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, 23 août 2023

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'inter-comparaisons 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 Module4 permettent d'atteindre ce résultat.

1 Introduction

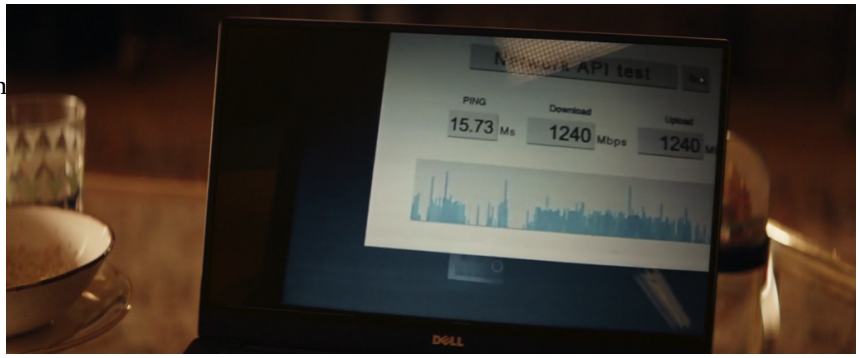


FIGURE 1: Extrait du film *The Hummingbird Project* qui illustre la débauche de moyens pour réduire les délais de communication dans les transaction boursières. Vincent @ 5820 s : "it's not the destination that's important, it's the people we meet and the lessons we learn". Et pourtant, ils ont réussi 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.

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 $\sin(\varphi)$ avec $\varphi = 2\pi f\tau$ la phase du signal injecté donc son retard τ à fréquence f), de la synchronisation des réseaux de communication 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 radiotéléscopes 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 fréquence ultrastables inaccessibles au commun 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 récepteur 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 terme 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 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.

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 possibles 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 **synthonisation**, i.e. 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 $f = 1/t \Rightarrow \frac{df}{f} = -\frac{dt}{t}$) 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 $400\cdot 10^3/v \simeq 2000$ 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 sub-microseconde 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 émetteur et récepteur [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 quelques 67 ms (au zénith, plus à l'horizon) que met l'onde électromagétique à parcourir les quelques 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 pseudo-ranges (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 synthonisation, 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 terme 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 analogique-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 déclenchée par un tel signal.

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 tel que 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 fichier permet de stocker 9 décimales qu'elles le sont. Ainsi nous avons découvert que les opérations sur les fichiers (`cp`, `mv` ...) dans Busybox met 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
```

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

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
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 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

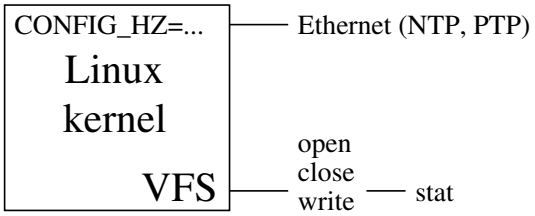


FIGURE 2: Couches d'abstractions intervenant dans le problème avec les divers oscillateurs physiques cadençant les périphériques et le processeur

```
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)
```

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.

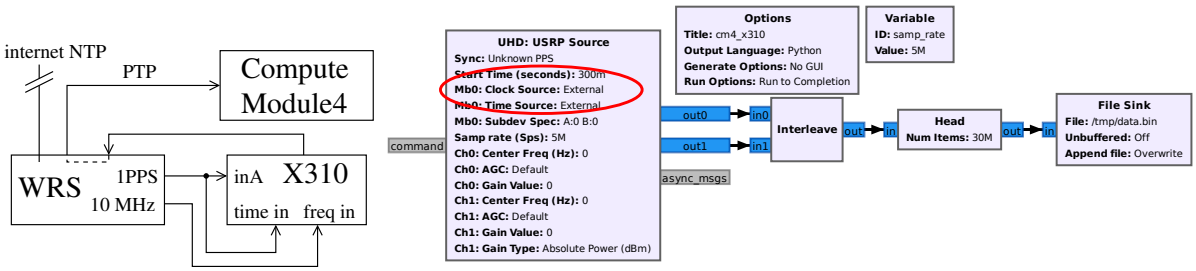


FIGURE 3 – Gauche : principe de la mesure, avec un récepteur de radio logicielle X310 connectée par Ethernet à une Raspberry Pi Compute Module 4, aussi connectée au travers de cette interface via un switch GbE à un serveur de temps PTP White Rabbit asservi sur maser à hydrogène, et à internet via une 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).

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 observable 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 dent 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,

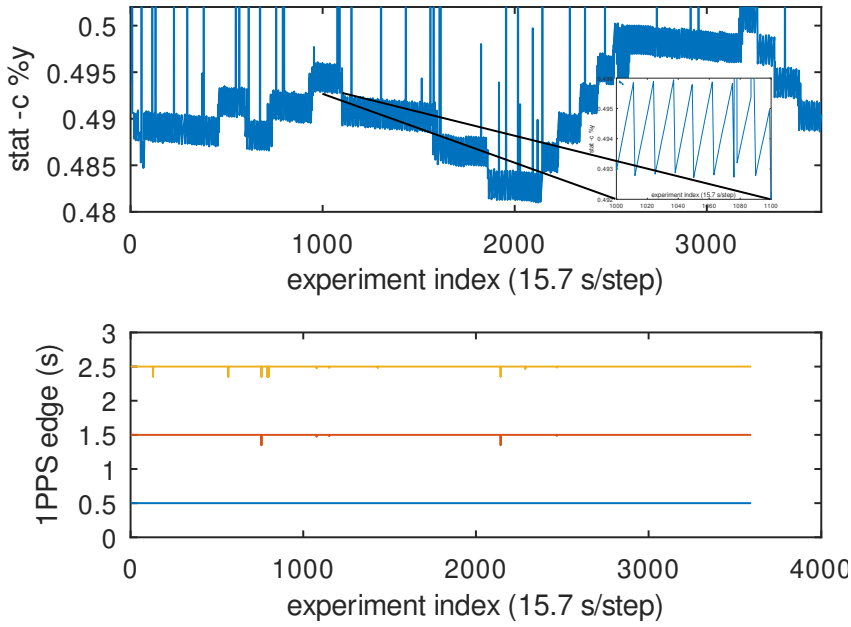


FIGURE 4 – Haut : date de création des fichiers acquis de la X310, démontrant les fluctuations long terme de fréquence que NTP peine à corriger. En insert, zoom sur une centaine d'acquisitions illustrant le motif en dent de scie induit par la granularité du compteur cadencé dans le noyau Linux pour fournir 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.

— 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 dent 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.

Ce motif en dent de scie rappelle immédiatement celui de la sortie 1-PPS des récepteurs GPS Motorola tel que 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 dent 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'alligner à nouveau sur le temps NTP. Cette procédure est caractéristique de **synchronisation** (ajout de retards) en l'absence de **synthonisation** (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 encapsulés 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 tel que 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'oscilateurs 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)
...
```

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).

`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 patchs sur `https://`

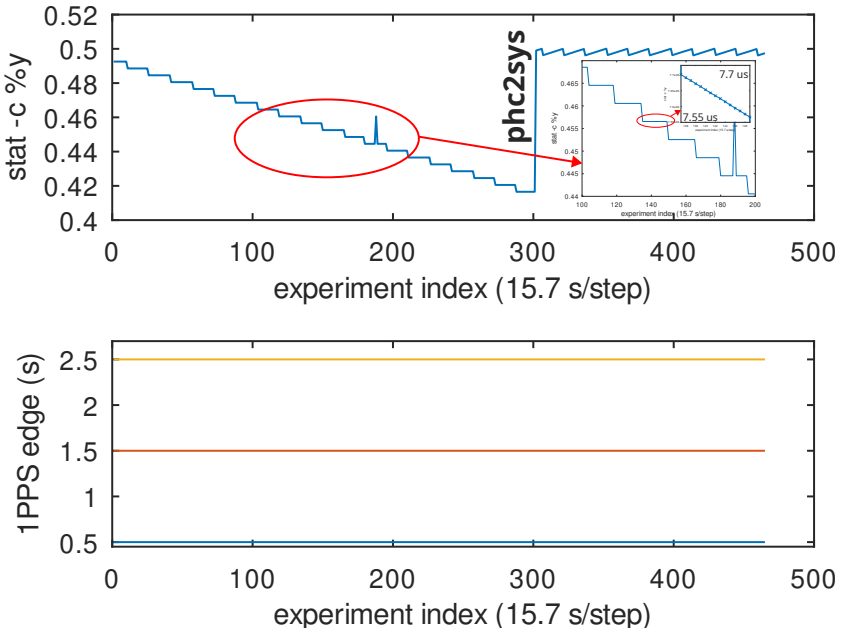


FIGURE 5 – Transfert du temps PTP observé par l'interface Ethernet vers le système en activant le daemon `phc2sys`.

