DÉC. 2014 / JAN. / FÉV. 2015 N°13

Open Silicium

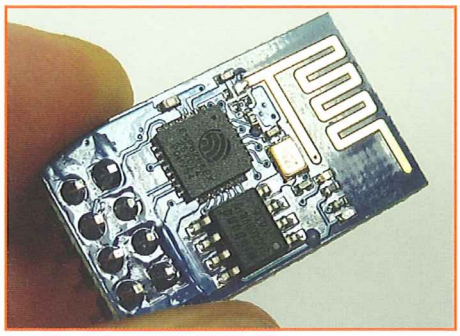
MAGAZINE

INFORMATIQUE
OPEN SOURCE
EMBARQUÉ
INDUSTRIEL ET R&D

LE MAGAZINE 100 % TECHNIQUE, 100 % PRATIQUE, 100 % EMBARQUÉ !

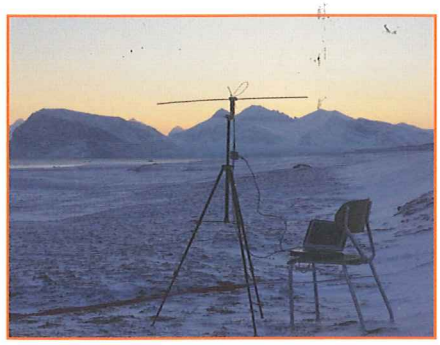
RÉSEAU / WIFI

Ajouter simplement une connectivité Wifi AP+STA et TCP/IP avec un module ESP8266 à trois sous p.76



SDR / SATELLITES

La réception de signaux satellites NOAA, GPS et ISS avec un adaptateur DVB-T p.46

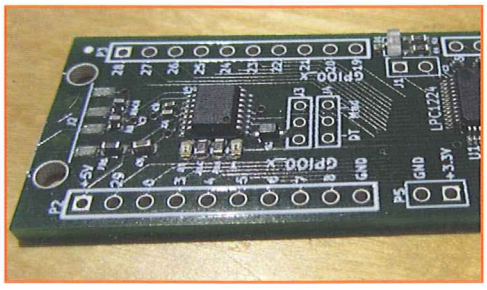


RPI / CLUSTER

Découvrir la parallélisation des compilations avec distcc et un cluster de Raspberry Pi p.04

PCB / FABRICATION

Passez votre projet de la conception à la réalisation de circuits professionnels : l'exemple domotab p.64



OUTILS / UNIX

Leçon de choses ou comment quelques lignes de codes peuvent devenir un cauchemar avec sed p.40

REPÈRE / NET

Tout ce que vous devez savoir sur les entrailles et le fonctionnement des protocoles TCP/IP p.08

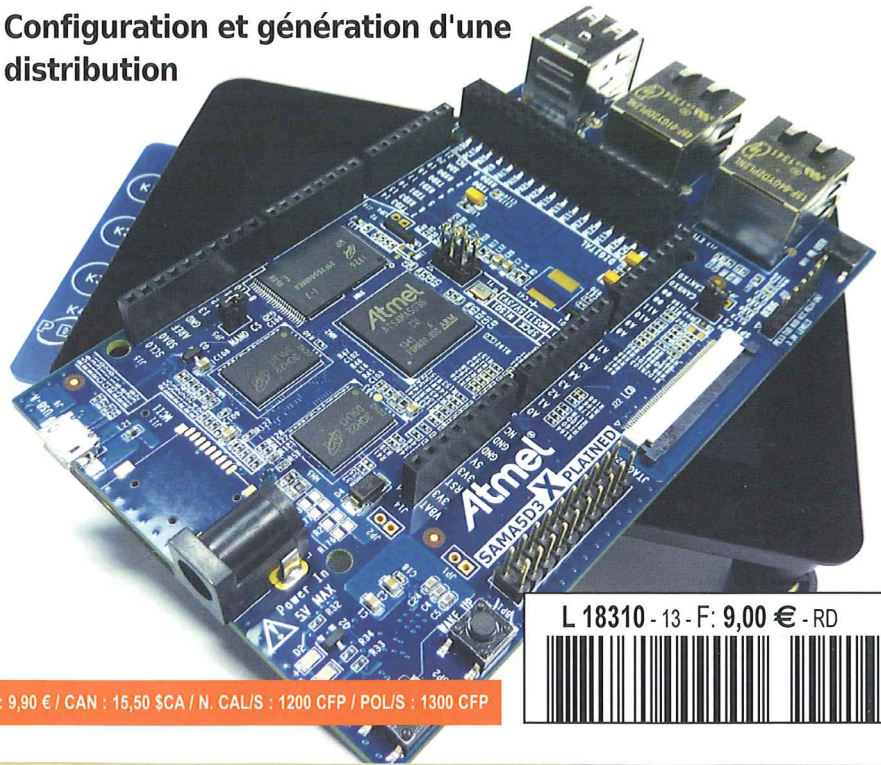
DISTRIBUTION / CORTEX A5

Besoin d'un système de construction modulaire, souple, moderne et globalement adopté ?

DÉMARREZ AVEC YOCTO/POKY p.22

...sur Atmel SAMA5D3 Xplained

- **Découverte et exploration de la plateforme SAMA5D3**
- **S'y retrouver dans Yocto, Poky, OpenEmbedded, OE-Core...**
- **Installation du système de construction**
- **Configuration et génération d'une distribution**



L 18310 - 13 - F: 9,00 € - RD

France Métro : 9 € / Belgique - Luxembourg : 9,50 € / Suisse : 14 CHF / DOM : 9,90 € / CAN : 15,50 $CA / N. CAL/S : 1200 CFP / POL/S : 1300 CFP

SOMMAIRE N°13

SUIVEZ LES DERNIÈRES ACTUALITÉS DE VOTRE MAGAZINE SUR :

FACEBOOK : https://www.facebook.com/editionsdiamond

TWITTER : https://twitter.com/open_silicium

Open Silicium Magazine
est édité par Les Éditions Diamond

LES ÉDITIONS DIAMOND

B.P. 20142 - 67603 Sélestat Cedex
Tél. : 03 67 10 00 20 – **Fax** : 03 67 10 00 21
E-mail : lecteurs@opensilicium.com
Service commercial : abo@opensilicium.com
Sites : www.opensilicium.com – www.ed-diamond.com

Directeur de publication : Arnaud Metzler
Rédacteur en chef : Denis Bodor
Réalisation graphique : Kathrin Scali

Responsable publicité : Black Mouse Communication
Tél. : 03 67 10 00 27
Service abonnement : Tél. : 03 67 10 00 20
Impression : pva, Druck und Medien-Dienstleistungen GmbH, Landau, Allemagne
Distribution France : (uniquement pour les dépositaires de presse)
MLP Réassort :
Plate-forme de Saint-Barthélemy-d'Anjou. Tél. : 02 41 27 53 12
Plate-forme de Saint-Quentin-Fallavier. Tél. : 04 74 82 63 04
Service des ventes : Distri-médias : Tél. : 05 34 52 34 01
IMPRIMÉ en Allemagne - PRINTED in Germany
Dépôt légal : À parution, **N° ISSN** : 2116-3324
Commission paritaire : K90 839
Périodicité : Trimestriel
Prix de vente : 9 €

FSC www.fsc.org — MIXTE Papier Issu de sources responsables FSC® C015136

ÉDITO

Toutes les initiatives ou projets sont-ils destinés à devenir des monstres ?

Vous souvenez-vous de « *The Art of Unix Programming* » de Eric S. Raymond (2003) ? En particulier sa description de la philosophie UNIX qu'il résume par l'acronyme KISS pour « *Keep it Simple, Stupid* » et surtout les 17 règles/idées que les anciens (*elders*) ont utilisés pour créer UNIX : modularité, clarté, composition, séparation, simplicité, parcimonie, transparence, robustesse, représentation, surprise minimum, silence, réparation, économie, génération, optimisation, diversité, extensibilité.

