IBM en a dérivé son transistor « drift ». Philco crée aussi le « surface barrier transistor » **[34]** et en 1953 son SBT100 commence à concurrencer les tubes pour la modique somme de 80 USD (de l'époque) la pièce. Il concurrence les tubes et trouve sa place dans les radios, les satellites, les ordinateurs... Selon Hannon Yourke **[35]** : ¢

« One of the types we used was the Philco surface barrier transistor, which was about the best device you could get your hands on at that time. These were very uniform – I guess they were etched to achieve very specific characteristics. The one problem with these was that the devices would fail if the well specified collector breakdown voltage was even slightly and transiently exceeded – otherwise, these were very reliable. »

Avec de la chance, on peut encore trouver quelques transistors de ce type (nous en verrons plus loin à la figure 19), mais ils ont tous été supplantés par la génération suivante.

6.4 Lithographie

Je place dans cette catégorie tous les procédés qui utilisent une projection photographique d'un motif répétitif sur une fine tranche de semi-conducteur (le fameux « wafer » découpé dans un monocristal créé de la même façon qu'une « boule » mais plus longue), ainsi que des étapes de gravure, de déposition chimique (liquide ou gazeuse) et de masquage. Il n'a pas fallu longtemps pour que la méthode soit étendue d'un transistor à tout un circuit, mais cela a demandé plusieurs innovations technologiques, comme les masques photosensibles adaptés ou les différentes méthodes d'implantation des ions.

- Le transistor mesa, ainsi nommé, car sa forme ressemble à une colline, est développé à Bell Labs en 1955 et commercialisé par Fairchild en 1958, ainsi que d'autres (dont Siemens) dans les années 60 (voir la figure 10). Il est rapide, la fabrication est mieux contrôlée, moins chère et la plupart des opérations initiales sont réalisées automatiquement sur des « wafers » de quelques centimètres de diamètre. Cela améliore l'homogénéité des caractéristiques mais le composant reste sensible.
- Le transistor planaire apparaît en 1959 grâce aux travaux de Jean Hoerni chez Fairchild. Cela résout la plupart des défauts précédents, en particulier grâce à l'invention du dopage sélectif, lui-même permis par la « passivation » : une couche de silice isolante est déposée sélectivement dans un four, protégeant aussi la surface du composant des contaminations. Les performances initiales sont modestes, mais le coût de production descend en flèche, annonçant l'arrivée des circuits intégrés et donc d'ordinateurs encore plus compacts et puissants.

La suite n'est plus très pertinente, car je n'ai pas connaissance de transistors planaires au germanium mis sur le marché, et la Loi de Moore prend le relais. Évidemment, d'innombrables variations de chacune de ces approches ont été explorées donc ce n'est pas exhaustif, mais ces quatre grandes étapes permettent de comprendre le chemin parcouru, et accessoirement de mieux apprécier les défauts d'un modèle donné.

7. UTILISATION INFORMATIQUE

À partir de 1954, le mot « transistor » est synonyme de « récepteur radio portable » avec la mise sur le marché du Regency TR-1, équipé de 4 transistors fabriqués par Texas Instruments, montrant la voie à Sony et à une toute nouvelle industrie. Ce n'est qu'une des nombreuses révolutions permises par ce composant : l'informatique aussi se métamorphose, même si les diodes avaient déjà joué un rôle essentiel. Ken Shiriff me rejoint d'ailleurs sur ce constat **[30b]** : on peut tout à fait fabriquer des circuits logiques sans transistor (avec des relais, des tubes à vide ou des amplificateurs magnétiques), mais c'est beaucoup plus pénible sans diode.

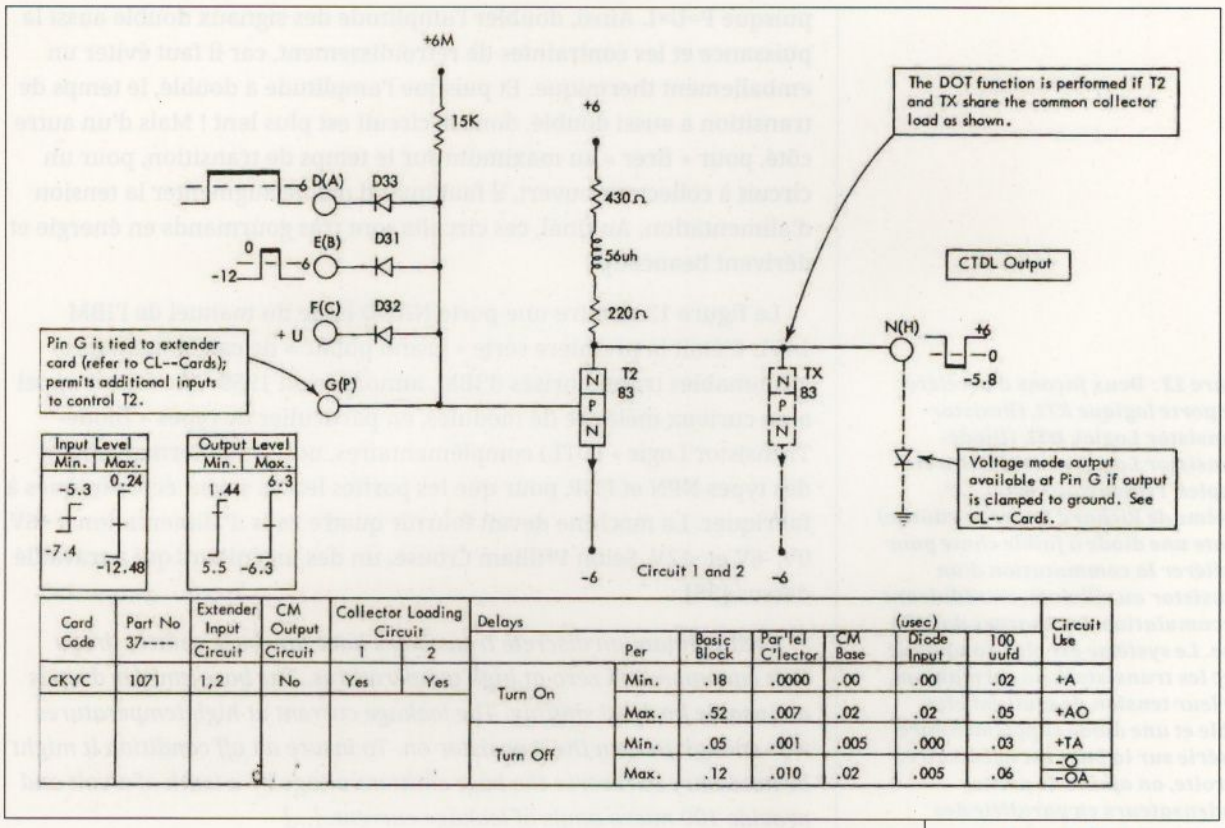
Il est difficile de dire exactement quel fut le premier ordinateur entièrement transistorisé, car les coûts considérables pour l'époque (20 USD environ le mouton à trois pattes, vers 1960 chez IBM) le réservaient initialement aux circuits les plus critiques et rentables, laissant des tubes (bien plus économiques, à environ un dollar la pièce) faire le reste du travail.

Un historique des ordinateurs occidentaux est disponible sur Wikipédia **[36]** **[37]** alors qu'il est difficile de trouver une liste des ordinateurs soviétiques transistorisés. Il semblerait que l'éclosion se situe vers 1954, avec une explosion de prototypes et de modèles à partir de 1956 (coïncidant aussi avec l'apparition des mémoires à tores de ferrites). Comme un ordinateur nécessite des milliers de transistors, leur coût unitaire doit être minimisé, donc ils n'utilisent pas forcément le tout dernier modèle disponible, qui serait trop cher en grands volumes. En fait, la viabilité de cette nouvelle génération de machines dépend non seulement du minimalisme de l'architecture, mais surtout des efforts de l'industrie du semi-conducteur.