`github.com/richardcochran/linuxptp/commit/2a2532d66121d0060b042c5bd6020a62153f1e0a` et 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.*" donc 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 paliatif logiciel à un problème connu, même si d'origine matériel, semble aussi incompréhensible qu'improductif et nécessite pour le moment la modification manuelle pour les utilisateurs de CM4 au moins.

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 se recaler sur le temps transmis par PTP, toujours limité par la granularité du compteur qui cadence le noyau. Ainsi, nous retrouvons les motifs en dent de scie que nous avions observé avec NTP, mais avons cette fois éliminé les fluctuations long terme avec une courbe fluctuant peu au cours du temps autour du motif en dent 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 validée lorsque `date` ne fournit plus le premier janvier 1970 mais la date actuelle fournie par le Grand Master 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 s'appuyant sur la couche Ethernet (`-2`) au lieu de la couche IP. Par ailleur, 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 dent de scie subsiste. Sachant que ce motif en dent 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 dent 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 cœur de processeur multiplié par cette fréquence, et que pour un système répondant rapidement aux solicitations 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` recompile le noyau et `make` place `output/images/Image` de l'hôte que nous pourrons copier dans le première partition de la carte SD de la cible pour rebooter sur le nouveau noyau. Nous observons les résutats en Fig. 6 : les dents de s'atténuent avec une fréquence croissante de 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

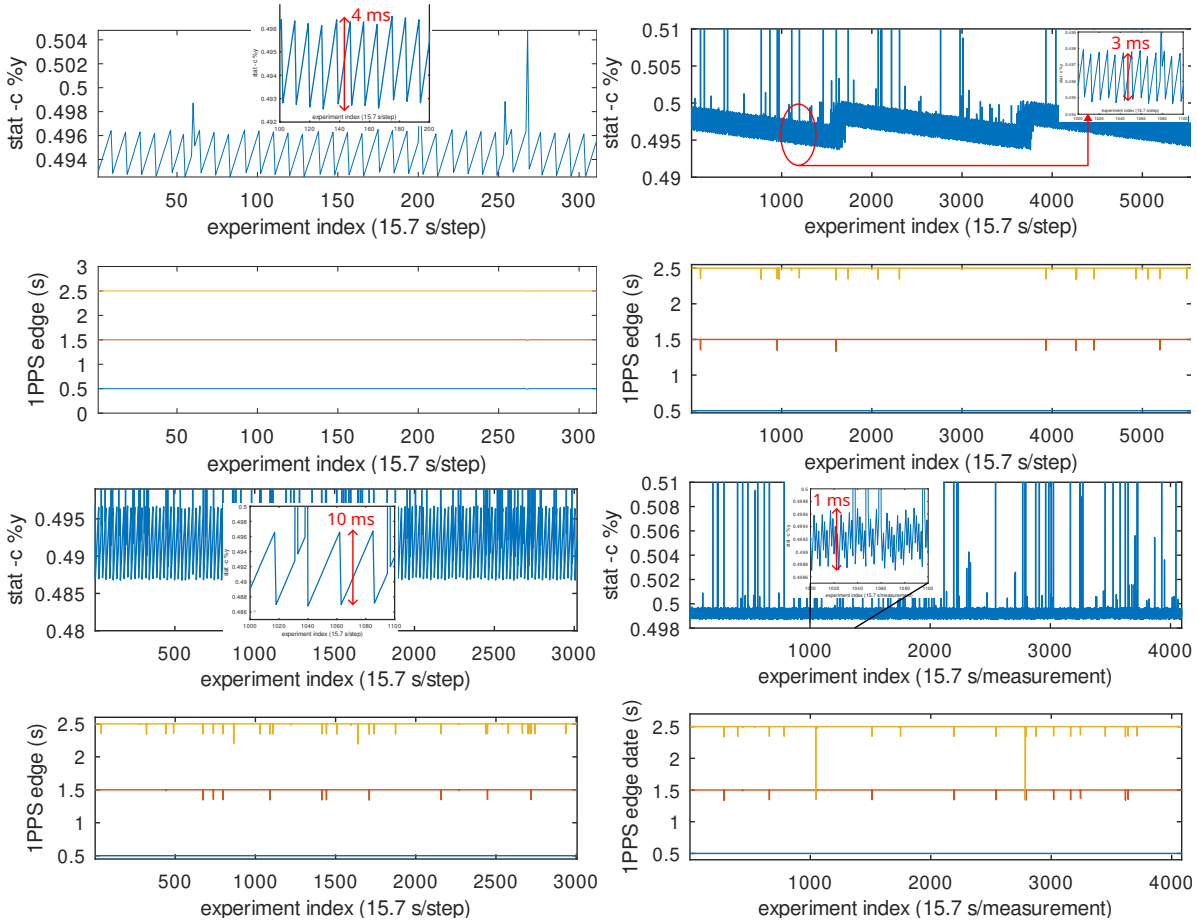


FIGURE 6 – Quatre mesures successives de la datation des fichiers acquis par la X310 en fonction de la configuration du noyau en sélectionnant un timer à 250 Hz, la valeur par défaut dans le noyau Linux (haut-gauche), 300 Hz (haut-droite), 100 Hz (bas-gauche) et 1 kHz (bas-droite). Pour chaque courbe, 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.

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).

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 synthonisation – asservissement de la fréquence – à la synchronisation déjà fournie par PTP. L'étude du circuit imprimé indique que seuls

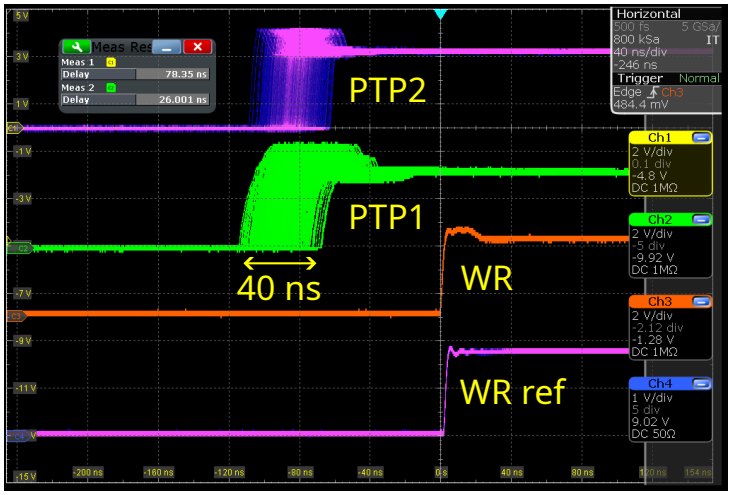


FIGURE 7 – Capture d'écran des fronts montants des signaux 1-PPS déclenché sur un signal issu d'un switch White Rabbit (WR ref) de référence (violet en bas). Un second switch White Rabbit (WR) séparé par environ 2 km de fibres optiques du premier présente une gigue d'une soixantaine de ps (courbe orange). 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.

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, voir 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.