La question que je me pose ces derniers jours (ou nuits) est la suivante : ces lois, règles ou idées sont-elles encore effectivement des guides dans les développements actuels, du moins dans le monde de l'open source ? Si oui, pourquoi cette philosophie ne transparait-elle pas ou ne nous ébloui-t-elle plus avec autant d'ardeur que jusqu'alors ? Si non, est-ce à dire que ces concepts sont maintenant obsolètes et communément considérés comme inutiles ou inutilisables (en d'autres termes, et passez-moi l'expression, ceux qui comme moi se sentent étrangement à l'aise avec ces principes sont-ils devenus des « vieux cons » d'un autre âge ? *boudidiou* ?!).

De manière générale pourtant, certains de ces points me semblent importants sinon critiques. La transparence, « *Design for visibility to make inspection and debugging easier* » implique par exemple, à l'échelle de n'importe quel système, qu'on puisse retrouver rapidement comment un programme ou un service est exécuté et non en arriver, tard dans la nuit, à envisager la magie ou la génération spontanée comme une option. Le silence, « *When a program has nothing surprising to say, it should say nothing* », semble également un peu moins de mise, au bénéfice d'un bavardage masqué par un *boot screen* (se boucher les oreilles ce n'est pas le silence, c'est une dissimulation du bruit, nuance). Et que dire du minimum de surprise, « *In interface design, always do the least surprising thing* » (ESR parle ici d'interface au sens large, non d'IHM ou de GUI) ? Quand vous êtes-vous dit pour la dernière fois en découvrant un environnement, un système ou une solution, « *ah oui, c'est logique que ce soit là* » et non « *@&#$%?!, mais ça sort d'où ça* » ? L'obfuscation dans l'open source, en principe, est une activité quasi-ludique...

Et enfin, et non des moindres, l'idée qui me semble être celle disparaissant dans les ténèbres et l'oubli le plus rapidement, la simplicité : « *Design for simplicity; add complexity only where you must* ». Cette complexité que nous devons en principe ajouter que lorsque nous n'avons plus le choix est presque omniprésente, des systèmes de démarrage, aux outils de construction en passant par les systèmes graphiques, les interfaces utilisateurs et même les aspects serveurs.

Il est intéressant de relever la similitude du phénomène, l'augmentation de complexité doublée consécutivement du manque d'information, avec la notion d'entropie. Si nous envisageons les systèmes informatiques dans leur ensemble comme un tout divisé en deux systèmes thermodynamiques, l'aspect développement (code, architecture, structure) et l'aspect utilisateur (GUI, IHM, bref la « *user experience* »), que dites-vous de ceci : « *l'entropie d'un système isolé ne peut qu'augmenter ou rester constante, et l'entropie d'un système peut diminuer mais cela signifie que l'entropie du milieu extérieur augmente de façon plus importante* » (merci Wikipédia). C'est un peu tiré par les cheveux mais avec un peu de café et pas mal de manque de sommeil, ça se tient.

Tablettes, IoT, domotique, PC/Mac, smartphone, réseaux sociaux, services en ligne... tout ceci semble de plus en plus simplifié, contrasté et intuitif pour l'utilisateur. Ne serait-ce pas là la fameuse baisse d'entropie du système qui, par définition, ne peut qu'augmenter celle du système « extérieur » ?

Est-ce alors quelque chose d'inéluctable ? Nos systèmes sont-ils destinés à devenir de plus en plus complexes et à s'éloigner petit à petit d'une philosophie qui les a fait naître et les a portés jusqu'au présent ? Je n'ai bien entendu pas la réponse, mais uniquement l'espoir que ce ne soit pas le cas car si tel devait se dérouler l'avenir, le prix du ticket d'entrée de la prochaine génération de développeur risque d'être salé... Ou alors tout le monde développera des modules noyaux en Lava (mix de Java et Logo) avec *Eclipse 12.6 Blackhole* en faisant des *drag'n'drop* dans un IDE 3D, et le C, le shell ou Sed seront des langues mortes uniquement connues des trois développeurs GNU/Hurd et Minix encore vivants, qui sait...

Ceci dit, certains, et j'en suis, resteront fermement, intimement et résolument attachés à ce qu'on pourrait presque qualifier de ligne de conduite ou de système de valeurs, quitte à passer un jour ou l'autre pour des hérétiques...

Denis Bodor

CLUSTER DE COMPILATION SUR RPI AVEC DISTCC

par Christophe BLAESS

Le faible coût du Raspberry Pi en fait une plate-forme de choix pour essayer de construire un petit cluster de compilation. La mise en œuvre est très simple, mais quelles en seront finalement les performances ?

1 Compilation distribuée

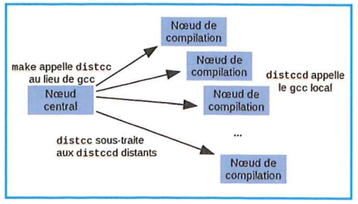
Il y a quelques temps, j'ai posté sur mon blog un article **[1]** expliquant comment compiler des modules pour le noyau Linux directement sur le Raspberry Pi. Ceci nécessitait de recompiler au préalable le noyau complet nativement sur cette carte, ce qui prend facilement six à huit heures. Un commentaire sur le blog suggérait d'utiliser l'outil **distcc** pour lancer des compilations sur le Raspberry Pi, tout en faisant tourner le compilateur proprement dit (**gcc**) sur un PC plus puissant (en employant bien entendu un *cross-compiler*). C'est une approche très intéressante, mais j'ai eu envie d'essayer une méthode différente, en utilisant un réseau de Raspberry Pi pour réaliser une compilation parallélisée.

Restons bien conscients que cela relève plus de l'expérience amusante que d'une mise en œuvre réellement utilisable ! Nous allons voir comment réaliser un petit cluster de compilation distribuée, et surtout vérifier si les résultats obtenus sont intéressants.

2 Distcc

L'outil **distcc** **[2]** permet de répartir sur un ou plusieurs nœuds (serveurs) des tâches de compilation sous-traitées par un client central. Il est essentiellement prévu pour fonctionner avec le compilateur **gcc**. Il y a deux utilitaires distincts :

- **distcc** : le client que l'on appelle en remplacement de **gcc** sur l'hôte central où tourne la compilation,
- **distccd** : le serveur qui fonctionne en arrière-plan sous forme de démon sur chaque nœud de compilation et qui invoque son **gcc** local à la demande d'un client.



Le nœud central peut également participer à la compilation en faisant tourner **distccd** localement pour réaliser une partie du travail. J'ai toutefois choisi de ne pas utiliser cette possibilité, préférant réserver un Raspberry Pi dédié pour distribuer les tâches de compilation sur les autres nœuds du cluster.

3 Préparation du cluster de compilation

La première étape consiste à préparer les serveurs de compilation qui constitueront les nœuds de notre réseau. Pour cela il faut installer sur une première cible une distribution neuve (j'ai choisi une *Raspbian*, mais cela devrait fonctionner avec la plupart des autres distributions), la configurer, puis dupliquer la configuration sur les autres postes.

Je prépare donc sur un PC une carte SD avec la dernière *Raspbian* **[3]** au moment de la rédaction de ces lignes (mais il ne devrait pas y avoir beaucoup de changements pour les versions ultérieures).

```
[~]$ sudo dd if=2014-06-20-wheezy-raspbian.
img of=/dev/sdc bs=1M
2825+0 enregistrements lus
2825+0 enregistrements écrits
2962227200 octets (3,0 GB) copiés, 633,46 s,
4,7 MB/s
```

L'image fait exactement 2825 Mo. Notons cette valeur, nous la réutiliserons plus loin.

3.1 Configuration initiale

Je démarre le premier Raspberry Pi sur l'image ainsi préparée, puis je me connecte en utilisant une console texte sur le port série (on peut bien entendu procéder avec un clavier et un écran).

```
Raspbian GNU/Linux 7 raspberrypi ttyAMA0

raspberrypi login: pi
Password: (raspberry)
Linux raspberrypi 3.12.22+ #691 PREEMPT
Wed Jun 18 18:29:58 BST 2014 armv6l
[...]
NOTICE: the software on this Raspberry Pi
has not been fully configured.
Please run 'sudo raspi-config'
pi@raspberrypi:~$
```

Contrairement à ce que le système nous demande (cf dernière ligne ci-dessus) nous n'allons pas lancer immédiatement **raspi-config** pour lui demander d'utiliser toute la place disponible sur la carte SD, car je souhaite dupliquer ma configuration sur des cartes hétérogènes, ayant des tailles variables suivant les fabricants. Je vais donc me contenter dans un premier temps de la place occupée par l'image initiale (2825 Mo).