Pour être utiles en informatique, les transistors doivent aussi être très rapides et les modèles concurrençant les tubes d'alors (comme les transistors à alliage) ont mis du temps pour être développés et arriver en quantité sur le marché. Sans oublier que les premiers adopteurs d'une technologie paient le prix de son développement, aussi bien financièrement qu'en subissant les contrecoups des défauts qui ne se révèlent qu'une fois installés en masse chez les clients (par exemple : Fairchild et les problèmes de contamination).

Des circuits logiques rudimentaires étaient possibles avec les premiers exemplaires, mais il faut un gain suffisant, une tension de saturation suffisamment faible, un courant de fuite minimal et bien sûr une vitesse de commutation élevée. Or, les tout premiers modèles dépassaient à peine le spectre audible : c'était suffisant pour amplifier du son pour un petit haut-parleur, et c'est tout. Les tubes étaient au sommet de leur adoption avec des mégaprojets comme SAGE, il aurait été ridicule pour un projet militaire d'utiliser une technologie qui n'avait pas encore fait ses preuves, ce qui prendra toute une décennie.

Avec les améliorations rapides des performances et des coûts des transistors, ces derniers ont été assemblés par dizaines de milliers dans de nouvelles machines, plus compactes que les prédécesseurs à tubes. Alors que ces derniers s'accommodent des températures élevées qu'ils génèrent, les transistors de l'époque supportent mal d'être agrégés avec une forte densité. Par exemple, en supposant une puissance typique de 50 mW par transistor (sans compter les nombreuses résistances autour), un récepteur radio portable avec sa dizaine de transistors va consommer un demi-watt, ce qui est raisonnable, car la dissipation est aisée, le circuit imprimé étant assez aéré.



Card Code	Part No 37----	Extender Input Circuit	CM Output Circuit	Collector Loading Circuit 1	Collector Loading Circuit 2
CKYC	1071	1,2	No	Yes	Yes

Delays (usec)	Per	Basic Block	Par'lel C'lector	CM Base	Diode Input	100 uufd	Circuit Use
Turn On	Min.	.18	.0000	.00	.00	.02	+A
	Max.	.52	.007	.02	.02	.05	+AO
Turn Off	Min.	.05	.001	.005	.000	.03	+TA
	Max.	.12	.010	.02	.005	.06	-O -OA

Figure 12 : Exemple de porte NAND à 3 entrées d'un IBM 1401 (1959), réalisée dans une version particulière de logique à diodes (Diode-Transistor Logic). Source : https://ibm-1401.info/IBM-StandardModularSystem-Neff7.pdf.

Dans un ordinateur par contre, la dissipation est plus difficile et tous ces transistors et composants passifs agglutinés augmentent la température des jonctions, parfois au-delà de la limite des 70 °C ; en fait, puisque le courant de fuite dépend exponentiellement de la température, les soucis commencent dès 40 °C. Dans le cas du « petit » IBM 1401 avec ses *seulement* 10 000 transistors, rien que l'ensemble de ceux-ci dissipe plus de 500 watts ! Le temps moyen de fonctionnement (MTBF) s'en trouve considérablement réduit (parfois à quelques heures, selon les conditions) et la climatisation n'est pas une option !

À haute température, le bruit électronique augmente et provoque deux effets parasites :

- La tension de seuil baisse et le transistor se déclenche à un niveau différent de la valeur « à froid », parfois à seulement 0,1V. Cela oblige à polariser la base du transistor à une valeur légèrement négative pour garantir qu'il soit *vraiment* bloqué.
- Les courants de fuite augmentent, parfois au point que le transistor soit perçu comme passant par un circuit en aval.

Ces deux effets des dérives dues à la température obligent à augmenter la marge, donc l'amplitude des signaux logiques, donc les tensions d'alimentation, donc les puissances à dissiper,

puisque P=U×I. Ainsi, doubler l'amplitude des signaux double aussi la puissance et les contraintes de refroidissement, car il faut éviter un emballement thermique. Et puisque l'amplitude a doublé, le temps de transition a aussi doublé, donc le circuit est plus lent ! Mais d'un autre côté, pour « tirer » au maximum sur le temps de transition, pour un circuit à collecteur ouvert, il faut quand même augmenter la tension d'alimentation. Au final, ces circuits sont très gourmands en énergie et dérivent beaucoup !

La figure 12 montre une porte NAND issue du manuel de l'IBM 1401. C'était la première série « grand public » de calculateurs programmables transistorisés d'IBM, annoncée en 1959. Elle faisait appel à un curieux mélange de modules, en particulier de types « Diode-Transistor Logic » (DTL) complémentaires, utilisant alternativement des types NPN et PNP, pour que les parties lentes soient économiques à fabriquer. La machine devait fournir quatre rails d'alimentation à +6V, 0V, -6V et -12V. Selon William Crouse, un des ingénieurs qui a travaillé dessus **[38]** :

« All germanium discrete transistors have low base emitter drops that can approach zero at high temperatures. The base-emitter drop is not usable for level shifting. The leakage current at high temperatures was enough to turn the transistor on. To insure an off condition it might be necessary to reverse the base emitter voltage by a tenth of a volt and provide 100 micro amps of leakage current. [...]

Germanium transistors at high temperatures required the base-emitter be reverse biased and its large leakage current provided to insure the Off condition. R4 and the power supply provided this, an extra power supply but essential to a reliable design. I might mention a circuit for a hobbyist might work without R3 and its power supply but not reliably especially at elevated temperatures. Many circuits published in amateur hand books show circuits that won't work at elevated temperatures or for quantity production. Even component vendor handbooks sometimes had circuits that would not function as a reliable computer design. Design that could provide the reliability needed to insure 10,000 circuits would all work over full temperature and all the component tolerances was not understood by even college professors teaching new engineers. »

Pour résumer, un circuit qui fonctionne seul sur une table de laboratoire ne tiendra pas en pratique à cause de

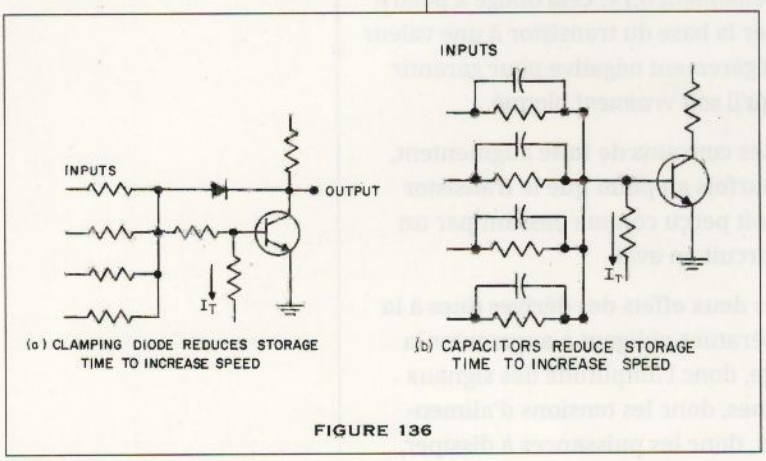


Figure 13 : Deux façons d'accélérer une porte logique RTL (Resistor-Transistor Logic), DTL (Diode-Transistor Logic) ou DCTL (Direct Coupled Transistor Logic). Le système de Richard Baker (à gauche) ajoute une diode à faible chute pour accélérer la commutation d'un transistor au silicium, en réduisant l'accumulation des charges dans la base. Le système est plus compliqué avec les transistors au germanium, car leur tension de seuil est plus faible et une diode supplémentaire en série sur la base est nécessaire. À droite, on ajoute de petits condensateurs en parallèle des résistances de limitation de courant de base, pour charger ou décharger plus rapidement la base durant les périodes transitoires. Lorsque les condensateurs et résistances ont les bonnes valeurs, et en ajoutant la diode, la porte logique peut être deux ou trois fois plus rapide ou consommer moins [47]. Source : p.97 de [39].