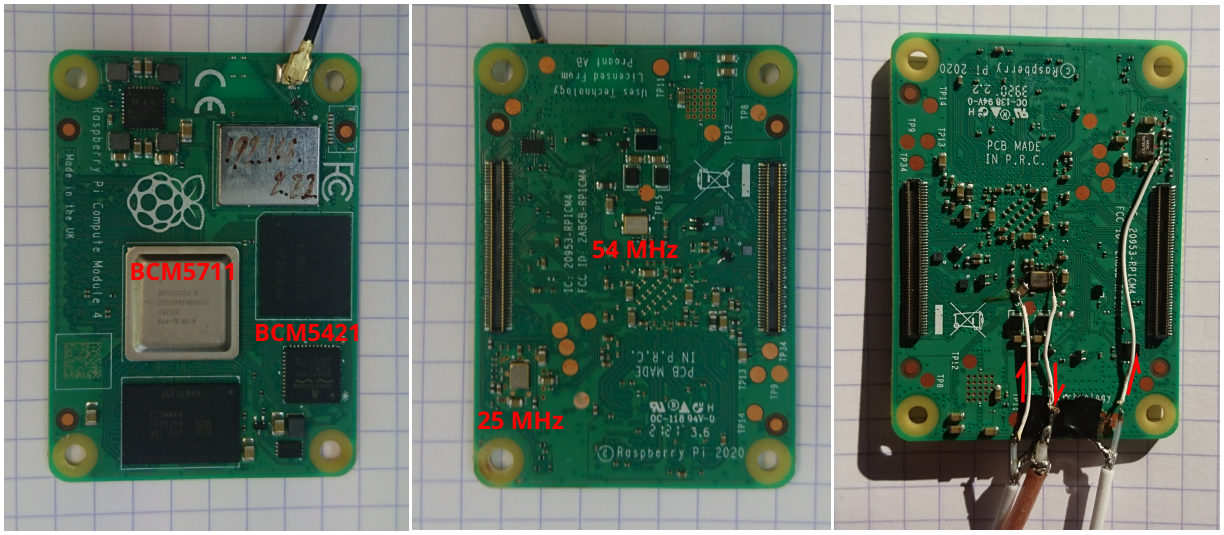
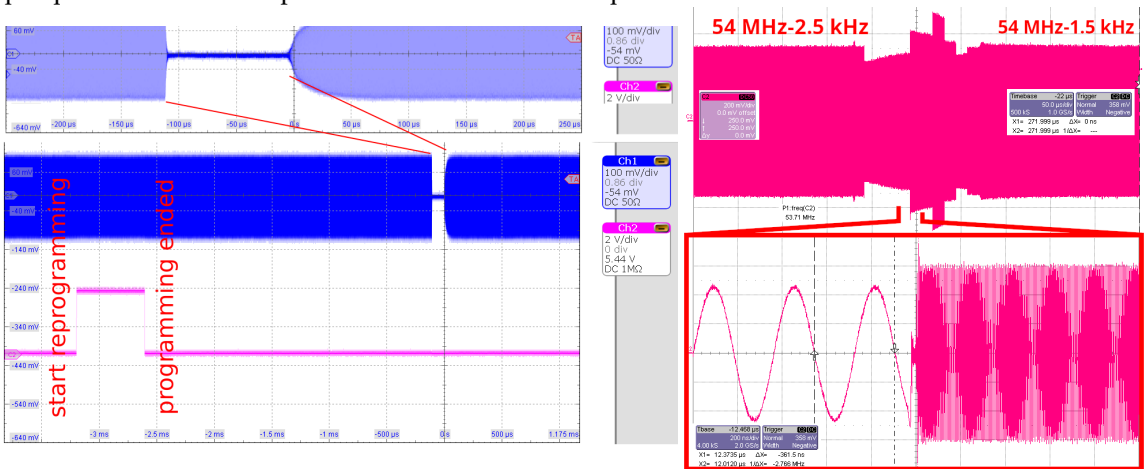


FIGURE 8 – La CM4 (gauche) est une plateforme idéale pour l'analyse du transfert de temps entre ordinateurs car elle n'est munie que de deux sources de cadencement (milieu), un résonateur à quartz à 54 MHz qui cadence le processeur BCM5711 et un résonateur à quartz à 25 MHz pour cadencer l'interface physique Ethernet BCM5421. Droite : quelques points de test pour observer ou imposer la fréquence des oscillateurs. Noter 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 synthétiseur 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'ajuster 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 transitoirs à quelques MHz suivis de périodes à 200 MHz avant de se stabiliser sur la valeur finale. Le processeur est incapable de s'accomoder 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 (e.g. 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 proposé à `https://github.com/jmfriedt/gr-ad9954_rpi/` mais son utilisation s'est révélée inadéquate pour ce projet.

Nous commençons ainsi dans un premier temps par une expérience de laboratoire consistant à remplacer les deux résonateurs – 25 et 54 MHz – par deux synhtétiseurs programmabes émettant une puissance de +4 dBm à leurs fréquences respectives (Fig. 9, droite). 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), voir que cette dérive s'annule lorsque nous

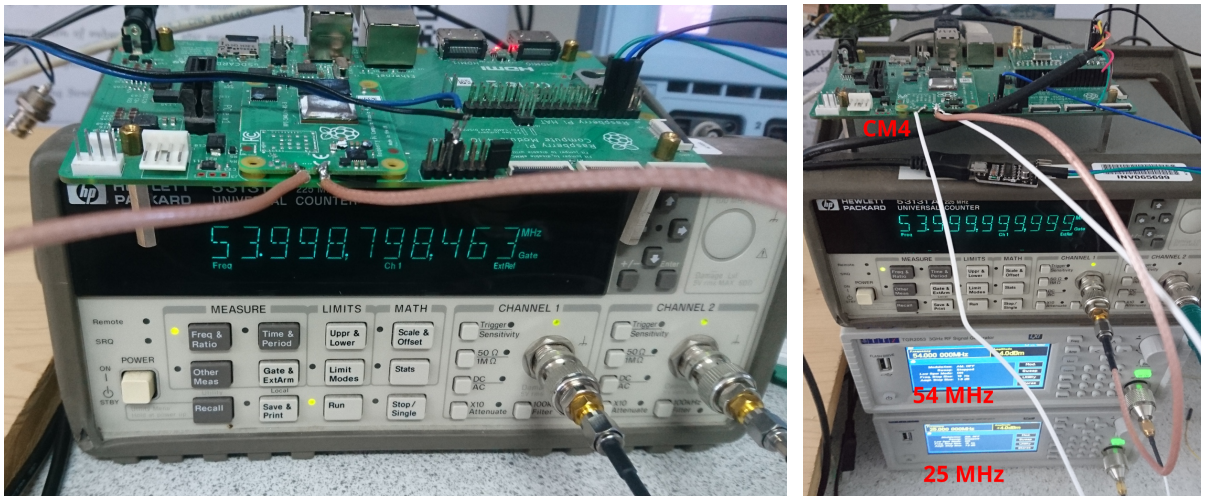


FIGURE 9 – Gauche : mesure de la fréquence de l'oscillateur cadençant le processeur, de fréquence nominale de 54 MHz mais en pratique décalé de -1,3 kHz environ. Droite : les fréquences cadençant le processeur et l'interface Ethernet sont imposées par des synthétiseurs contrôlés par la même source de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

imposons exactement 54 MHz (Fig. 10, droite).

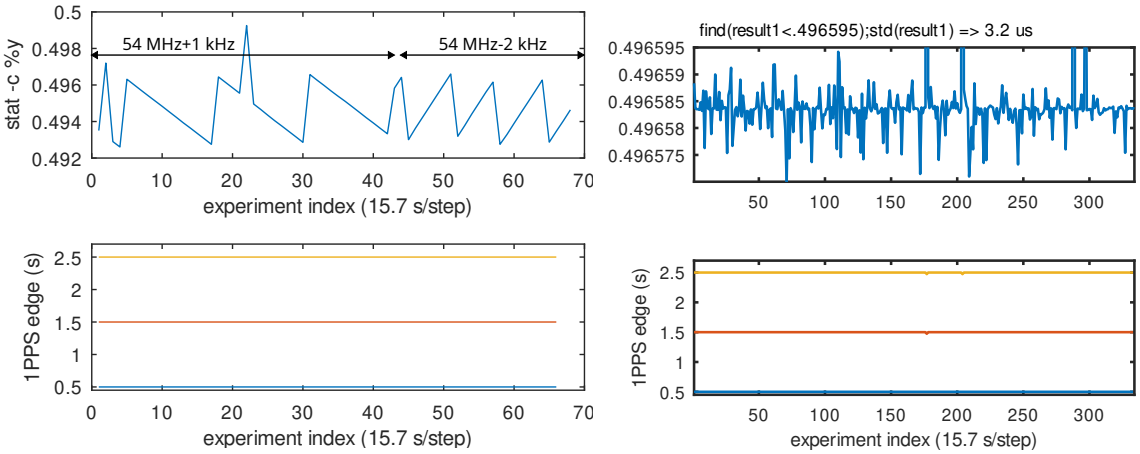


FIGURE 10 – Gauche : impact de la fréquence cadençant le processeur sur le motif en dent de scie, montrant clairement que la pente est déterminée par l'oscillateur qui cadence le processeur. Nous avons vérifié que la fréquence qui cadence l'interface Ethernet n'impacte pas cette mesure. Droite : perspectives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

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 systèmes 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 synthonisation, 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 dent 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 possibles 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, significant 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 $[2\pi]$ 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.