Sur les nœuds de compilation, je n'ai pas besoin d'environnement graphique, je peux donc libérer un peu de place dans le système de fichiers. Ceci sera surtout utile sur l'hôte central, car il devra contenir toutes les sources ainsi que les fichiers objets intermédiaires.

```
$ sudo apt-get update
[...]
$ sudo apt-get purge x11-common xserver-xorg
libx11-6
[...]
$ sudo apt-get autoremove
```

Je vais également supprimer un autre package : il s'agit de **dphys-swapfile** qui configure automatiquement un fichier d'échange. Ce mécanisme, activé par défaut dans la distribution Rapsbian, agit lorsque la mémoire RAM commence à être presque pleine (ce qui arrive rapidement en faisant de grosses compilations). Dans cette situation le système va *swapper* des pages de mémoire – c'est à dire les sauvegarder dans ce fichier situé sur la carte SD – pour les retrouver plus tard. Ceci provoque de nombreuses lectures / écritures sur la mémoire flash de la carte SD et la vieillit prématurément. Pour l'anecdote, lors de mes premiers essais j'avais oublié de supprimer ce package, et après quelques heures de compilation, **cinq cartes SD sur huit sont devenues inutilisables** à quelques dizaines de minutes d'intervalle !

```
$ sudo apt-get remove dphys-swapfile
```

Le fait de supprimer le mécanisme de swap n'est pas gênant en soi, cela limitera seulement le nombre de jobs de compilation susceptibles de s'exécuter en parallèle.

Ces premières étapes ont libéré de la place en supprimant les packages inutiles. Il nous faut maintenant installer les utilitaires pour la compilation distribuée :

```
$ sudo apt-get install distcc
```

La configuration du démon de compilation **distccd** doit être modifiée manuellement. J'utilise l'éditeur **vi**, mais on peut également employer **nano**, **joe**, etc.

```
$ sudo vi /etc/defaults/distcc
```

Dans ce fichier on modifie les lignes suivantes :

- **STARTDISTCC="false"** devient **STARTDISTCC="true"** pour valider le lancement automatique du démon **distccd** au démarrage du système.
- **ALLOWEDNETS="127.0.0.1"** devient **ALLOWEDNETS="192.168.3.0/24"** les machines autorisées à demander une compilation distribuée se trouveront toutes dans le sous-réseau privé **192.168.3.x** (à adapter évidemment en fonction de votre environnement de travail).
- **LISTENER="127.0.0.1"** est mis en commentaire car le serveur de compilation ne se limitera pas à une interface réseau particulière (on pourrait imaginer l'utilisation avec outre le contrôleur Ethernet standard, une interface par *dongle* USB/Wifi).

Je vais alors arrêter cette première cible, et sauvegarder cette image dans un fichier sur PC, afin de pouvoir la déployer ultérieurement sur les autres nœuds.

J'insère la carte SD sur mon PC et récupère les 2825 premiers méga-octets (valeur obtenue lors du premier **dd**).

```
[~]$ sudo dd if=/dev/sdc of=distcc-node.img
bs=1M count=2825
```

Je peux alors recopier cette image sur toutes les cartes SD que je souhaite en les insérant l'une après l'autre et en saisissant la ligne suivante (attention à bien démonter auparavant les partitions de la carte SD qui peuvent être montées automatiquement à l'insertion) :

```
[~]$ sudo dd if=distcc-node.img of=/dev/
sdc bs=1M
```

3.2 Finalisation d'un nœud de compilation

Après avoir démarré un poste serveur de compilation, je termine sa configuration en lui affectant une adresse IP de manière statique. On peut très bien laisser un serveur DHCP affecter les adresses IP des nœuds de compilation, il suffira dans ce cas de bien noter les adresses obtenues pour les indiquer à **distcc** (voir plus loin). Cette étape est un peu fastidieuse car il faut la répéter sur chaque Raspberry Pi de compilation.

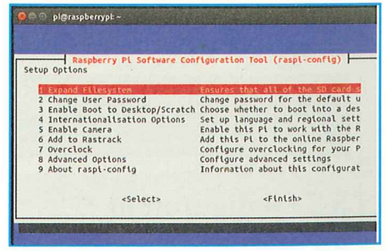
```
raspberrypi login: pi
Password: (raspberry)
pi@raspberrypi:~$ sudo vi /etc/network/interfaces
```

Je modifie le fichier de manière à attribuer l'adresse de manière statique (les lignes concernées sont les quatre à partir de **iface eth0**) :

```
auto lo

iface lo inet loopback

iface eth0 inet static
        address 192.168.3.203
        netmask 255.255.255.0
        gateway 192.168.3.254

allow-hotplug wlan0
iface wlan0 inet manual
wpa-roam /etc/wpa_supplicant/wpa_supplicant.conf
iface default inet dhcp
```

Puis j'appelle l'utilitaire **raspi-config** pour utiliser toute la place disponible sur la carte SD.

```
$ sudo raspi-config
```



Je choisis la première option « *Expand Filesystem* », puis je redémarre à nouveau le serveur de compilation.

3.3 Installation du nœud central

Pour le nœud central du cluster, celui qui sous-traitera les compilations aux nœuds serveurs, j'utilise exactement la même configuration que pour les autres.

Pour obtenir des résultats significatifs, il nous faut compiler un programme d'envergure assez importante afin que sa durée de compilation soit suffisamment longue pour voir les variations en fonction du nombre de nœuds, du nombre de jobs, etc. J'ai choisi de compiler le noyau Linux comme si nous voulions l'installer sur le Raspberry Pi.

Cela réclame l'ajout (sur le Raspberry Pi central uniquement) de l'utilitaire **bc**, utilisé pendant la compilation :

```
$ sudo apt-get install bc
```

3.4 Préparation des sources de Linux

Nous devons disposer des sources de Linux sur le nœud central. Inutile toutefois de les copier sur les serveurs de compilation. Le téléchargement dure 30 minutes environ suivant votre débit réseau.

```
$ git clone http://github.com/raspberrypi/
linux rpi-kernel
$ cd rpi-kernel/
```

Pour pouvoir lancer plusieurs expériences successives, je dois effacer toutes les traces des compilations précédentes. Bien entendu ceci ne sert à rien la première fois, mais sera répété à chaque test.

```
~/rpi-kernel$ make mrproper
~/rpi-kernel$ ccache --clear
~/rpi-kernel$ sync
```

3.5 Configuration du noyau

Le noyau obtenu n'étant pas destiné à être démarré, sa configuration importe peu, pourvu qu'elle soit identique à chaque tests – et qu'elle soit assez cohérente pour qu'il n'y ait pas d'erreur de compilation. Je prends donc la configuration du noyau courant (on pourrait également utiliser **bcmrpi_defconfig**).

```
$ zcat /proc/config.gz > .config
$ make oldconfig
```

4 Compilation distribuée

Sur le Raspberry Pi central, la variable d'environnement **DISTCC_HOSTS** doit contenir la liste des serveurs à joindre. Il serait possible d'y ajouter **127.0.0.1** pour que ce nœud central participe aussi à la compilation.

Le principe consiste à se placer, sur l'hôte central, dans le répertoire des sources de Linux et à lancer la compilation normalement avec **make** en ayant simplement rempli la variable d'environnement **CC** avec la chaîne de caractères **"distcc"**. Ceci remplacera automatiquement les appels au compilateur **gcc** local par des invocations de la version cliente de **distcc** répartissant ainsi la compilation vers les nœuds indiqués dans la variable **DISTCC_HOSTS**.