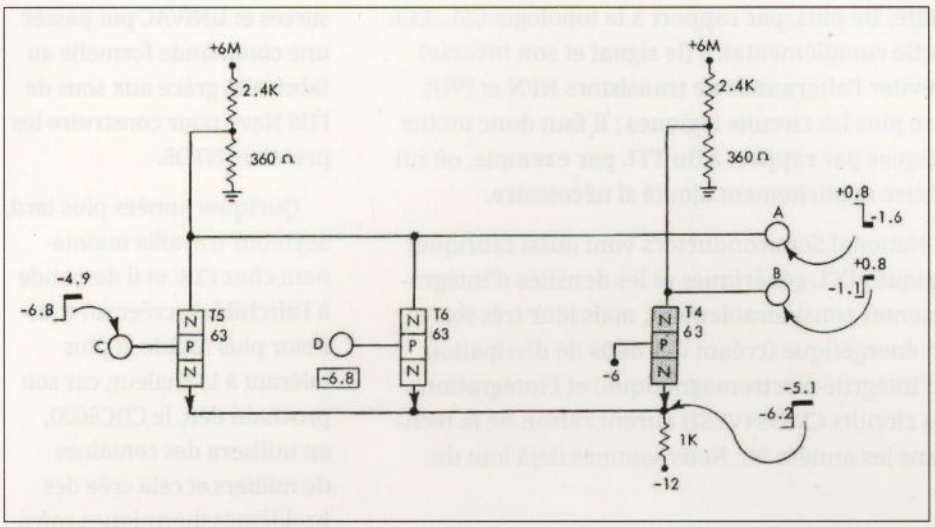


Figure 14 : Un circuit OR en topologie Current Mode Logic pour le 1401 d'IBM. Les transistors ne saturent pas donc commutent beaucoup plus vite, mais le circuit est plus cher qu'en version DTL. Source : https://ibm-1401.info/IBM-StandardModularSystem-Neff7.pdf.

l'échauffement, beaucoup d'idées et de schémas circulent mais ne sont pas fiables. L'art subtil d'utiliser ces composants se développe alors progressivement grâce aux manuels fournis par les fabricants **[39]**, mais il reste beaucoup de malentendus, en particulier pour les ingénieurs qui utilisaient des tubes avec leurs particularités différentes. La gestion thermique devient critique.

Et ce n'est pas fini. Les transistors dédiés aux circuits radio sont relativement rapides, mais traitent des signaux faibles, alors qu'un ordinateur travaille avec des 1 et des 0 séparés par une grande amplitude. Nous venons de voir que c'est nécessaire pour assurer la polarisation du circuit dans toutes les conditions de température, mais cela sature la base avec des charges électriques qu'il faut pomper à chaque transition, ralentissant encore plus tout le circuit. Cela a freiné l'adoption de la topologie DTL.

Comme le remarqua Richard H. Baker en 1956, le transistor saturé se comporte alors comme un condensateur, dont il propose de limiter la charge par une diode placée stratégiquement (figure 13). Cela donne le « Baker clamp » **[40]**, souvent réalisé avec une diode (Schottky ou germanium) entre le collecteur et la base. On retrouvera ensuite ce principe dans les circuits TTL des séries S, LS et ALS par exemple.

En 1956, Hannon S. Yourke développe chez IBM une autre méthode pour ne pas saturer les transistors et garder une vitesse élevée : c'est la technologie *Current Steering Logic* (ou *Current Mode Logic*) **[41] [42]**, qui évoluera et deviendra la technologie ECL, signifiant *Emitter Coupled Logic* **[43]**. Les modules de l'IBM 1401 qui l'utilisent sont beaucoup plus chers, mais bien plus rapides par rapport à la technologie DTL rudimentaire. Pour y arriver, la porte logique comprend plusieurs transistors travaillant en régime linéaire au lieu de diodes, comme sur la figure 14.

Ce type d'électronique analogique appliquée au numérique a suivi la transition vers le silicium, entre autres sous l'impulsion de Motorola : sa famille de composants intégrés MECL (introduite en 1962) a permis la construction des ordinateurs les plus rapides du monde, dont le Cray-1 en 1976. Ces circuits intégrés (sans germanium) compensent les variations de température ainsi que des tensions d'alimentation, augmentant

encore la fiabilité. De plus, par rapport à la topologie CSL, ECL fournit une sortie complémentaire (le signal et son inverse) qui, en plus d'éviter l'alternance de transistors NPN et PNP, simplifie encore plus les circuits logiques ; il faut donc moins de couches logiques par rapport à du TTL par exemple, où un inverseur doit être explicitement ajouté si nécessaire.

Fairchild et National Semiconductors vont aussi fabriquer des circuits logiques ECL génériques et les densités d'intégration vont augmenter considérablement, mais leur très forte consommation énergétique (créant des défis de dissipation thermique et d'intégrité électromagnétique) et l'intégration inéluctable des circuits CMOS (VLSI) auront raison de la technologie ECL dans les années 90. Nous sommes déjà loin du germanium.

Bien que les transistors à alliage soient créés en 1950 (juste après les transistors à « jonction tirée »), il faudra attendre 1953 environ pour que les premiers transistors germanium adaptés aux ordinateurs apparaissent. Si l'on en croit une page dithyrambique de Wikipédia **[34]**, c'est Philco qui a développé un nouveau type plus rapide, avec une nouvelle méthode de gravure chimique permettant une base mieux contrôlée et encore plus fine. Cela ouvre la voie à de nouvelles applications tant en radio qu'en informatique et on assiste dès 1955 à une sorte d'explosion cambrienne, avec la multiplication des fabricants sous licence (dont *General Transistor Corporation* ou GTC). Malgré le coût initial (80 USD de l'époque la pièce), de nombreux autres ordinateurs expérimentaux, plus performants et moins encombrants voient le jour. Par exemple, le TX-0 **[44]** du MIT sert (entre autres) à programmer les premiers jeux vidéos et est l'ancêtre du PDP-1, donc de l'empire Digital Equipment Corporation.

La rumeur raconte qu'en raison du coût de ces nouveaux transistors, les rebuts de fabrication suffisamment fonctionnels étaient vendus au rabais aux bricoleurs ou bien offerts par les commerciaux pour attirer les nouveaux clients. En 1956, un petit malin testait justement le nouveau composant pour le projet TRANSTEC chez UNIVAC **[45]** : il constata que des rebuts convenaient très bien pour les circuits numériques, qui s'accommodent des larges variations de caractéristiques (contrairement aux radios ou aux circuits audio). Après avoir écumé sa région, des milliers d'échantillons ont ainsi été collectés par Seymour Cray pour créer son prototype d'ordinateur transistorisé (le premier d'une longue liste). Heureusement, le prototype fut un succès et UNIVAC put passer une commande formelle au fabricant, grâce aux sous de l'US Navy pour construire les premiers NTDS.

Quelques années plus tard, Seymour travaille maintenant chez CDC et il demande à Fairchild de créer un transistor plus rapide et plus tolérant à la chaleur, car son prochain défi, le CDC6600, en utilisera des centaines de milliers et cela crée des problèmes thermiques considérables. Fairchild annonce l'aboutissement de ses recherches en 1961 **[46]** : le 2N709 au silicium, développé par Jean Hoerni, supplante le germanium et CDC en commandera des dizaines de millions (assez pour construire cent exemplaires de CDC6600 contenant chacun un demi-million de transistors, en fonction des options). Le germanium commence à décliner dans l'industrie et ne se relèvera plus.