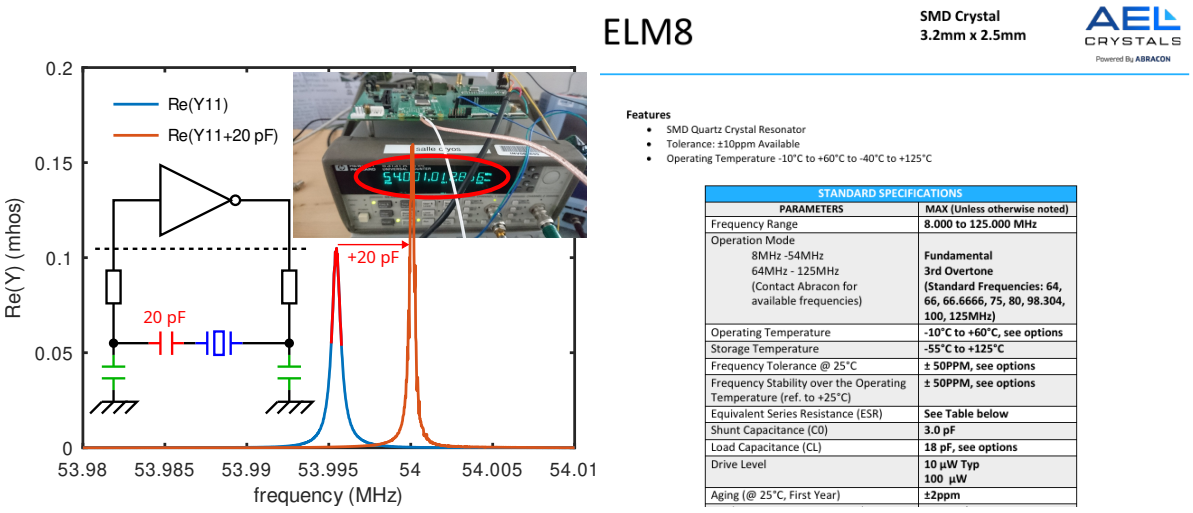


FIGURE 11 – Gauche : caractérisation à l'analyseur de réseau vectoriel du résonateur AEL sur son mode fondamental à 54 MHz (bleu) et largeur à mi-hauteur permettant d'identifier son facteur de qualité, et simulation d'une capacité de 20 pF en série pour amener la résonance à sa valeur nominale de 54 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'ampificateur 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-deça** 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).

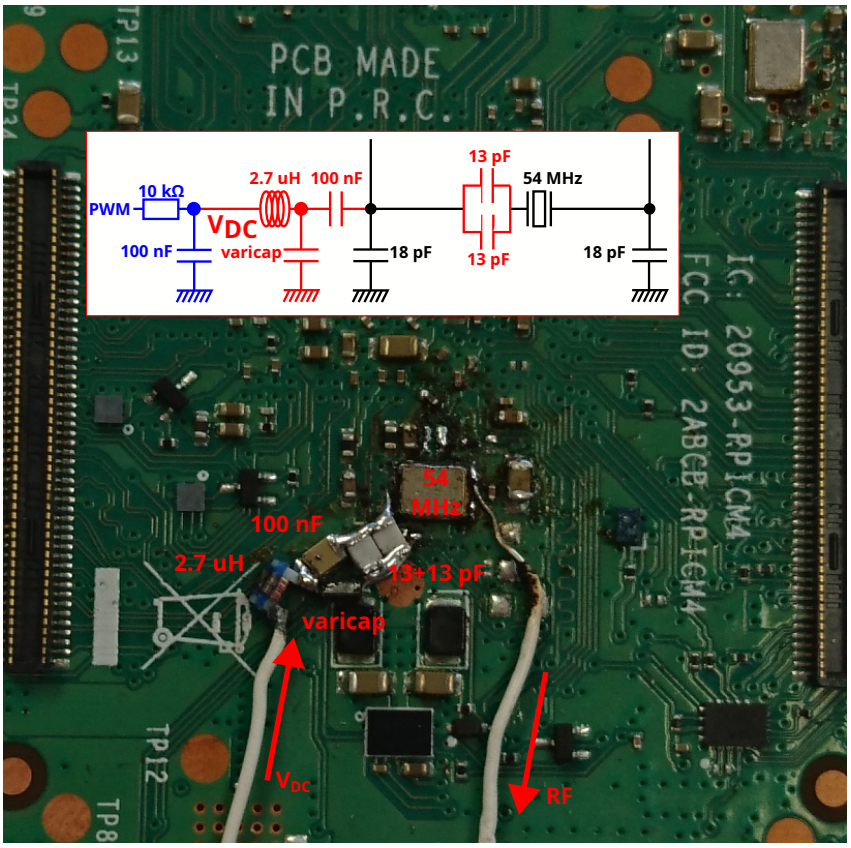


FIGURE 12 – Ajout de la varicap (rouge) entre le résonateur 54 MHz et la capacité de pied de valeur nominale 18 pF (noir). La varicap est polarisée par une tension DC issue de la PWM filtrée (bleu) mais afin de protéger la porte inverseuse, une capacité de grande valeur (100 nF ou une impédance de 0,03 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.

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 relentissement 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 com-

pliqué (un module Peltier est encombrant et nécessite un radiateur sur sa face chaude) : nous constatons que l'orientation cristalline sélectionnée par le fabriquant 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.

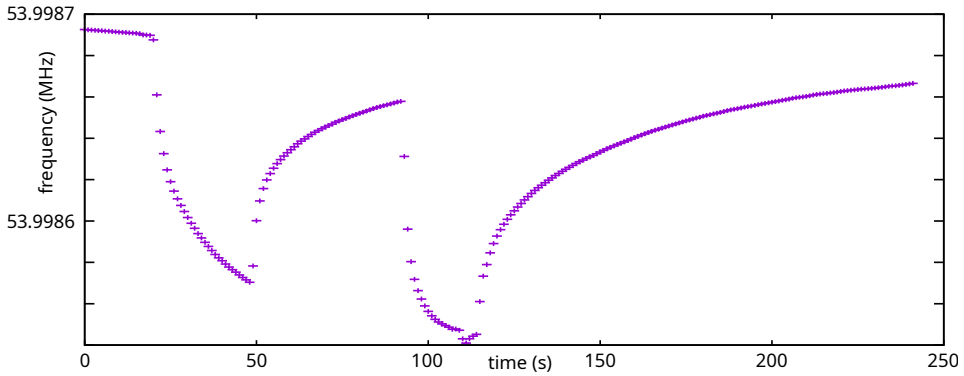


FIGURE 13 – Impact de l'échauffement du résonateur à quartz à 54 MHz sur sa fréquence en plaçant à deux reprises (dates 15 et 90) un fer à souder sur la face supérieure de la CM4 au niveau de la feuille de la framboise la plus proche du processeur : la chaleur diffuse à travers le circuit imprimé et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

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 affectée 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).