Afin d'avoir un temps de compilation qui ne soit pas trop prohibitif, j'ai choisi de ne compiler que le noyau proprement dit, pas ses modules (ce qui aurait approximativement doublé les temps). Voici un script permettant d'automatiser la compilation :

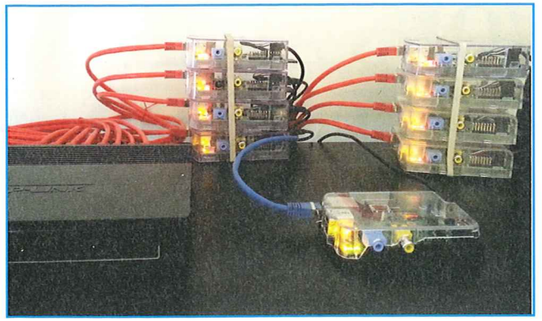
```
#! /bin/bash

if [ $# -lt 1 ]
then
   echo "usage: $0 <nb_nodes>" >&2
   exit 1
fi

NB_NODES=$(($1))
if [ $NB_NODES -le 0 ]
then
   echo "$0: wrong node number" >&2
   exit 1
fi

HOST[1]=192.168.3.201
HOST[2]=192.168.3.202
HOST[3]=192.168.3.203
HOST[4]=192.168.3.204
HOST[5]=192.168.3.205
HOST[6]=192.168.3.206
HOST[7]=192.168.3.207
HOST[8]=192.168.3.208

export DISTCC_HOSTS=""
for node in $(seq 1 ${NB_NODES} )
do
   DISTCC_HOSTS="${DISTCC_HOSTS}${HOST[$node]} "
done

for NB_JOBS in 1 2 4 8 16 32 64 128
do
   make mrproper
   zcat /proc/config.gz > .config
   make oldconfig
   sync
   echo "${NB_NODES} nodes" >> ../compile-time.txt
   echo "${NB_JOBS} jobs" >> ../compile-time.txt
   date >> ../compile-time.txt
   make CC="distcc" -j ${NB_JOBS} zImage
   date >> ../compile-time.txt
   echo >> ../compile-time.txt
done
```

On renseigne dans les variables **HOST[]** les adresses IP ou les noms d'hôtes des nœuds de calcul, et on lance le script en lui précisant en argument le nombre de nœuds à impliquer. Le script va alors réaliser une série de compilations successives avec 1, 2, 4, 8... 128 jobs en parallèle.

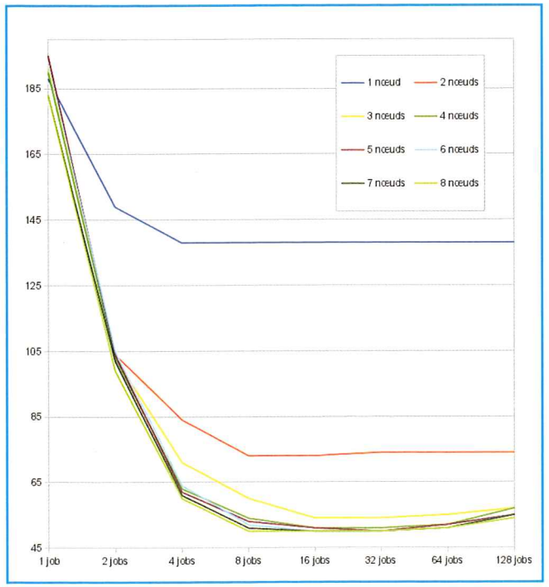
5 Résultats

J'ai mené cette série d'expériences sur une batterie de huit Raspberry Pi pour les compilations et un Raspberry Pi pour la répartition des tâches. Les compilations successives ont duré un peu plus d'une semaine car je souhaitais reprendre la main de temps à autres pour voir les résultats partiels. Le temps de calcul global est d'environ quatre jours.

Dans le tableau ci-contre figurent les durées (en minutes) des compilations du noyau Linux sans ses modules. Chaque ligne représente une série de compilations pour un nombre donné de nœuds physiques. Les colonnes indiquent le nombre de jobs de compilation lancés en parallèle (argument **-j** de la commande **make**).



Tout à fait logiquement, plus on utilise de nœuds de calcul, meilleurs sont les temps. Pour cela il convient évidemment de lancer au moins autant de jobs de compilation qu'il y a de nœuds disponibles.



Les résultats obtenus exprimés en minutes de compilation du noyau Linux en fonction du nombre de jobs et de nœuds.

Il ne faut pas non plus lancer trop de jobs en parallèle, car le Raspberry Pi central est vite saturé à essayer de répartir ses tâches vers ses nœuds de calcul qui tournent déjà à 100 %. Augmenter le nombre de jobs au-delà de 16 n'améliore pas les performances, quelque soit le nombre de nœuds de calcul. Il faut être bien conscients que ceci est surtout une limitation du Raspberry Pi dont la taille mémoire, la vitesse processeur, et l'interface réseau sont très limités.

On voit très bien d'ailleurs que dans notre cas, augmenter le nombre de Raspberry Pi de calcul au-delà de 6 nœuds n'améliore pas vraiment les temps d'exécution. La compilation distribuée nécessite de fréquents échanges de données, et le goulet d'étranglement que nous rencontrons n'est plus la puissance CPU (au delà de cinq à six nœuds de calcul) mais la performance réseau du contrôleur Ethernet du nœud central. Ce contrôleur est de qualité médiocre sur les Raspberry Pi et ses drivers ne sont pas optimisés (et sont de surcroît très mal écrits).

En conclusion, on peut dire que l'utilisation d'un réseau de Raspberry Pi pour réaliser des compilations n'est pas vraiment une solution avantageuse. Il ne faut pas oublier qu'un PC d'entrée de gamme compile un noyau ajusté pour une cible embarqué en quelques minutes.

Cette expérience présente néanmoins un intérêt pédagogique certain : on peut ainsi facilement se former à la mise en œuvre de serveurs de compilation distribuée en employant des plates-formes bon marché avec une distribution Linux standard, et maquetter ainsi l'architecture que l'on emploiera ensuite en production pour des serveurs d'intégration continue plus puissants. ■

Liens

[1] http://www.blaess.fr/christophe/2014/03/06/ : compilation native de modules kernel sur Raspberry Pi ;

[2] https://code.google.com/p/distcc/ : site de référence de distcc ;

[3] http://www.raspbian.org/ : site principal de la distribution Raspbian

	1 job	2 jobs	4 jobs	8 jobs	16 jobs	32 jobs	64 jobs	128 jobs
1 nœud	188	149	138	138	138	138	138	138
2 nœuds	195	104	84	73	73	74	74	74
3 nœuds	192	103	71	60	54	54	55	57
4 nœuds	190	103	63	54	51	51	52	57
5 nœuds	195	104	62	53	51	50	52	55
6 nœuds	193	105	64	52	50	50	51	55
7 nœuds	183	102	61	51	50	50	51	55
8 nœuds	183	99	60	50	50	50	51	54

TOUT CE QUE VOUS AVEZ TOUJOURS VOULU SAVOIR SUR TCP/IP... OU PRESQUE

par Cédric PELLERIN - Utilisateur de GNU/Linux depuis 1993

Rien n'est plus naturel depuis une ou deux décennies que de brancher son ordinateur sur une prise réseau, de le laisser récupérer une adresse IP puis de naviguer sur le net ou administrer des serveurs à grands coups de ssh rageurs. Mais en fait que se passe-t-il réellement ? Quels sont les mécanismes mis en jeu ? Comment mon ftp est-il transformé en signaux électriques dans les paires de cuivre ?

Ce que nous utilisons presque sans y penser à l'heure actuelle est le fruit d'une longue évolution et d'une succession de changements plus ou moins radicaux dans les protocoles utilisés pour que notre ordinateur puisse interagir avec un autre via le réseau.

1 TCP/IP mais au fond, qu'est-ce que c'est ?

Toute personne ou presque qui utilise un ordinateur de nos jours a entendu parler de l' « adresse IP ». Parmi ces personnes, certaines sont au courant qu'il s'agit d'une série de quatre octets séparés par un point. Mais au-delà, c'est le grand trou noir. Mettons donc un peu de lumière là-dedans et examinons tout ça de près.

1.1 Un peu d'histoire

Elaboré depuis 1973 à l'université de Stanford, TCP/IP subit ses premiers tests grandeurs réelles en 1975 en transportant des données entre Stanford et l'University College London (UCL). La première interconnexion de réseaux a lieu en 1977 entre les USA, l'Angleterre et la Norvège. De nombreuses retouches furent apportées au protocole jusqu'au 1er Janvier 1983, date officielle de sortie de la version définitive. En 1982 il fut adopté par les militaires américains avant de devenir en 1985 le protocole officiel de ce qui va devenir internet.