Le descendant direct du 2N709 est le 2N2369 que l'on trouve aujourd'hui à prix modique en grandes quantités. Son dopage à l'or est spécifiquement conçu pour désaturer rapidement, mais l'ajout d'une diode de Baker entre la base et le collecteur, ainsi que d'un condensateur de liaison, augmentent encore le rapport vitesse/consommation. Des essais montrent qu'un inverseur à

base de 2N2369 peut commuter en moins de 2 nanosecondes en soignant un peu les valeurs des composants qui l'entourent **[47]**. Donc un CDC6600 aurait pu être moins énergivore, mais Seymour avait déjà en tête les circuits intégrés pour le CDC7600.

8. LE GERMANIUM AUJOURD'HUI

Comme les dinosaures, les transistors au germanium sont éteints, mais ils n'ont pas disparu puisque nous les trouvons maintenant dans des collections ou des sortes de musées. Il n'y a plus d'intérêt industriel et pour l'illustrer, lorsque l'on cherche le mot « germanium » sur les sites des grands distributeurs de composants, on trouve plutôt des transistors SiGe (Silicium-Germanium) avec des fréquences de transition de dizaines de gigahertz. Plus personne n'utilise les antiques composants dans de nouveaux produits, à part, peut-être, comme avec les tubes à vide, pour ajouter un cachet « vintage » à un produit musical, par exemple.

Figure 15 : Des transistors au germanium sont encore disponibles au comptoir de Saint-Quentin Radio à Paris ! Notez aussi les prix indiqués en nouveaux francs.

Une catégorie d'audiophiles ne jure que par la chaleur des triodes ou des pentodes et, quelle que soit la validité de leurs arguments subjectifs ou pseudoscientifiques, ils sont prêts à payer parfois très cher leur mouton à cinq pattes. Lorsqu'il y a une demande, le marché adapte l'offre, puis la culture alternative correspondante se développe pour amplifier le biais de confirmation. La citation de Wikipédia du début de l'article nous rappelle que les mordus de pédales d'effets pour guitare sont friands de germanium et ils semblent constituer la majorité des clients actuellement, au vu des intitulés des annonces sur eBay par exemple, où le mot-clé « germanium » est souvent synonyme de « fuzz » (un type de pédale d'effet).

En soi, c'est un loisir comme un autre : comme la plupart des stocks d'origine ont fondu, les meilleurs composants ont été déjà utilisés et ceux restant sur le marché n'ont plus des caractéristiques aussi homogènes. Les transistors sont souvent vendus sans indiquer de gain spécifique donc c'est une loterie, et pour ne pas tomber sur un « citron » (une pièce aux performances intolérables), les clients sont implicitement encouragés à en acheter plus pour tomber sur une perle rare. L'incitation est plus forte encore si le circuit à réaliser comporte une paire dont les gains doivent être égaux (dans la limite des dérives de mesure, de la température, du courant de polarisation...).

En suivant cette logique, les composants dont les gains sont déjà mesurés sont plus chers, et les paires parfaitement complémentaires le sont encore plus. C'est une activité que peu d'électroniciens pratiquent de nos jours puisque la qualité des composants modernes assure des tolérances de quelques pour cent, et pas cinquante ou cent.

Une autre activité concernée par les transistors au germanium est la réparation des appareils anciens ou la préservation du patrimoine (comme les passionnées de **http://www.tsf-radio.org/** ou **https://www.radiomuseum.org** par exemple). Il y a aussi des radioamateurs qui essaient de réaliser un circuit trouvé dans un livre ou un magazine rédigé il y a cinquante ou soixante ans. Ils puisent aussi dans les fonds de tiroirs et les fins de stocks qui ont pris la poussière durant plusieurs décennies, dans les réserves des magasins d'électronique qui n'ont pas encore fermé. J'ai trouvé quelques références intéressantes chez mon habituel dealer parisien, en AF et AC, comme le montre la photographie de la figure 15, page précédente.

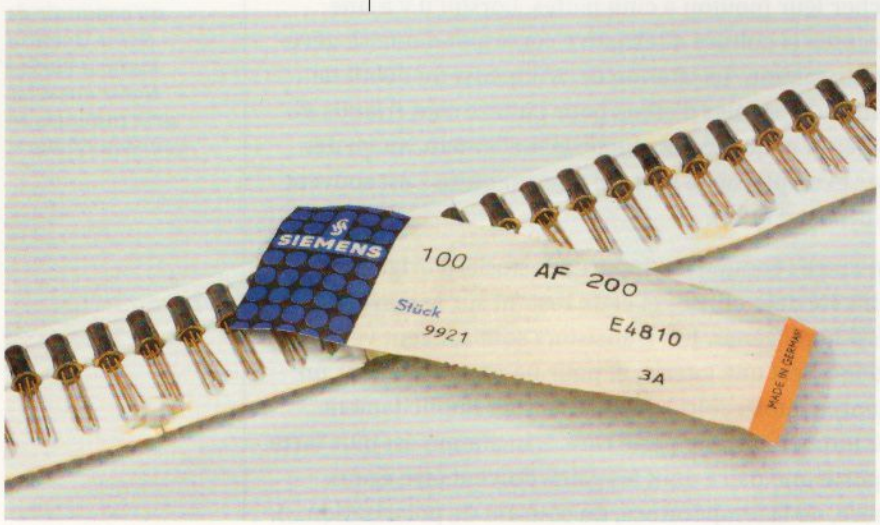


Figure 16 : Selon cette étiquette d'un lot de 100 AF200 (PNP germanium mesa, 25V 10 mA 225 mW hFE>30, préamplificateur UHF donc probablement 200 MHz), ce modèle de transistor introduit vers 1965 aurait été produit jusqu'en 1999 (21e semaine ?). C'est une sacrée carrière ! Je me demande juste qui en utilisait encore à cette époque, pour justifier de garder en état de marche une ligne de production pour une technologie si rudimentaire. Qui sait si le semi-conducteur à l'intérieur est une version plus récente, mais vendue plus chère en lui mettant une référence plus ancienne, car qui peut le plus peut le moins, n'est-ce pas ? Origine : https://www.ebay.fr/str/germaniumtransistor.

J'ai été surpris qu'une des références, en rupture de stock au moment de la visite, soit en attente de livraison pour des pièces supplémentaires. Je ne pensais pas qu'il y aurait encore des clients et encore moins des fournisseurs réputés pour ce type de composant. Il semblerait que tous les stocks des distributeurs n'aient pas encore été écoulés et on trouve des composants NOS (*New Old Stock*) fabriqués relativement récemment (figure 16).

J'ai acheté quelques références pour jouer avec, en particulier des AF106 à 1 € l'unité qui devraient être suffisamment vieux pour être « lents » et capricieux. Mais puisqu'il s'agit d'une série AF, on s'attend quand même à un transistor capable de travailler à quelques mégahertz *au moins*. En cherchant les caractéristiques sur le Web, le deuxième résultat renvoie vers un magasin qui vend des AF106 avec l'argument suivant :

« AF106 est un petit transistor à germanium conçu pour les audiophiles exigeants. Ce composant électronique offre des performances musicales exceptionnelles avec une distorsion harmonique agréable et une réponse en fréquence étendue. Il est idéal pour les applications audio de haute qualité, telles que

les préamplificateurs, les amplificateurs de puissance et les égaliseurs. Avec une capacité de courant élevée et une faible tension d'offset, l'AF106 garantit une stabilité et une fiabilité optimales. Ce transistor germanium petit signal est indispensable pour tout passionné de musique cherchant le son parfait. »

Une tension d'offset ? Est-ce un amplificateur opérationnel ?

Vous comprendrez pourquoi je ne voudrais pas faire de publicité pour ce magasin en donnant son adresse : l'AF106 est en fait un transistor haute fréquence type mesa, introduit vers 1962, souvent d'origine Telefunken/Siemens ou Tungsram et couramment donné pour une fréquence de transition de 220 MHz. Ici, il est vendu trois fois plus cher que dans mon magasin de quartier grâce à des foutaises, alors qu'on peut aussi en trouver à cinquante centimes l'unité si l'on en prend une centaine autre part. C'est bon à savoir, car avec un courant de collecteur maximal de 10 mA, il faudra en mettre beaucoup en parallèle pour réaliser un amplificateur de puissance !