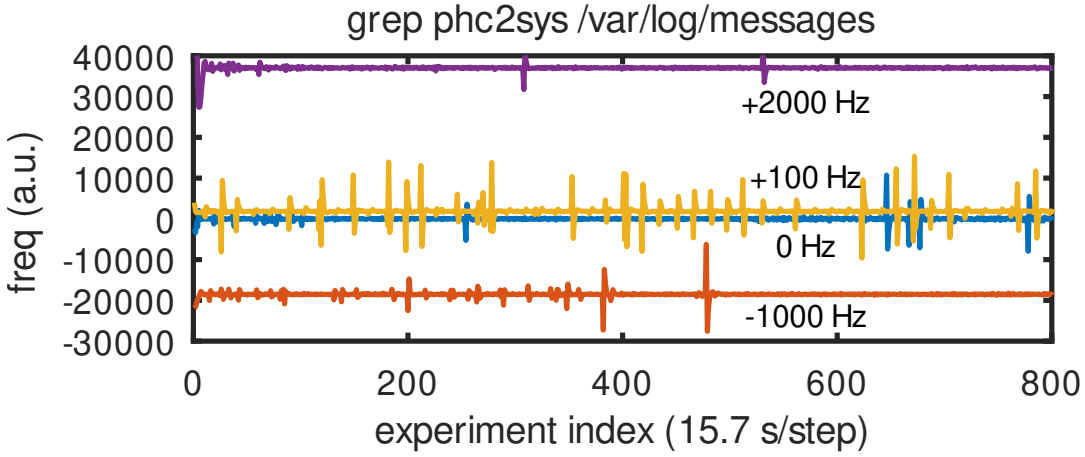


FIGURE 14 – Information de fréquence fournie dans le fichier de `messages` dans les `logs` en fonction de l'écart de fréquence de cadencement du processeur à sa valeur nominale de 54 MHz.

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 servi 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.

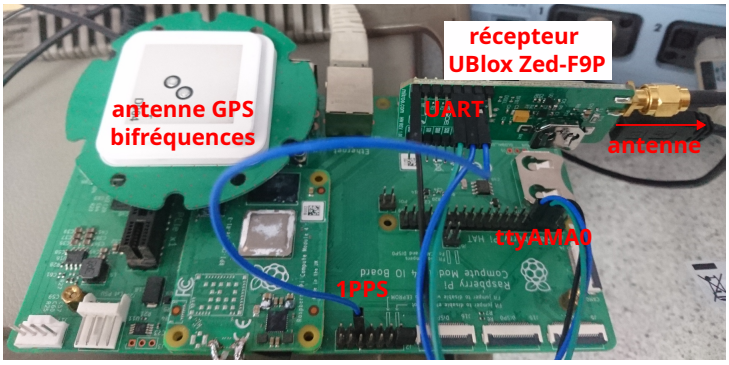


FIGURE 15: Montage final : un récepteur UBlox Zed-F9P sur carte MIKROE-4456 est connecté par UART pour transmettre date et heure à la CM4 tandis que l'impulsion horaire est transmise au PHC dont la broche est cette fois configurée en entrée.

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. Cependant, nous désirons conserver la console UART de la CM4 nommé `ttyAMA0` pour déverminer 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 lequel ce résulat 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 à 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, graĉe 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 la 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 $(53999625-54\cdot10^6)/54 = -6.9$ 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. Noter dès à présent que l'asservissement proportionnel-intégral de `-E pi` sera source d'ennuis et sera remplacée finalement par `-E linreg`.

6 Synthonisation 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 numérique-analogique, 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 f_{PWM} est un compromis entre la capacité à filtrer la sortie oscillante pour produire un signal de commande continu, et la résolution de la commande 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).

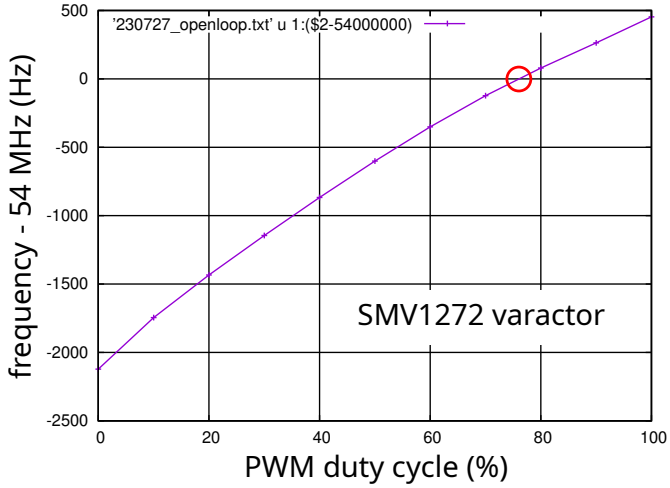
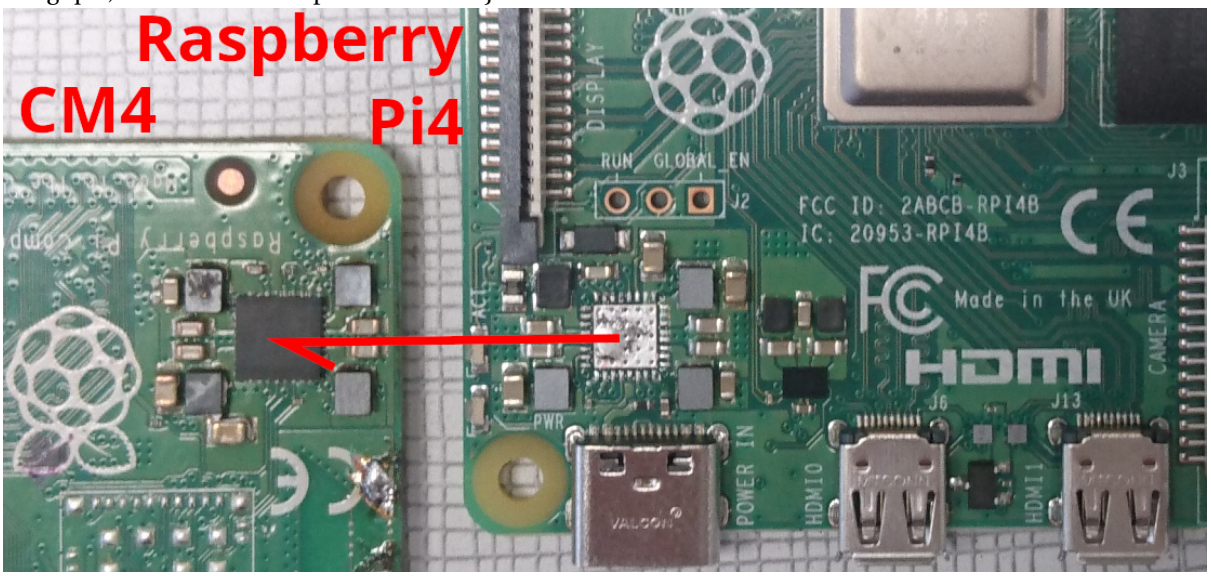


FIGURE 16 – Relation fréquence-tension pour un rapport cyclique de la PWM variant de 0% (0 V) à 100% (3,3 V) en utilisant une varicap Skyworks SMV1272. Le point de fonctionnement recherché est mis en évidence par le cercle rouge lorsque la fréquence de l'oscillateur est 54 MHz. La fréquence de l'oscillateur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

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 coeur∧w PMIC 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 ressuciter cette dernière après une manipulation malheureuse.

Avant tout asservissement, nous validons la capacité de tirage de l'oscillateur en boucle ouverte. Pour ce faire, la tension de commande étant limité 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 modèle, mais devra être re-évaluée par chaque CM4, la gamme de tirage étant relativement réduite néanmoins. 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 4IO 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.*".