Fig. 1 : Flux TCP/IP et découpage en couches

1.2 Un protocole en « couches »

L'idée à la base de ce découpage est que chaque couche fasse uniquement ce pour quoi elle a été conçue et ensuite passe la main à la couche immédiatement supérieure (en réception) ou inférieure (à l'émission). Cela se passe comme montré Figure 1.

Dans le cas de TCP/IP, nous pouvons énumérer les cinq couches suivantes :

Niveau	Nom	Exemples	Interconnexion via
5	Application	HTTP, FTP, Telnet...	
4	Transport	TCP, UDP...	N° de port réseau
3	Réseau	IP	Adresse IP
2	Liaison	Ethernet, Token Ring...	Adresse MAC
1	Physique	Ligne téléphonique, ADSL...	

De la couche *Application* proviennent les données à transférer et ensuite chaque couche rajoute son en-tête avec ses informations propres (par exemple l'adresse IP de destination est l'une de ces informations rajoutées par la couche Réseau).

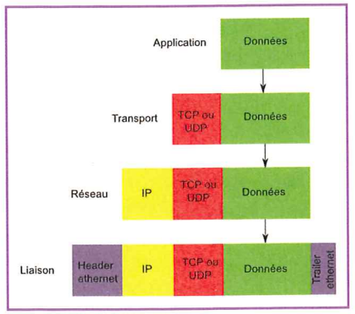


Fig. 2 : Encapsulation des données

Une fois toutes les couches descendues, la carte réseau transforme tout ce petit monde en impulsions électriques sur le câble réseau jusqu'à la carte réseau de destination. Alors les couches sont remontées les unes après les autres, les entêtes enlevées après vérification et les données parviennent à la couche Application où elles sont traitées.

Le protocole connu sous le nom de TCP/IP est en fait une suite de protocoles qui ont chacun leurs utilités et qui interviennent en général au niveau 3 ou 4. Parmi ceux-ci on trouve ICMP (pour les ping entre autres), les protocoles de routage comme BGP ou OSPF, ARP pour Address Resolution Protocol dont nous allons bientôt parler plus en détail et de nombreux autres.

Une autre chose importante à connaître concernant TCP/IP est qu'il travaille par segments, c'est à dire que les informations à envoyer sont découpées en « paquets » de longueurs fixes qui sont réassemblés à l'arrivée. Cette méthode permet de n'avoir qu'un paquet à renvoyer s'il est perdu au lieu de toutes les données. Elle permet aussi de mieux contrôler le trafic et de vérifier plus facilement l'intégrité des données. Ces contrôles se font en TCP mais pas en UDP, protocole plus léger, plus rapide mais sans aucune garantie. Nous en reparlerons plus tard.

2 Un exemple vaut mieux qu'un long discours

Afin de bien comprendre comment tout cela fonctionne et pouvoir ensuite généraliser sans trop de dommages, nous allons prendre l'exemple d'une requête HTTP ayant pour but d'afficher une page HTML faisant partie de l'interface d'administration d'une imprimante réseau. La station de travail aura pour adresse 172.20.1.10 et l'imprimante 172.20.0.200. Le réseau est 172.20.0.0/22 ce qui fait que les deux appareils sont dans le même sous-réseau, nous éviterons ainsi tous les problèmes de routage. Dans ce qui suit nous allons essayer d'expliquer le plus clairement possible les mécanismes mis en jeu mais nous nous en tiendrons au cadre de notre exemple. Une explication de tous les mécanismes TCP/IP possibles prendrait un livre entier et serait assez indigeste.

2.1 « Vivisectionnons » avec ordre et méthode

Le scalpel utilisé pour cette vivisection s'appelle *Wireshark* – anciennement *Ethereal* – et les traces sont directement exportées. Pour ceux qui ne connaissent pas encore vous en avez un aperçu en Figure 3.

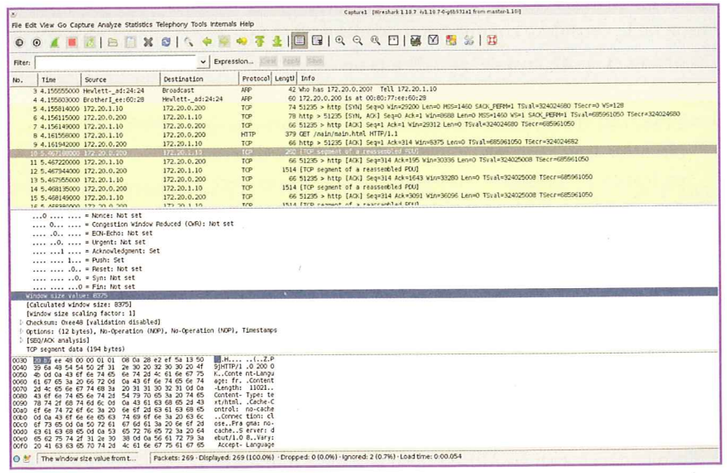


Fig. 3 : Ecran de Wireshark

La requête tapée dans la barre d'url de notre station est : http://172.20.0.200/main/main.html.

Connaissant son adresse IP propre, l'adresse de son réseau et le masque de sous-réseau, la station sait qu'elle est sur le même sous-réseau que le serveur et donc que c'est à elle de tout faire. Afin de pouvoir générer des trames ethernet complètes, elle a besoin de connaître l'adresse MAC de la carte réseau distante. Pour ce faire elle utilise un protocole spécial nommé ARP qui va lui renvoyer l'adresse MAC de la station possédant l'adresse IP de destination. Cela donne un échange comme celui-là :

```
No.     Time        Source                Destination           Protocol  Length  Info
3   4.155555000     Hewlett-_ad:24:24     Broadcast             ARP        42     Who has
172.20.0.200?  Tell 172.20.1.10

Frame 3: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface 0
Ethernet II, Src: Hewlett-_ad:24:24 (00:1c:c4:ad:24:24), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
    Destination: Broadcast (ff:ff:ff:ff:ff:ff)
```

```
        Address: Broadcast (ff:ff:ff:ff:ff:ff)
        .... ..1. .... .... .... .... = LG bit: Locally administered address
(this is NOT the factory default)
        .... ...1 .... .... .... .... = IG bit: Group address (multicast/
broadcast)
    Source: Hewlett-_ad:24:24 (00:1c:c4:ad:24:24)
        Address: Hewlett-_ad:24:24 (00:1c:c4:ad:24:24)
        .... ..0. .... .... .... .... = LG bit: Globally unique address (factory
default)
        .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
    Type: ARP (0x0806)
Address Resolution Protocol (request)
    Hardware type: Ethernet (1)
    Protocol type: IP (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: request (1)
    Sender MAC address: Hewlett-_ad:24:24 (00:1c:c4:ad:24:24)
    Sender IP address: 172.20.1.10 (172.20.1.10)
    Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
    Target IP address: 172.20.0.200 (172.20.0.200)
```

La question est claire : « Qui a [l'adresse IP] 172.20.0.200 ? ». On peut vérifier que le protocole ARP est de très bas niveau car il agit directement encapsulé dans ethernet, pour cette raison on le considère comme étant de niveau 2 :

- **Frame 3 : 42 bytes on wire** là on est en couche 1, il s'agit des bits remontés depuis le câble
- **Ethernet II** on passe en couche 2, protocole ethernet où l'on peut voir comment fonctionne ARP. Une traduction en français – presque – courant pourrait donner ceci :

  Src: Hewlett-_ad:24:24 (00:1c:c4:ad:24:24), Dst: Broadcast (ff:ff:ff:ff:ff:ff) signifie : je suis une station ayant l'adresse MAC **00:1C:C4:AD:24:24**[1] et je m'adresse à la cantonade (broadcast)

  Type: ARP (0x0806) signifie : ceci est une requête de type ARP.

Pour être encore plus explicite : « Oh les mecs, je suis **00:1C:C4:AD:24:24** et j'envoie une requête ARP, répondez SVP »

Ensuite, dans la requête ARP encapsulée dans la trame ethernet que nous venons de traduire rapidement on trouve ceci :

- **Sender MAC address: Hewlett-_ad:24:24 (00:1c:c4:ad:24:24)**
- **Sender IP address: 172.20.1.10 (172.20.1.10)**

Là on a répétition de l'adresse MAC de l'expéditeur mais on trouve aussi son adresse IP

- **Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)**
- **Target IP address: 172.20.0.200 (172.20.0.200)**

On a ici une adresse MAC à zéro (c'est ce qu'on cherche à connaître) et l'adresse IP de l'ordinateur dont on cherche justement à obtenir l'adresse MAC.