En fait, à moins de cibler spécifiquement un transistor de puissance, la plupart des transistors au germanium encore disponibles sont limités en puissance et en courant, de par leur conception et leur orientation vers les petits signaux (AF et AC) : typiquement, le courant de base est autour de 1 mA et le courant de collecteur est 10 mA, la différence se situe ensuite dans le boîtier ou la fréquence de coupure. Cela donne l'impression qu'ils se ressemblent tous...

9. LA DERNIÈRE TECHNOLOGIE ENCORE UN PEU « VISIBLE »

Les relais électromécaniques ont cet avantage que leur fonctionnement est suffisamment lent pour être observable de nos propres yeux, et leurs cliquetis complètent le panorama sensoriel par le son et les vibrations. C'est une technologie excellente pour montrer le fonctionnement d'un circuit numérique aux débutants et aux curieux, pour relier le physique à l'informatique. Les tubes à vide (triodes ou pentodes) sont beaucoup moins adaptés : on ne peut observer directement que la lueur de la cathode chaude, qui ne reflète pas l'activité logique d'un circuit. Et les tubes chauffent et utilisent de hautes tensions, ce qui n'est pas le plus sûr pour présenter les fondements de l'informatique à de jeunes inconscients aux mains pleines de doigts baladeurs.

Les transistors au germanium sont les derniers à permettre une observation presque directe du fonctionnement d'un circuit, sans haute tension ni fort courant, grâce à l'effet photoélectrique inhérent aux semi-conducteurs, dans des conditions très particulières cependant. L'astuce est qu'un transistor est formé par deux jonctions PN, une jonction PN agit comme une diode et qui dit diode dit électroluminescence. Donc il devrait être possible d'observer la lumière d'un transistor, variant avec son activité ! Mais je n'ai rien inventé puisque c'est une procédure de test des composants dans certaines usines, et les bricoleurs savent depuis longtemps utiliser une LED comme détecteur de lumière sensible à une plage précise du spectre visible.

Le premier obstacle est que le transistor classique est très protégé, à la fois mécaniquement, thermiquement et aussi de la lumière, puisque c'est typiquement le genre de composant dont les caractéristiques changent lorsqu'ils reçoivent des photons. L'AF200 (figure 16) ou l'AF240 (figure 10) sont

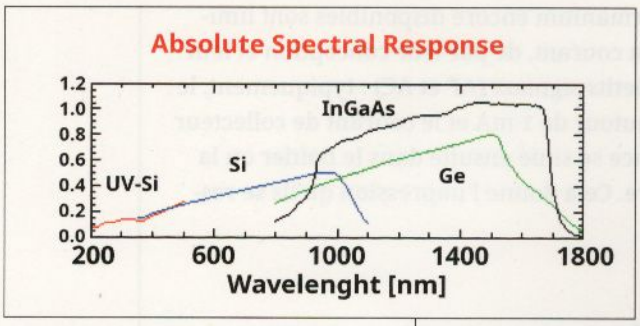


Figure 17 : Spectre de sensibilité du germanium. Source : https://chem.libretexts.org/Bookshelves/Analytical_Chemistry/Supplemental_Modules_(Analytical_Chemistry)/Instrumentation_and_Analysis/Spectrometer/Detectors/Detectors.

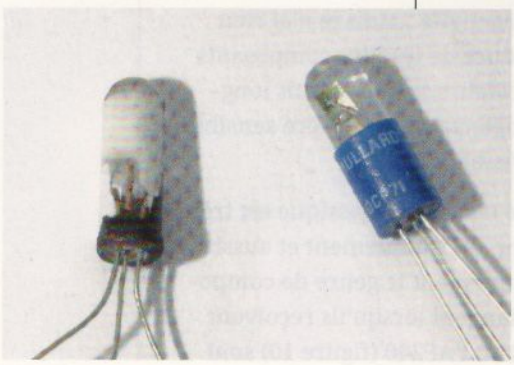
en boîtier métallique pour réduire au maximum toute perturbation lumineuse. Mais malgré la symétrie brisée, l'effet reste plus ou moins réciproque : si la jonction peut émettre, elle peut recevoir, et vice versa. *C'est quantique !*

Il se trouve que cette réciprocité fut employée assez tôt dans l'industrie, avec la création des phototransistors. Le germanium réagit à la plage 400nm-1.8nm avec une sensibilité maximale vers 1,5 µm (figure 17). Des bricoleurs peu fortunés (en particulier derrière le « Rideau de fer ») ont utilisé des transistors standard dont ils ont enlevé le capot métallique, permettant la détection de lumière ou la génération d'électricité **[48]**. L'histoire ne dit pas combien de temps les composants ont résisté à l'oxydation à l'air libre.

Les premiers phototransistors qui furent commercialisés par Mullard sont aussi une légère modification d'un modèle standard. La figure 18 montre l'OC71 original avec son ampoule de verre recouverte de peinture noire (écaillée pour l'occasion) et l'OCP71 qui en est dérivé à droite, dans son ampoule sans peinture et sans la graisse de silicone opaque (c'est plus durable que de laisser le transistor à l'air). Introduit en 1956 **[49]**, sa fabrication était donc aisée puisque c'est le même circuit à l'intérieur et ils économisaient la peinture noire. Malgré cela, l'OCP71 était vendu trois fois plus cher !

Figure 18 : Deux transistors quasiment identiques mais aux fonctions différentes : à gauche, l'OC71 est un transistor de préamplification basse fréquence que l'on ne veut pas sensible à la lumière, donc l'ampoule en verre est peinte en noir et remplie de gel silicone opaque. À droite, l'OCP71 a le même assemblage électronique dans un boîtier légèrement différent : le verre n'est pas peint et le silicone est clair pour laisser passer la lumière.

La rumeur dit que les premières versions de l'OC71 avaient la même graisse de silicone translucide et les petits malins économisaient un peu d'argent en enlevant la peinture noire pour obtenir un phototransistor à plus bas prix. Cela aurait incité Mullard à changer la formulation et utiliser une graisse opaque, chargée de poudre d'alumine « pour améliorer le transfert thermique ». Malgré cela et la peinture noire, ce modèle reste sensible à la lumière : son courant de fuite explose lorsqu'il est placé sous une ampoule, et la peinture noire (à base de cellulose) n'est pas très durable, ce qui a conduit à des pannes très étranges lorsqu'elle s'écaillait. Vous comprenez pourquoi l'industrie a unanimement adopté les boîtiers métalliques et la résine époxy noire.



La figure 19 montre le détail de la partie sensible de l'OCP71. C'est simplement une pastille de germanium, avec ses deux électrodes d'indium fondues de chaque côté. C'est très simple, et la pastille est orientée vers la paroi de verre pour mieux capter la lumière. Mais s'il est photosensible, il n'en reste pas moins un transistor et c'est exactement la même chose à l'intérieur de l'OC71 (mais vendu à un prix différent).

Donc contrairement à l'OC71, ce composant permet non seulement de voir clairement à l'intérieur, mais aussi, *éventuellement*, de détecter l'émission de lumière lorsque du courant est commuté. Il est encore possible d'acheter quelques OCP71 de nos jours, à plusieurs euros la pièce, mais la détection est bien plus difficile. En effet, le germanium a une largeur de bande interdite (*bandgap*) indirecte d'environ 0,66 eV à 300 °K, ce qui signifie en langage normal que c'est un émetteur de photons naturellement peu efficace qui émet autour de 1,84 µm (95 THz). Cela correspond à la courbe de la figure 17 : pour activer l'effet photoélectrique, les photons incidents doivent avoir une énergie supérieure (et longueur d'onde inférieure) au *bandgap*.