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

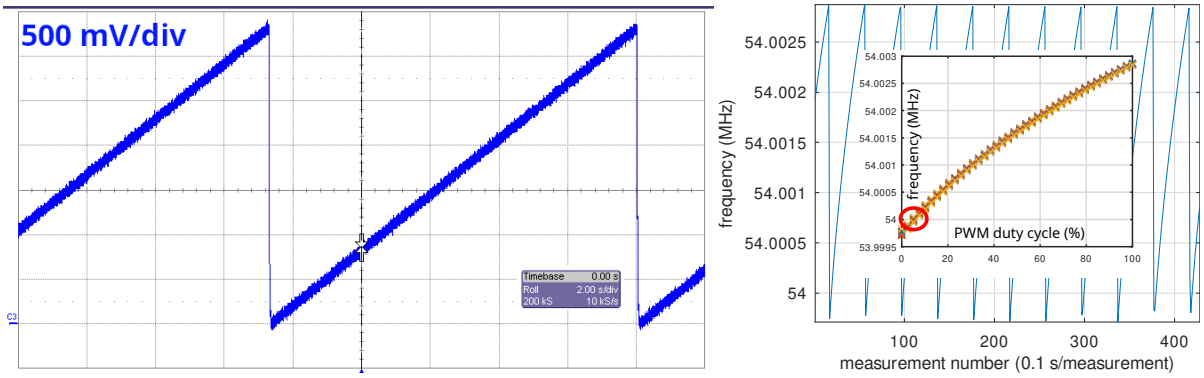


FIGURE 17 – Gauche : capture d'écran d'oscilloscope représentant les rampes de tension en sortie de PWM configurée avec une période de 10 μs (fréquence 100 kHz) et un rapport cyclique variable par le script bash `while true; do for i in ‘seq 0 10000 1000000‘; do pigs HP 18 100000 $i;sleep 0.1;done;done` 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.

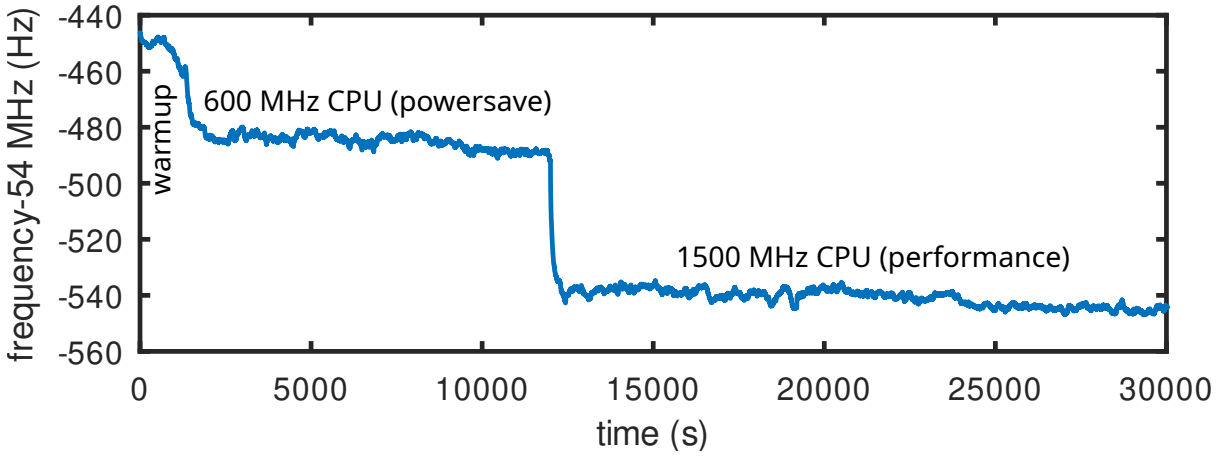


FIGURE 18 – Échauffement du circuit imprimé de la CM4 lors de sa mise sous tension (*warmup*) induisant la dérive de l'oscillateur, puis échauffement accru lors de l'augmentation de la fréquence de cadencement du processeur configuré autour de l'abscisse 12000 pour passer du mode `powersave` à 600 MHz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