Ensuite vient la réponse :

```
No.     Time       Source                Destination           Protocol
Length     Info
4    4.155803000   BrotherI_ee:60:28     Hewlett-_ad:24:24     ARP          60
172.20.0.200 is at 00:80:77:ee:60:28

Frame 4: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface
0
Ethernet II, Src: BrotherI_ee:60:28 (00:80:77:ee:60:28), Dst: Hewlett-_ad:24:24
(00:1c:c4:ad:24:24)
    Destination: Hewlett-_ad:24:24 (00:1c:c4:ad:24:24)
        Address: Hewlett-_ad:24:24 (00:1c:c4:ad:24:24)
        .... ..0. .... .... .... .... = LG bit: Globally unique address (factory
default)
        .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
    Source: BrotherI_ee:60:28 (00:80:77:ee:60:28)
        Address: BrotherI_ee:60:28 (00:80:77:ee:60:28)
        .... ..0. .... .... .... .... = LG bit: Globally unique address (factory
default)
        .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
    Type: ARP (0x0806)
    Padding: 000000000000000000000000000000000000000000000000
Address Resolution Protocol (reply)
    Hardware type: Ethernet (1)
    Protocol type: IP (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: reply (2)
    Sender MAC address: BrotherI_ee:60:28 (00:80:77:ee:60:28)
    Sender IP address: 172.20.0.200 (172.20.0.200)
    Target MAC address: Hewlett-_ad:24:24 (00:1c:c4:ad:24:24)
    Target IP address: 172.20.1.10 (172.20.1.10)
```

La machine qui répond est une **Brother : Src: BrotherI_ee:60:28 (00:80:77:ee:60:28), Dst: Hewlett-_ad:24:24 (00:1c:c4:ad:24:24)** qui renvoie simplement le même type de trame que pour la question mais, comme elle est adressée à une seule station et non plus broadcastée, il s'agit d'une réponse et non d'une question. Dans la requête encapsulée nous avons la réponse attendue :

```
Sender MAC address: BrotherI_ee:60:28 (00:80:77:ee:60:28)
Sender IP address: 172.20.0.200 (172.20.0.200)
Target MAC address: Hewlett-_ad:24:24 (00:1c:c4:ad:24:24)
Target IP address: 172.20.1.10 (172.20.1.10)
```

L'adresse MAC de la Brother avec son IP et l'adresse MAC et l'adresse IP de la station demandeuse, la HP.

Voilà donc comment on récupère une adresse MAC à partir d'une adresse IP. Chaque machine possède un cache ARP permettant d'éviter cette requête les fois suivantes.

Une fois l'adresse MAC de destination obtenue, l'émetteur dispose de tout ce dont il a besoin pour initier la communication en TCP/IP. La première chose à faire consiste à choisir les ports source et destination car c'est ainsi que TCP travaille : par ports. Le port de destination est déterminé par le protocole applicatif employé, ici il s'agit de HTTP donc du port 80[2]. Il s'agit là d'une convention qui permet de savoir sur quel(s) port(s) va discuter tel protocole et donc de savoir sur quel(s) port(s) le logiciel serveur va écouter et le logiciel

[1] Plus précisément j'utilise pour parler au réseau ma carte ayant l'adresse MAC 00:1C:C4:AD:24:24. Les 3 premiers octets permettent de savoir qu'il s'agit d'une carte réseau fabriquée par HP. On peut chercher le fabriquant d'une carte ou télécharger le listing complet sur le site de l'IEEE : http://standards.ieee.org/develop/regauth/oui/public.html

[2] On peut bien entendu le changer, dans ce cas c'est la couche applicative qui transmet l'information à TCP.

client va le contacter. Cependant il faut aussi déterminer un port source pour que le serveur puisse envoyer sa réponse. Celui-ci est choisi au hasard, en général parmi les plus hauts numéros et bien entendu on vérifie d'abord qu'il n'est pas en cours d'utilisation. Les ports TCP sont divisés en deux groupes principaux : les ports dits « well-known » entre 1 et 1023 qui sont en général fixés par convention pour un protocole donnée (voir /etc/services pour une liste presque exhaustive) et les ports dits éphémères, compris entre 1024 et 65535. Ce découpage est assez théorique car de nombreux ports supérieurs à 1024 sont pris par des protocoles, par exemple ircd qui écoute sur le 6667 ou git sur le 9418. Comme chacun le sait, sous Unix l'utilisation des ports inférieurs à 1024 est réservée à l'utilisateur root.

Une fois ce choix effectué, il reste à démarrer la numérotation des segments TCP avec un beau zéro (ou à reprendre le dernier numéro utilisé dans une communication identique – cf plus bas la partie théorique) dans le champ « Sequence number » qui est la marque d'un début de communication. De la même façon le champ « SYN flag » sera mis à 1 ce qui indique qu'il s'agit d'une demande de synchronisation. En effet, toute communication TCP commence par un échange de synchronisation/accusé de réception nommé en général SYN/ACK pour SYNchronisation/ACKnowledgement. Cet échange est résumé dans le schéma de la Figure 4 (le temps va du haut vers le bas) :

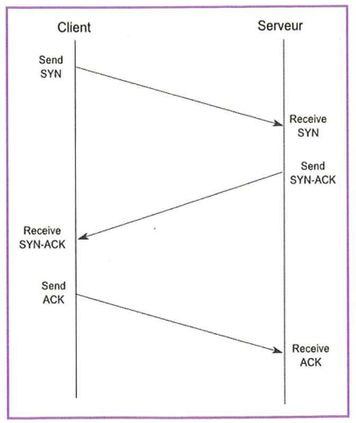


Fig. 4 : Echange SYN/ACK

Nous retrouvons ceci dans les captures effectuées :

1. Send SYN

```
No.     Time       Source        Destination   Protocol Length       Info
 5  4.155814000   172.20.1.10   172.20.0.200  TCP        74        51235 > http [SYN]

Transmission Control Protocol, Src Port: 51235 (51235), Dst Port: http (80), Seq: 0, Len: 0
  Source port: 51235 (51235)
  Destination port: http (80)
  [Stream index: 0]
  Sequence number: 0    (relative sequence number)
  Header length: 40 bytes
  Flags: 0x002 (SYN)
        000. .... .... = Reserved: Not set
        ...0 .... .... = Nonce: Not set
        .... 0... .... = Congestion Window Reduced (CWR): Not set
        .... .0.. .... = ECN-Echo: Not set
        .... ..0. .... = Urgent: Not set
        .... ...0 .... = Acknowledgment: Not set
        .... .... 0... = Push: Not set
        .... .... .0.. = Reset: Not set
        .... .... ..1. = Syn: Set
        .... .... ...0 = Fin: Not set
```

où l'on voit le port source choisi qui est le 51235, le port destination 80 comme prévu, le n° de séquence à 0 (Seq: 0) ainsi que le flag SYN mis seul à 1.

2. Receive SYN qui s'est certainement bien déroulé (je n'ai pas effectué de captures côté serveur) car la suite se déroule normalement.

3. Send SYN-ACK.

```
No.     Time       Source        Destination   Protocol Length Info
 6  4.156115000   172.20.0.200  172.20.1.10   TCP        78   http > 51235 [SYN, ACK]
Transmission Control Protocol, Src Port: http (80), Dst Port: 51235 (51235), Seq: 0, Ack: 1, Len: 0
    Source port: http (80)
    Destination port: 51235 (51235)
    [Stream index: 0]
    Sequence number: 0    (relative sequence number)
    Acknowledgment number: 1    (relative ack number)
    Header length: 44 bytes
    Flags: 0x012 (SYN, ACK)
        000. .... .... = Reserved: Not set
        ...0 .... .... = Nonce: Not set
        .... 0... .... = Congestion Window Reduced (CWR): Not set
        .... .0.. .... = ECN-Echo: Not set
        .... ..0. .... = Urgent: Not set
        .... ...1 .... = Acknowledgment: Set
        .... .... 0... = Push: Not set
        .... .... .0.. = Reset: Not set
        .... .... ..1. = Syn: Set
        .... .... ...0 = Fin: Not set
```

Nous avons donc la réponse du serveur envoyée sur notre port 51235 depuis son port 80 avec le n° de séquence toujours à 0 et nos deux flags ACK et SYN à 1.