La température de « corps noir » correspondante est environ 1500 °K à 1600 °K, soit trop loin pour l'infrarouge proche. Donc, non seulement on ne peut pas le voir directement avec nos yeux, mais c'est aussi hors d'atteinte des détecteurs infrarouges courants en silicium (qui s'arrêtent vers 1100 nm). Vous ne pourrez pas observer le transistor fonctionner en enlevant le filtre infrarouge de votre appareil photo (comme l'a fait Denis Bodor dans plusieurs articles dont **[50]**). J'ai une minicaméra thermique 8×8 pixels, mais la température détectable n'atteint pas les 1600K (1300 °C) requis.

Nous entrons dans un domaine très particulier où les appareils sont chers (près de 2 000 € en temps de paix) et ont souvent un usage militaire, compromettant dès le début un projet ludo-éducatif qui s'annonçait passionnant. Mais au moins vous comprenez un peu mieux les implications photoélectriques du *bandgap* et pourquoi il y a un tel engouement actuellement pour les semi-conducteurs à « wide bandgap », tels le carbure de silicium (SiC) ou le nitrure de gallium (GaN).

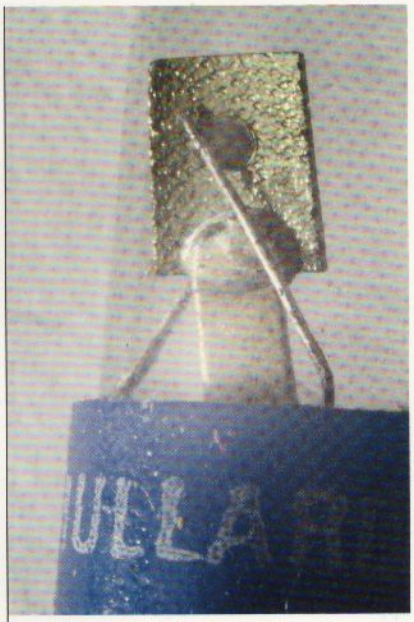


Figure 19 : Détail de l'ampoule de l'OCP71, révélant la lamelle de germanium (la base) sur laquelle sont soudées les électrodes avec de l'indium. C'est un superbe exemple de transistor à alliage !

CONCLUSION

Merci à mes « informateurs » d'avoir partagé leurs connaissances et parfois même des échantillons ! Plonger dans ce genre de recherche historique et technique est passionnant, surtout depuis qu'Internet a permis de restaurer et réunir d'innombrables informations. Il subsiste quelques informations confuses ou contradictoires, ainsi que des sujets que je n'ai pas pu aborder (pour l'instant), mais nous avons mis en perspective l'affirmation simpliste que « Bardeen, Brattain & Shockley ont inventé le transistor à Bell Labs en 1948 ». Au-delà de la vieille question « est-ce que la paternité revient à la découverte ou au dépôt de brevet ? », c'est aussi une incroyable histoire technique, scientifique, industrielle, économique et politique qui a façonné notre mode de vie actuel. Vous aurez peut-être plus de considération et de respect lorsque vous verrez un circuit au germanium en état de marche !

Mais si la nostalgie vous empoigne, pensez à ces quelques allégories. Le bois a laissé la place au charbon puis au pétrole comme source d'énergie favorite, à mesure que l'industrie s'est développée. Il est toujours agréable de faire un feu de bois dans sa cheminée, mais de moins en moins de personnes en Europe en ont une et c'est encore du pétrole raffiné que l'on verse dans les réservoirs des voitures. De même, les mémoires à bulles magnétiques, à lignes à retard ou à tores magnétiques, comme les écrans cathodiques ou même la télévision hertzienne : toutes ont eu leurs moments de succès avant de s'éclipser.

Les transistors au germanium ont eu une brève période de gloire : les ordinateurs en utilisant ont été introduits de 1953 à 1963, soit une dizaine d'années en sandwich entre les tubes à vide et les transistors au silicium puis les circuits intégrés. La victoire de ces derniers était inéluctable et la marche du progrès a été sans pitié pour les technologies au rapport performance/prix trop bas. Une fois la supériorité du silicium établie, le germanium n'était plus utilisé que par commodité : par habitude, par continuité, par disponibilité ou parce que « ça suffisait ». Dans les années 80, c'était encore un « truc à connaître parce qu'il existe toujours des livres qui en parlent », mais tout comme l'invention de Mataré et Welker, c'est aujourd'hui passé aux oubliettes. D'ailleurs, ce n'est qu'après de longues recherches d'Armand Van Dormael **[12]** puis d'autres que l'histoire du Transistron a été reconstituée et acceptée il y a vingt ans.

Figure 20 : De nos jours, un sachet de 1000 transistors au silicium (ici des BC559 : PNP - 30 V/100 mA/0,5 W/150 MHz, hfe=100-800) coûte approximativement 15 €. Non seulement ils sont moins chers que le germanium, mais les caractéristiques sont meilleures, très stables en température, et varient très peu d'une pièce à l'autre. Vive la modernité !

Il y a dix ans, je savais juste que les transistors au germanium existaient bien avant ma naissance et que cela ne valait pas la peine de perdre son temps avec. Puis un jour, j'ai trouvé un transistor neuf de référence inconnue dans un lot en vrac, lors du déstockage d'un magasin de composants électroniques, et j'ai essayé de comprendre à quoi il pouvait bien servir. De fil en aiguille, de Wikipédia à eBay, j'ai redécouvert cette technologie qui me fait d'autant plus apprécier notre confort moderne (comme celui de la figure 20). « Un gain de 20 seulement ? Quelle idée grotesque ! »

Si le mythe des transistors au germanium est encore entretenu par les bricoleurs de pédales rétro, on en retrouve aussi parfois dans de vieux récepteurs radio ou des télévisions avec une fréquence de transition de

500 MHz ou 1 GHz. Mais même là, un bête BFS480 (double NPN silicium à 10 centimes avec Ft=7GHz) conçu il y a vingt ans peut monter à 2 GHz avec moins de bruit tout en amplifiant de 10 dB. Il est déjà surclassé par ses petits frères au Silicium-Germanium, mais l'intégration croissante de toutes ces fonctions dans des puces de plus en plus complexes rend là aussi ces types de composants discrets inintéressants au niveau industriel.

À mesure que les stocks de composants d'origine s'épuisent, les prix flambent et la spéculation prend le pas, nourrie par des mythes et des légendes urbaines. Comme pour les amplificateurs audio à tubes, réputés pour la « chaleur du son », le timbre du germanium fascine quelques irréductibles à la recherche d'un Graal sonore. Et comme pour les tubes, il paraîtrait que les modèles les plus recherchés sont de nouveau en production, mais je n'ai pas pu vérifier cette rumeur. En plus de cela, les contrefaçons ou les arnaques aux « citrons » sont courantes : tout le monde se refile les mauvais composants (comme une grenade dégoupillée) pour ne pas être le perdant, et la réputation du germanium s'effondre encore plus.

Si les modèles antiques vous intriguent, on trouve encore des exemplaires de types anciens, fabriqués tardivement (tel l'AF200 du chapitre 8). Ceux vraiment fabriqués avant 1960 sont de plus en plus rares, certainement oxydés et leur utilité est anecdotique. Ils ont autant d'intérêt que la collection des timbres : on ne peut pas les utiliser et ils ont surtout un intérêt historique ou culturel. On peut toujours en montrer aux enfants pour leur dire : « votre arrière-grand-père utilisait ce truc pour écouter les émissions à la radio, c'était une révolution, comparé aux tubes ! » *Et là, évidemment, un petit malin demande ce qu'est un tube.*

Paradoxalement, les transistors au germanium ont un intérêt pour le lecteur soucieux de performance et de fiabilité. Tous leurs défauts, cités précédemment, ont été gommés avec les années, mais pas éliminés. Ils subsistent dans tous les composants actuels, à des échelles bien moindres et nous les avons oubliés, nous ne prenons même plus la peine d'examiner les courbes de leurs *datasheets*, car elles se ressemblent toutes, probablement copiées-collées d'une référence à l'autre en changeant un nombre ici et là. Les circuits modernes sont intégrés et contiennent des mécanismes de compensation.