conséquent. La principale subtilité tient aux protections contre les dysfonctionnements (messages autres que ceux de `phc2sys`), contre les valeurs aberrantes, et contres 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

```
1  command=89770  # initial setpoint
2  pigs HP 18 100000 ${command}
3  mesm4=0;mesm3=0;mesm2=0;mesm1=0
4  pair=1;sumpair=0
5  n=0
6  Kp=0.1;Ki=0.1
7  moy=0;cmdi=0
8  while true; do
9   d=‘date +%s‘ ; echo -n $d "␣"
10  df=‘tail -30 /var/log/messages | grep phc2sys | tail -1 | cut -dq -f2 | cut -dd -f1‘
11  if [[ $df -ne $mesm1 ]] ; then
```

```
12   moy=$((moy-(mesm4)));moy=$((moy+(df)))  # proportional x[n+1]-x[n]
13   mesm4=$mesm3;mesm3=$mesm2;mesm2=$mesm1;mesm1=$df
14   sumpair=$((sumpair+df))
15   if [[ $pair -eq 2 ]] ; then
16    pair=0
17    if [[ $sumpair -lt 50000 ]] &&  [[ $sumpair -gt -50000 ]]; then
18     cmdi=‘echo "${sumpair}*16.632/4*${Ki}" | bc | cut -d\. -f1‘
19     sumpair=0
20    fi
21   fi
22   pair=$((pair+1))
23   cmdp=‘echo "${moy}*16.632/4*${Kp}" | bc | cut -d\. -f1‘
24   if [ -n "$cmdp" ] && [ "$cmdp" -eq "$cmdp" ] && [ -n "$cmdi" ] && [ "$cmdi" -eq "$cmdi" ] ; then
25    if [[ $n -gt 4 ]] ; then      # ^^^ verifie que cmdp et cmdi sont bien des nombres
26     command=$((command-(cmdp)-(cmdi)))  # PI controller
27     n=5
28    else
29     n=$((n+1))                  # attend de remplir le tampon
30    fi
31   fi
32   echo $n $mesm1 $mesm2 $mesm3 $mesm4 $cmdp $cmdi $command
33   if [[ $command -gt 0 ]] && [[ $command -lt 1000000 ]];  then
34    pigs HP 18 100000 ${command}  # applique la commande a la PWM
35   fi
36  fi
37  sleep 5
38 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, i.e. 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 `mesm`i 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 raffraîchissement 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 paire 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).

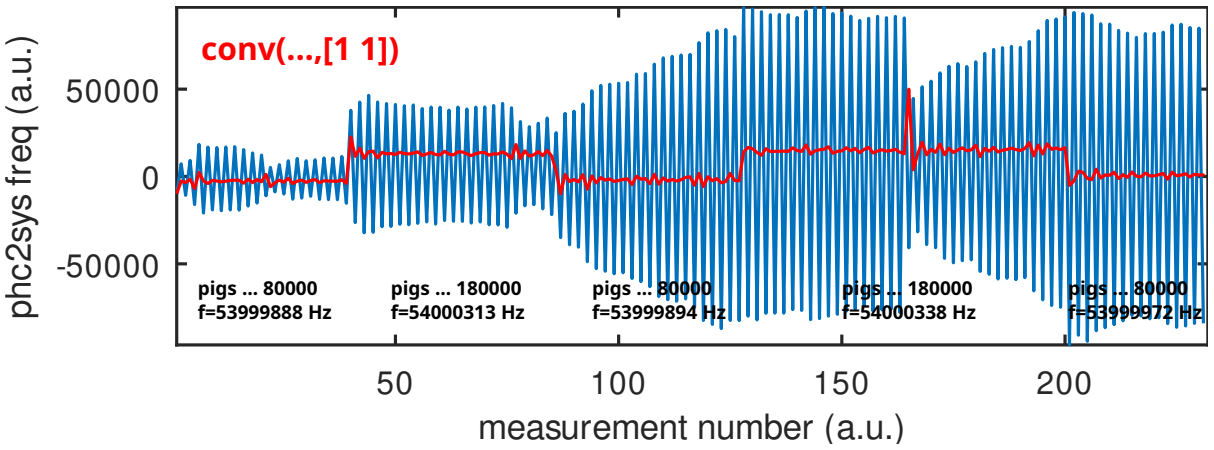


FIGURE 19 – Évolution du champ `freq` des messages de `phc2sys` dans `/var/log/messages` (bleu) et moyenne des valeurs adjacentes (rouge). Le texte en dessous de la courbe indique la commande de la PWM définissant la tension qui polarise la varicap, et la fréquence résultante de l'oscillateur observée. 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.

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 intégrale, mais au moins on aura un résultat (Fig. 20).

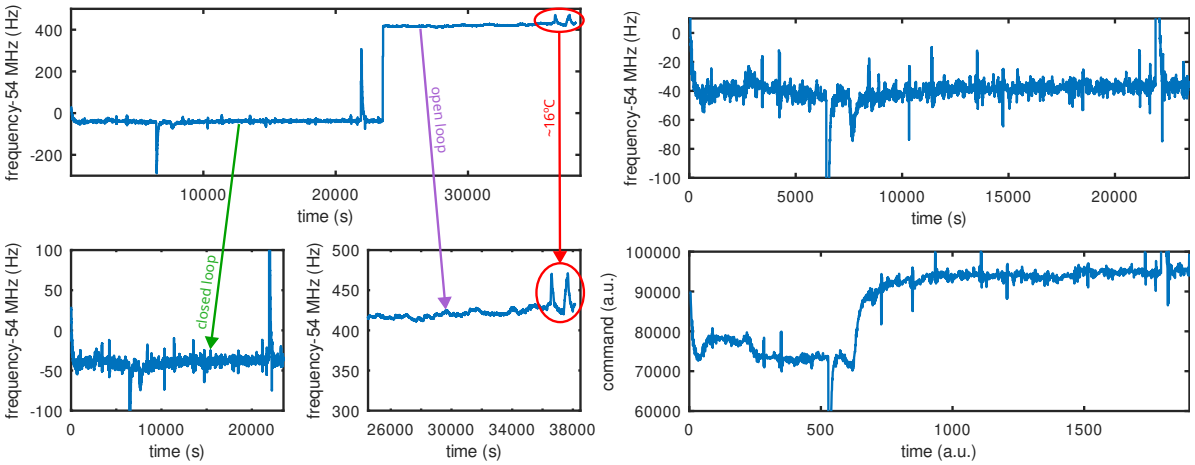


FIGURE 20 – Gauche : en haut l'évolution de la fréquence de l'oscillateur qui cadence le processeur, au début en boucle fermée puis en boucle ouverte quand le programme d'asservissement a cessé de fonctionner. Les deux sauts à droite de la courbe sont dûs à une fenêtre ouverte exposant la CM4 à une baisse 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.

De cette première mesure nous apprenons que

— chaque mesure individuelle est excessivement bruitée, mais les mesures prises deux par deux compensant 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 10^5 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,

— que les corrections à **court terme**, une fois toutes les 15 secondes environ, au rythme du raffraîchissement 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 telle 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.

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, $n \in \mathbb{N}$ 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'erreur et la colonne 7 la borne supérieure. Noter que `uncertainties` fournit le script `gnuplot` pour atteindre le même résultat.

Le deuxième point est corrigé en remplaçant le correction `-E pi` de `phc2sys` par `-E linreg` qui stabilise l'estimation de `freq` et la maintient stable pendant plusieurs heures, tel que 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 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 à 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

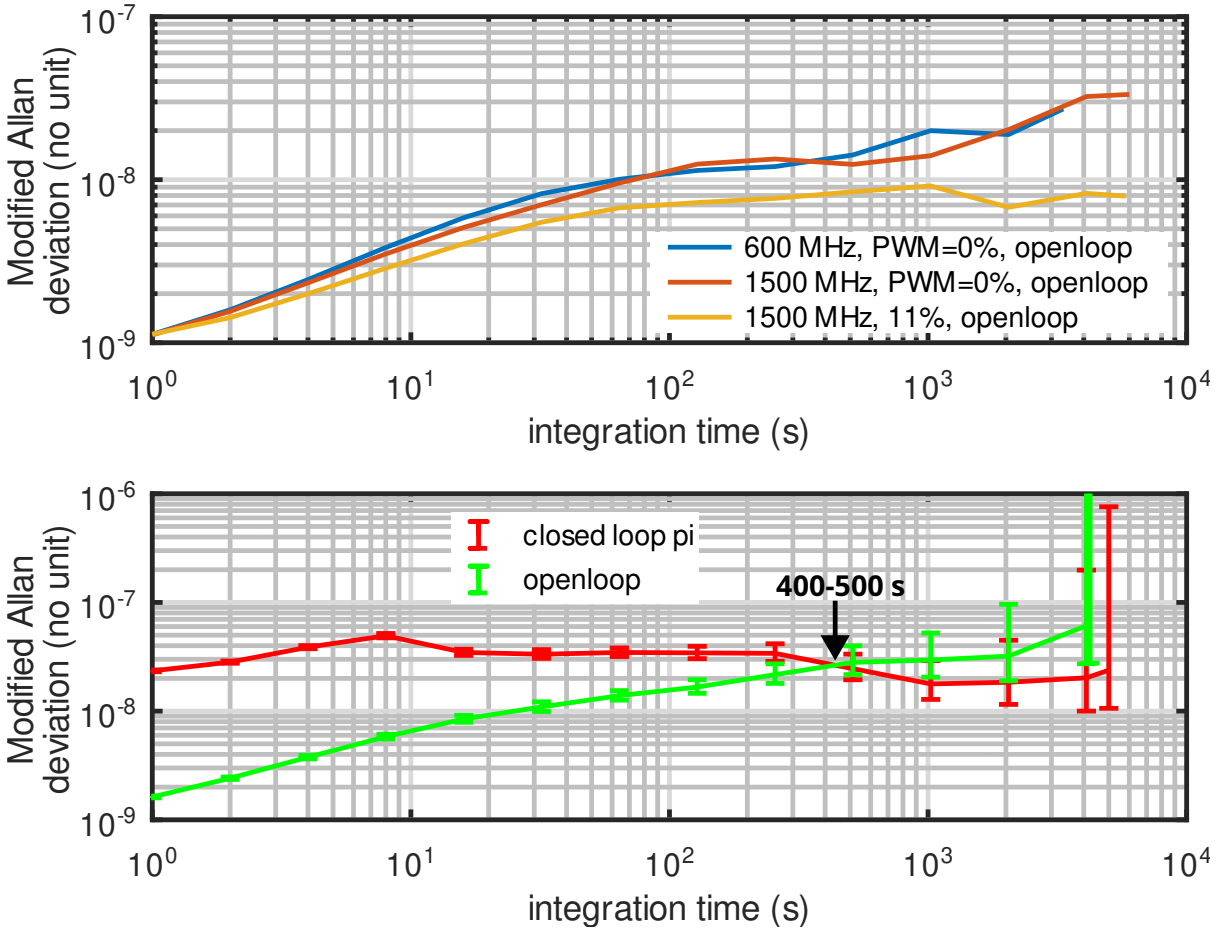


FIGURE 21 – Variance d'Allan de l'oscillateur en boucle ouverte en haut pour diverses configurations, que le processeur soit cadencé à 600 MHz ou 1500 MHz, avec une commande nulle ou une PWM à 11% de rapport cyclique, qui ne semble avoir aucun impact significatif sur la stabilité. En bas : comparaison 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 à la millier de secondes (flèche verticale).