4. Receive SYN-ACK.

5. Send ACK.

```
        Address: Broadcast (ff:ff:ff:ff:ff:ff)
        .... ..1. .... .... .... .... = LG bit: Locally administered address
(this is NOT the factory default)
        .... ...1 .... .... .... .... = IG bit: Group address (multicast/
broadcast)
    Source: Hewlett-_ad:24:24 (00:1c:c4:ad:24:24)
        Address: Hewlett-_ad:24:24 (00:1c:c4:ad:24:24)
        .... ..0. .... .... .... .... = LG bit: Globally unique address (factory
default)
        .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
    Type: ARP (0x0806)
Address Resolution Protocol (request)
    Hardware type: Ethernet (1)
    Protocol type: IP (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: request (1)
    Sender MAC address: Hewlett-_ad:24:24 (00:1c:c4:ad:24:24)
    Sender IP address: 172.20.1.10 (172.20.1.10)
    Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
    Target IP address: 172.20.0.200 (172.20.0.200)
```

La question est claire : « Qui a [l'adresse IP] 172.20.0.200 ? ». On peut vérifier que le protocole ARP est de très bas niveau car il agit directement encapsulé dans ethernet, pour cette raison on le considère comme étant de niveau 2 :

- **Frame 3 : 42 bytes on wire** là on est en couche 1, il s'agit des bits remontés depuis le câble
- **Ethernet II** on passe en couche 2, protocole ethernet où l'on peut voir comment fonctionne ARP. Une traduction en français – presque – courant pourrait donner ceci :
  Src: Hewlett-_ad:24:24 (00:1c:c4:ad:24:24), Dst: Broadcast (ff:ff:ff:ff:ff:ff) signifie : je suis une station ayant l'adresse MAC **00:1C:C4:AD:24:24**[1] et je m'adresse à la cantonade (broadcast)

  Type: ARP (0x0806) signifie : ceci est une requête de type ARP.

Pour être encore plus explicite : « Oh les mecs, je suis **00:1C:C4:AD:24:24** et j'envoie une requête ARP, répondez SVP »

Ensuite, dans la requête ARP encapsulée dans la trame ethernet que nous venons de traduire rapidement on trouve ceci :

- **Sender MAC address: Hewlett-_ad:24:24 (00:1c:c4:ad:24:24)**
- **Sender IP address: 172.20.1.10 (172.20.1.10)**

Là on a répétition de l'adresse MAC de l'expéditeur mais on trouve aussi son adresse IP

- **Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)**
- **Target IP address: 172.20.0.200 (172.20.0.200)**

On a ici une adresse MAC à zéro (c'est ce qu'on cherche à connaître) et l'adresse IP de l'ordinateur dont on cherche justement à obtenir l'adresse MAC.

Ensuite vient la réponse :

```
No.     Time       Source                Destination           Protocol
Length     Info
4    4.155803000   BrotherI_ee:60:28     Hewlett-_ad:24:24     ARP          60
172.20.0.200 is at 00:80:77:ee:60:28

Frame 4: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface
0
Ethernet II, Src: BrotherI_ee:60:28 (00:80:77:ee:60:28), Dst: Hewlett-_ad:24:24
(00:1c:c4:ad:24:24)
    Destination: Hewlett-_ad:24:24 (00:1c:c4:ad:24:24)
        Address: Hewlett-_ad:24:24 (00:1c:c4:ad:24:24)
        .... ..0. .... .... .... .... = LG bit: Globally unique address (factory
default)
        .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
    Source: BrotherI_ee:60:28 (00:80:77:ee:60:28)
        Address: BrotherI_ee:60:28 (00:80:77:ee:60:28)
        .... ..0. .... .... .... .... = LG bit: Globally unique address (factory
default)
        .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
    Type: ARP (0x0806)
    Padding: 000000000000000000000000000000000000
Address Resolution Protocol (reply)
    Hardware type: Ethernet (1)
    Protocol type: IP (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: reply (2)
    Sender MAC address: BrotherI_ee:60:28 (00:80:77:ee:60:28)
    Sender IP address: 172.20.0.200 (172.20.0.200)
    Target MAC address: Hewlett-_ad:24:24 (00:1c:c4:ad:24:24)
    Target IP address: 172.20.1.10 (172.20.1.10)
```

La machine qui répond est une **Brother : Src: BrotherI_ee:60:28 (00:80:77:ee:60:28), Dst: Hewlett-_ad:24:24 (00:1c:c4:ad:24:24)** qui renvoie simplement le même type de trame que pour la question mais, comme elle est adressée à une seule station et non plus broadcastée, il s'agit d'une réponse et non d'une question. Dans la requête encapsulée nous avons la réponse attendue :

```
Sender MAC address: BrotherI_ee:60:28 (00:80:77:ee:60:28)
Sender IP address: 172.20.0.200 (172.20.0.200)
Target MAC address: Hewlett-_ad:24:24 (00:1c:c4:ad:24:24)
Target IP address: 172.20.1.10 (172.20.1.10)
```

L'adresse MAC de la Brother avec son IP et l'adresse MAC et l'adresse IP de la station demandeuse, la HP.

Voilà donc comment on récupère une adresse MAC à partir d'une adresse IP. Chaque machine possède un cache ARP permettant d'éviter cette requête les fois suivantes.

Une fois l'adresse MAC de destination obtenue, l'émetteur dispose de tout ce dont il a besoin pour initier la communication en TCP/IP. La première chose à faire consiste à choisir les ports source et destination car c'est ainsi que TCP travaille : par ports. Le port de destination est déterminé par le protocole applicatif employé, ici il s'agit de HTTP donc du port 80[2]. Il s'agit là d'une convention qui permet de savoir sur quel(s) port(s) va discuter tel protocole et donc de savoir sur quel(s) port(s) le logiciel serveur va écouter et le logiciel

[1] Plus précisément j'utilise pour parler au réseau ma carte ayant l'adresse MAC 00:1C:C4:AD:24:24. Les 3 premiers octets permettent de savoir qu'il s'agit d'une carte réseau fabriquée par HP. On peut chercher le fabriquant d'une carte ou télécharger le listing complet sur le site de l'IEEE : http://standards.ieee.org/develop/regauth/oui/public.html

[2] On peut bien entendu le changer, dans ce cas c'est la couche applicative qui transmet l'information à TCP.

client va le contacter. Cependant il faut aussi déterminer un port source pour que le serveur puisse envoyer sa réponse. Celui-ci est choisi au hasard, en général parmi les plus hauts numéros et bien entendu on vérifie d'abord qu'il n'est pas en cours d'utilisation. Les ports TCP sont divisés en deux groupes principaux : les ports dits « well-known » entre 1 et 1023 qui sont en général fixés par convention pour un protocole donnée (voir /etc/services pour une liste presque exhaustive) et les ports dits éphémères, compris entre 1024 et 65535. Ce découpage est assez théorique car de nombreux ports supérieurs à 1024 sont pris par des protocoles, par exemple ircd qui écoute sur le 6667 ou git sur le 9418. Comme chacun le sait, sous Unix l'utilisation des ports inférieurs à 1024 est réservée à l'utilisateur root.