Si vous concevez des circuits à ultra-hautes fréquences ou micro-ondes, il est très difficile d'observer tous les paramètres d'un simple circuit alors qu'une maquette à plus basse fréquence, réalisée avec un « vieux citron », vous permettra d'observer toutes les dérives avec un oscilloscope bon marché. Il en est de même pour les circuits audio, l'instrumentation scientifique ou la conception de portes logiques pour votre prochain ordinateur : les transistors au germanium, par leurs caractéristiques parasites proéminentes, sont comme des loupes pour observer les défauts des composants les plus modernes lorsqu'ils sont poussés à leurs limites. Un sèche-cheveu ou une lampe halogène sur du germanium peut simuler des conditions extrêmes pour du silicium. Mieux qu'une simulation SPICE, ils vous apprendront à mieux appréhender les contraintes et les solutions de compensation qui vous seraient inaccessibles, vous permettant ensuite de les appliquer aux dernières technologies. D'ailleurs, les transistors des circuits intégrés de toute dernière génération (nanométriques) souffrent beaucoup de problèmes de fuites, qui nous rappellent justement ceux des tout premiers modèles discrets... **YG**

RÉFÉRENCES

[1] Guidon, Yann : *« Résistons à l'informatisation galopante de l'électronique ! » Hackable n°50*, septembre 2023, pp. 4-20, **https://connect.ed-diamond.com/hackable/hk-050/resistons-a-l-informatisation-galopante-de-l-electronique**

[2] Guidon, Yann : *« La fabuleuse histoire des calculateurs numériques à l'ère électromécanique » Hackable n°25*, juillet 2018 pp. 72-80, **https://connect.ed-diamond.com/Hackable/hk-025/la-fabuleuse-histoire-des-calculateurs-numeriques-a-l-ere-electromecanique**

[3] Guidon, Yann : *« Quelques applications des Arbres Binaires à Commande Équilibrée » GLMF n°218*, septembre 2019, pp. 19-27 **https://connect.ed-diamond.com/GNU-Linux-Magazine/glmf-218/quelques-applications-des-arbres-binaires-a-commande-equilibree**

[4] **https://fr.wikipedia.org/wiki/Germanium#Électronique**

[5] Asianometry : *« China's Gallium & Germanium Export Controls »* 10 juillet 2023, **https://www.youtube.com/watch?v=J4XU-QxXJMw**

[6] **https://en.wikipedia.org/wiki/Semiconductor_device#History_of_semiconductor_device_development**

[7] *« The Copper-Oxide Rectifier [Metallic Rectifiers (1957)] »* à **https://www.industrial-electronics.com/sams_metallic_4.html** voir aussi : **https://en.wikipedia.org/wiki/Metal_rectifier**

[8] **https://fr.wikipedia.org/wiki/Récepteur_à_cristal**

a. *« Combinaisons de cristaux des détecteurs les plus populaires : Galène/Cuivre • Galène/Laiton • Galène/Argent • Pyrite/Or • Carborundum/Acier • Cuivre/Silicium • Zincite/Acier • Chalcopyrite/Zincite. »*

b. *« Dès 1939 : Dans quelques pays occupés [par les Allemands], il y avait des confiscations des postes radio de la population. Ceci conduit des auditeurs particulièrement déterminés à construire leurs propres récepteurs à pyrite clandestins, car c'était un minerai facile à trouver, à ajuster, stable [...] »*

[9] H. Peter Friedrichs : *« The Voice Of The Crystal - How To Build Working Radio Receiver Components Entirely From Scratch »* 1999, ISBN 0-9671905-0-9 **https://hpfriedrichs.com/mybooks/votc/bks-votc.htm**

[10] H. Peter Friedrichs : *« Instruments of Amplification - Fun With Homemade Tubes, Transistors, and More »* 2003, ISBN 0-9671905-1-7 **https://hpfriedrichs.com/mybooks/ioa/bks-ioa.htm**

[11] **https://fr.wikipedia.org/wiki/Procédé_de_Czochralski**

[12] Riordan, Michael : *« How Europe missed the transistor »* IEEE Spectrum, novembre 2005 *« The most important invention of the 20th century was conceived not just once, but twice »* **https://www-tc.pbs.org/transistor/materials/how-europe-missed-transistor.pdf**

[13] **https://en.wikipedia.org/wiki/History_of_the_transistor**

[14] **https://en.wikipedia.org/wiki/Joel_Barr**

[15] **https://sudonull.com/post/21885-The-first-transistor-in-space-little-known-aspects-of-the-space-race** (404, j'ai une copie locale).

[16] **https://fr.wikipedia.org/wiki/Herbert_Mataré**

[17] **https://www.computerhistory.org/collections/catalog/102702580** *« Oral History of Herbert F. Mataré »* Interviewed by Michael Riordan, Recorded: December 20, 2004, Malibu, California.

[18] **https://fr.wikipedia.org/wiki/William_Shockley#Heurs_et_malheurs_du_«_laboratoire_Shockley_ »**

[19] **https://www.computerhistory.org/revolution/digital-logic/12/273/1355**

[20] **http://www.feb-patrimoine.com/catalogue/collections_catalog05.htm** liste un Transistron prêté au CNAM.

[21] Christian ADAM : **https://www.radiomuseum.org/forum/1ere_production_de_transistors_en_europe_survivants.html** montre un Transistron Westcrel exposé à Munich.

[22] Christian Licoppe : *« Chapitre 5 - Les premières années des recherches sur les semi-conducteurs et les «transistrons» au CNET (1946-1956) »* **https://www.persee.fr/doc/reso_0984-5372_1996_hos_14_1_3670**

[23] **https://en.wikipedia.org/wiki/Kingsbury_Commitment#Continued_monopoly** *« In 1956, AT&T and the Justice Department agreed on a consent decree to end an antitrust suit brought against AT&T in 1949. Under the decree, AT&T restricted its activities to those related to running the national telephone system, and special projects for the federal government. »*
Cela affectera aussi la diffusion d'UNIX que AT&T ne pouvait pas commercialiser.

[24] Asianometry : *« How Japan Learned Semiconductors »* 6 juin 2022, **https://www.youtube.com/watch?v=bwhU9goCiaI**

[25] NHK Inc. / ministère des Affaires étrangères du Japon (approx. 1999 ?) :

a) *« Birth of The Transistor: A video history of Japan's electronic industry. (Part 1) »* (documentaire anglophone qui complète et confirme beaucoup de détails techniques et historiques) **https://www.youtube.com/watch?v=XLlIb_p11cM**

b) *« Electronic Circuits in Stone - A Video History of Japan's electronic industry - (Part 2) »* (développement du silicium) **https://www.youtube.com/watch?v=fRCqhSONNiE**

[26] Asianometry : *« Silicon Shrank the Hearing Aid »* **https://www.youtube.com/watch?v=3ykz4JAO91g**

[27] **https://www.computerhistory.org/siliconengine/silicon-transistors-offer-superior-operating-characteristics/**

[28] **https://en.wikipedia.org/wiki/Shockley_diode**

[29] Andrew Wilie : *« French Vintage Semiconductors »* **http://www.wylie.org.uk/technology/semics/France/France.htm**