fait d'une correction trop rapide polluée par une mesure grossière de la fréquence par `phc2sys` et 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'inégration 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 I^2C ne pourrait qu'être bénéfique pour réduire le pas de correction. Il est fort probable qu'un lecteur compétent au 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.

7 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 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 cadence le noyau, et identifié la néces-

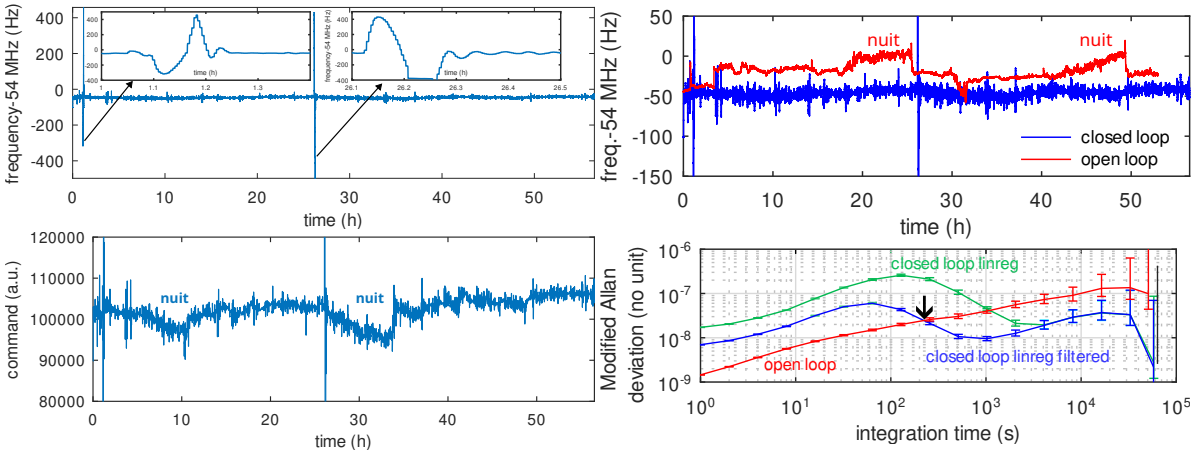


FIGURE 22 – Gauche : en haut la mesure en boucle fermée de l'évolution temporelle de la fréquence de l'oscillateur au cours de plus de 48 h pour mettre en évidence le bénéfice de l'asservissement, et en bas la commande appliqueée à la PWM pour polariser la varicap. Les deux dérives soudaines de fréquence 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 indiqué par la flèche noire verticale.

sité de synthoniser 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.

Au delà de la solution matérielle de synthonisation proposée, nos explorations de la gestion du temps par le noyau Linux nous ont amené 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.

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 résonant, 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 $Q = \frac{1}{R_1}\sqrt{\frac{L_1}{C_1}}$ et la fréquence de résonance $f = 1/(2\pi\sqrt{L_1 C_1})$ 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 C_1 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.

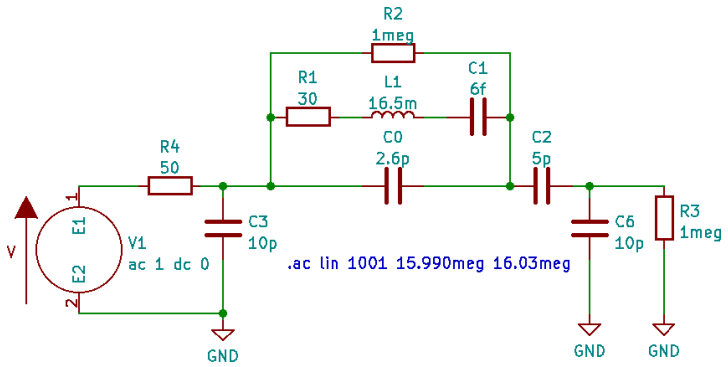


FIGURE 23 – Simulation d'un modèle de résonateur à quartz de type Butterworth-van Dyke RLC//C dans un circuit d'oscillation comprenant les deux capacités de pieds C_3 et C_6 et un condensateur de tirage en série avec le résonateur C_2 La résistance R_2 est incluse, comme souvent dans le vrai oscillateur, 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 mesure 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. 23). 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 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 fe-

nêtre 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. 24).

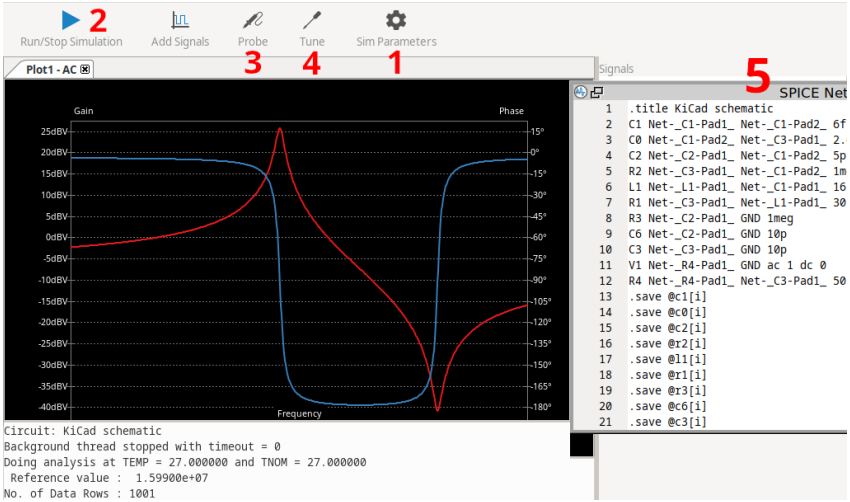


FIGURE 24 – Résultat de la simulation après avoir chargé les paramètres depuis le texte libre dans le circuit (1), lancé la simulation (2), choisi le signal à sonder (3) au dessus de R_3, et potentiellement choisi de faire varier C_2 avec le tournevis (4).

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. 24, 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. 25, 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 dispositif 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 exemples 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

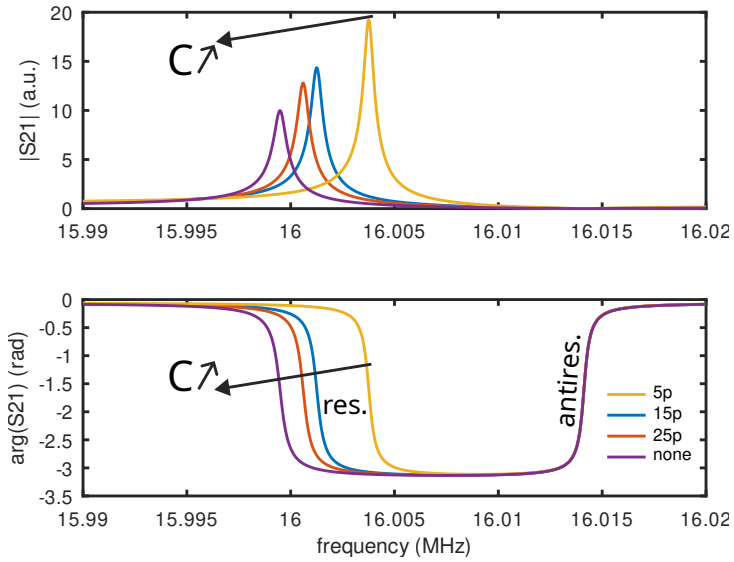


FIGURE 25 – Résultats de la simulation pour diverses valeurs de C_2, avec en haut le module de la tension et en bas sa phase, à gauche la résonance sensible à la valeur de C_2 et à droite l'anti-résonance insensible à C_2. La largeur de la résonance déterminée par R_1 fournit le facteur de qualité du résonateur.

contient que 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 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'accomoder d'un fichier S*n*P, 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 menu. 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. 26 (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 fréquence 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. **Il est inutile, voir 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 decibels 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 ayons 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. 26 est en parfait accord avec la mo-

délisation 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.

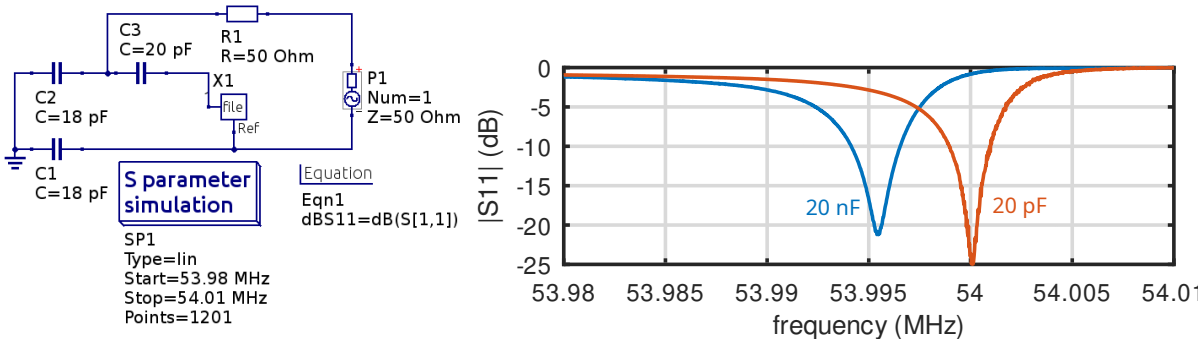


FIGURE 26 – Gauche : schéma de la modélisation par Qucs du tirage capacitif du résonateur dont la réponse a été mesurée et sauvegardée en fichier S1P. Droite : résultat de la simulation pour une capacité de tirage en série avec le résonateur de 20 pF et de 20 nF s'apparentant à une capacité infinie et fournissant donc presque la réponse libre du résonateur.

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.

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 HS (Fév. 2019)

[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-Dec. 2022)

[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 particlier 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)