[30] Ken Shirriff : *« Germanium transistors: logic circuits in the IBM 1401 computer »* (mars 2021) **http://www.righto.com/2021/03/germanium-transistors-logic-circuits-in.html#fn:diodes**

a) *« The 1950s were a time of rapid change in transistor technology. The transistor was invented at Bell Labs in 1947. General Electric invented the alloy junction transistor (used in the 1401) in 1950. In 1953, the drift transistor was created, faster because of its doping gradient. IBM used drift transistors in the Saturated Drift Transistor Diode Logic (SDTDL) family. The first silicon transistors were introduced in 1954. The wafer-based mesa transistor was invented in 1958, followed by the modern planar transistor in 1959. Thus, transistors were undergoing radical changes in the 1950s and IBM introduced new logic families to take advantage of these new transistor types. »*

b) *« note 4: Many vacuum tube computers used semiconductor diodes as a key part of their logic gates. I think that diodes don't get the recognition they deserve; computer generations are divided into tube versus transistor, without recognizing the gradual introduction of semiconductors in the form of diodes. »*

[31] G. Renard *« La découverte et le perfectionnement des transistors »* Revue d'histoire des sciences 1963 16-4 pp. 323-358 **https://www.persee.fr/doc/rhs_0048-7996_1963_num_16_4_4468**

[32] Walter H. Brattain *« Surface properties of semiconductors »* discours du 11 décembre 1956 **https://www.nobelprize.org/uploads/2018/06/brattain-lecture.pdf**

[33] **https://en.wikipedia.org/wiki/Alloy-junction_transistor**

[34] **https://en.wikipedia.org/wiki/Surface-barrier_transistor**

[35] **http://www.semiconductormuseum.com/Transistors/IBM/OralHistories/Yourke/Yourke_Page2.htm**

[36] **https://en.wikipedia.org/wiki/Transistor_computer**

[37] **https://en.wikipedia.org/wiki/List_of_transistorized_computers**

[38] Commentaires de William Crouse (UPCMaker / Thingmaker) à **https://en.wikipedia.org/wiki/Talk:Diode-transistor_logic**

[39] General Electric *« Transistor Manual (3rd Ed.) »* (1958) **https://worldradiohistory.com/Archive-Catalogs/GE/GE-Transistor-Manual-No.-3---1958.CV01.pdf**

[40] **https://en.wikipedia.org/wiki/Baker_clamp**

[41] **https://ieeexplore.ieee.org/document/1183968**

[42] **http://semiconductormuseum.com/Transistors/IBM/OralHistories/Yourke/Yourke_Index.htm**

[43] **https://en.wikipedia.org/wiki/Emitter-coupled_logic**

[44] **https://en.wikipedia.org/wiki/TX-0**

[45] Jack Ward : *« Historic Germanium Computer Transistors – General Transistor / General Instrument »* 2016 **http://semiconductormuseum.com/PhotoGallery/Photogallery_General_Transistor_General_Instrument_Historic_Germanium_Computer_Transistors.pdf**

voir aussi le livre *« The Supermen - The Story of Seymour Cray and the Technical Wizards behind the Supercomputer »*, 1997, ISBN 0-471-04885-2, ainsi que **https://ethw.org/First-Hand:Building_the_U.S._Navy's_First_Seagoing_Digital_System_-_Chapter_4_of_the_Story_of_the_Naval_Tactical_Data_System#Mr._Seymour_R._Cray.2C_a_Genius_With_Transistors**

[46] Computer History Museum : « CDC 6600 module with 2N709 transistor, circa 1964 » **https://www.computerhistory.org/revolution/digital-logic/12/329/1426** *« This uses the first silicon transistor faster than those made from germanium. Hoerni developed the 2N709 specifically for the CDC 6600 supercomputer. »*

[47] **https://hackaday.io/project/8449-hackaday-ttlers/log/177985-beyond-2ns-with-2n2369a**

[48] Shaos : *« Germanium Vision - Low-resolution video camera with a lot of opened germanium transistors used as light-sensitive elements »* **https://hackaday.io/project/11105-germanium-vision**

[49] *« MULLARD OCP71 PHOTO DIODE, 1960's »* **http://www.museumoftechnology.org.uk/objects/_expand.php?key=1178**

[50] Denis Bodor : *« Modifications matérielles de l'ESP32-CAM »* **https://connect.ed-diamond.com/contenu-premium/modifications-materielles-de-l-esp32-cam** et aussi **https://connect.ed-diamond.com/Hackable/hk-013/module-camera-raspberry-pi-un-mot-sur-l-infrarouge**

VOIR AUSSI

- **https://sites.google.com/site/transistorhistory/Home/** synthétise énormément d'informations très pertinentes.
- Article d'époque sur l'annonce du Transistron : **http://www.tsf-radio.org/forum/im/273714tansistron.pdf**
- Documentaire sur PBS : *« Transistorized! »* 1999 **https://www.youtube.com/watch?v=U4XknGqr3Bo** et **https://www.pbs.org/transistor/index.html**
- Ernst Erb « Erste Transistoren und Transistor-Radios » 18 février 2008, **https://www.radiomuseum.org/forum/transistoren_erste_und_transistor_radios_halbleitertechnik.html** en allemand, traduit en anglais à **https://www.radiomuseum.org/forum/transistors_early_research_first_applications.html**


LE SALON ONE TO ONE MEETINGS DES RÉSEAUX, DU CLOUD, DE LA MOBILITÉ ET DE LA CYBERSÉCURITÉ
IT AND CYBERSECURITY
MEETINGS BY WEYOU GROUP
WWW.IT-AND-CYBERSECURITY-MEETINGS.COM
19, 20 & 21 MARS 2024
PALAIS DES FESTIVALS ET DES CONGRÈS DE CANNES
ILS SONT DÉJÀ INSCRITS
6cure
adista
AISI
allentis
alternativ
APIXIT
aqua
arcserve
ARMIS
Avant de Cliquer
BCS Solutions
betaSystems
betcobe
BeyondTrust
Bitdefender
BlackBerry Cybersecurity
BlueFiles
Blue Trusty
board of cyber
Bouygues Entreprises
bso
CATO
cegedim
clariLog
CLOUDFLARE
COHESITY
conscio
CROWDSTRIKE
CRYPT.ON IT
cybereason
Cyberwatch
CYBLEX
dbSQWare
DATACORE
Delinea
devensys
DIGITAL REALTY
dust MOBILE
EASYVISTA
EATON
efficient IP
EGERIE
EPOS
ERCOM
eset
EV Observe
exabeam
EXCLUSIVE NETWORKS tenable
EXCUBE
FILIGRAN
Forcepoint
FORTINET
GATEWATCHER
GitGuardian
CONVERGENCE
HarfangLab
HEXANET
HUMAN
ilex
INFINIGATE
IRON MOUNTAIN
ITS Ibelem
Jabra GN
juniper
KADISKA
KDDI France
Linkt
Lutessa
MAKE IT SAFE
Malwarebytes
ManageEngine a division of Zoho Corp.
MANHATTAN
MIEL
MY
nameshield
NeoYAD
netskope
NOZOMI NETWORKS
oeil pour oeil
okta
ONE IDENTITY
opentext Cybersecurity
Orange Cyberdefense
OVERSOC
OWN
Patrowl
PENTERA
Ping Identity
platform.sh
pradeo
PRIMX
proofpoint
PURESTORAGE
pythéas
Qualys
radware
RingCentral
Riot
rubycat
SailPoint
sea tpi
SentinelOne
sewan
Skyhigh Security
SNS SECURITY
SOPHOS
sosafe
STORMSHIELD
Suricate
synetis
TANIUM
tdf
TENTRIS
TELEHOUSE
Tenacy
TREND MICRO
TrustHQ
TrustBuilder
TYREX
UBIKA
vade
VARONIS
wavetel
WITH
xmco
ZeroTrust
Zerto a Hewlett Packard Enterprise company
zpe
by weyou group
Professional Exhibitions and One to One Meetings Exhibitions