Une fois ce choix effectué, il reste à démarrer la numérotation des segments TCP avec un beau zéro (ou à reprendre le dernier numéro utilisé dans une communication identique – cf plus bas la partie théorique) dans le champ « Sequence number » qui est la marque d'un début de communication. De la même façon le champ « SYN flag » sera mis à 1 ce qui indique qu'il s'agit d'une demande de synchronisation. En effet, toute communication TCP commence par un échange de synchronisation/accusé de réception nommé en général SYN/ACK pour SYNchronisation/ACKnowledgement. Cet échange est résumé dans le schéma de la Figure 4 (le temps va du haut vers le bas) :

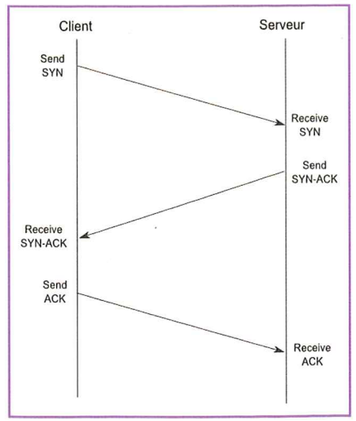


Fig. 4 : Echange SYN/ACK

Nous retrouvons ceci dans les captures effectuées :

1. Send SYN

```
No.     Time       Source        Destination   Protocol Length       Info
 5  4.155814000  172.20.1.10  172.20.0.200  TCP        74        51235 > http [SYN]

Transmission Control Protocol, Src Port: 51235 (51235), Dst Port: http (80), Seq: 0, Len: 0
  Source port: 51235 (51235)
  Destination port: http (80)
  [Stream index: 0]
  Sequence number: 0    (relative sequence number)
  Header length: 40 bytes
  Flags: 0x002 (SYN)
        000. .... .... = Reserved: Not set
        ...0 .... .... = Nonce: Not set
        .... 0... .... = Congestion Window Reduced (CWR): Not set
        .... .0.. .... = ECN-Echo: Not set
        .... ..0. .... = Urgent: Not set
        .... ...0 .... = Acknowledgment: Not set
        .... .... 0... = Push: Not set
        .... .... .0.. = Reset: Not set
        .... .... ..1. = Syn: Set
        .... .... ...0 = Fin: Not set
```

où l'on voit le port source choisi qui est le 51235, le port destination 80 comme prévu, le n° de séquence à 0 (Seq: 0) ainsi que le flag SYN mis seul à 1.

2. Receive SYN qui s'est certainement bien déroulé (je n'ai pas effectué de captures côté serveur) car la suite se déroule normalement.

3. Send SYN-ACK.

```
No.     Time       Source        Destination   Protocol Length Info
 6  4.156115000  172.20.0.200  172.20.1.10   TCP        78   http > 51235 [SYN, ACK]
Transmission Control Protocol, Src Port: http (80), Dst Port: 51235 (51235), Seq: 0, Ack: 1, Len: 0
    Source port: http (80)
    Destination port: 51235 (51235)
    [Stream index: 0]
    Sequence number: 0    (relative sequence number)
    Acknowledgment number: 1    (relative ack number)
    Header length: 44 bytes
    Flags: 0x012 (SYN, ACK)
        000. .... .... = Reserved: Not set
        ...0 .... .... = Nonce: Not set
        .... 0... .... = Congestion Window Reduced (CWR): Not set
        .... .0.. .... = ECN-Echo: Not set
        .... ..0. .... = Urgent: Not set
        .... ...1 .... = Acknowledgment: Set
        .... .... 0... = Push: Not set
        .... .... .0.. = Reset: Not set
        .... .... ..1. = Syn: Set
        .... .... ...0 = Fin: Not set
```

Nous avons donc la réponse du serveur envoyée sur notre port 51235 depuis son port 80 avec le n° de séquence toujours à 0 et nos deux flags ACK et SYN à 1.

4. Receive SYN-ACK.

5. Send ACK.

```
No.     Time     Source          Destination     Protocol     Length  Info
7 4.156149000    172.20.1.10     172.20.0.200    TCP            66    51235 > http [ACK]
Transmission Control Protocol, Src Port: 51235 (51235), Dst Port: http (80), Seq: 1, Ack: 1, Len: 0
    Source port: 51235 (51235)
    Destination port: http (80)
    [Stream index: 0]
    Sequence number: 1    (relative sequence number)
    Acknowledgment number: 1    (relative ack number)
    Header length: 32 bytes
    Flags: 0x010 (ACK)
        000. .... .... = Reserved: Not set
        ...0 .... .... = Nonce: Not set
        .... 0... .... = Congestion Window Reduced (CWR): Not set
        .... .0.. .... = ECN-Echo: Not set
        .... ..0. .... = Urgent: Not set
        .... ...1 .... = Acknowledgment: Set
        .... .... 0... = Push: Not set
        .... .... .0.. = Reset: Not set
        .... .... ..0. = Syn: Not set
        .... .... ...0 = Fin: Not set
```

La seule différence est le flag ACK seul à 1. Nous venons donc d'établir la communication.

Dans les captures ci-dessus j'ai volontairement omis les couches frame, ethernet et IP qui n'auraient fait ici qu'embrouiller les choses. Sachez cependant que les paquets TCP vus ici sont bien encapsulés chacun dans un paquet IP lui-même encapsulé dans une trame ethernet, elle-même transformée en impulsions sur le câble, ouf !

2.2 Petit passage par la théorie

2.2.1 En-tête TCP

Nous venons d'avoir un aperçu des trames TCP mais en fait qu'en est-il réellement ? Essayons de regarder plus en avant et d'expliquer les choses. Ce que nous appelons « trame » TCP devrait être nommé « segment » TCP si on veut respecter le vocabulaire de la RFC. Un segment TCP est constitué du header et des données. Le header est constitué de 20 octets auxquels on peut rajouter des options. Il se présente sous la forme visible Figure 5.

Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Source port																Destination port															
4	32	Sequence number																															
8	64	Acknowledgment number (if ACK set)																															
12	96	Data offset			Reserved 0 0 0				NS	CWR	ECE	URG	ACK	PSH	RST	SYN	FIN	Window Size															
16	128	Checksum																Urgent pointer (if URG set)															
20	160	Options (if *data offset* > 5. Padded at the end with "0" bytes if necessary.)																															
...	...	...																															

Fig. 5 : En-tête TCP

- « *Source port* », « *Destination port* » : on retrouve nos deux ports de source et de destination, 16 bits chacun.
- « *Sequence number* » : le numéro de séquence déjà rencontré lors du processus SYN/ACK. En fait cette valeur sur 32 bits sert aussi lors d'un transfert de données (flag SYN à 0). Dans ce cas il contient le numéro du premier octet de la zone de données du segment. Un segment transporte en général plusieurs octets de données et ils sont tous numérotés. La zone « Sequence number » contient le numéro du premier de ces octets. La valeur dans cette zone n'est pas remise à zéro à chaque début de transmission. Dans un segment numéro zéro (premier segment de SYN/ACK donc) la zone « Sequence Number » peut parfaitement contenir un nombre nettement supérieur. Ce qui va compter pour déterminer le bon numéro de séquence sera la différence entre la valeur en cours et celle du segment numéro zéro.
- « *Acknowledgement number* » : contient le numéro du premier octet de données attendu dans la réponse. Le même principe de relativité que pour la zone précédente est appliqué. Cela semble compliqué, ça l'est un peu mais on va y revenir dans la suite de l'exemple.
- « *Data offset* » : 4 bits qui permettent de décaler le début des données par rapport à la fin normale de l'entête afin d'y positionner des options supplémentaires. Il y est indiqué le nombre de mots de 32 bits qui composent l'en-tête et les éventuelles options.
- « *Flags* » sur les 12 bits restants dont voici la signification :
  - *Reserved* : réservé, toujours à 0.
  - NS (souvent noté ECN), CWR et ECE servent à gérer les congestions. Il s'agit de moments où le lien réseau est saturé et il faut que les stations soit ralentissent, soit bloquent pour un temps les nouvelles connexions. Afin que cela se fasse en bonne harmonie, les RFC 3168 et 3540 ont rajouté ces trois flags.
  - URG (« *URGent* ») à 1 signale que le champ « Urgent pointer » est significatif.
  - ACK (« *ACKnowledge* ») à 1 signale que le champ « Acknowledgement » doit être pris en considération. On verra que c'est le cas dans 95 % des échanges.
  - PSH (« *PuSH* ») à 1 demande explicitement à ce que les données en cache soient envoyées à l'application. En gros on vide le cache de données vers la couche 5.
  - RST (« *ReSeT* ») à 1 demande un reset de connexions.
  - SYN (« *SYNchronize* ») est utilisé pour démarrer une connexion.
  - FIN (« *FINalise* » sert à terminer une connexion.
- « *Window Size* » sur 16 bits permet de changer la taille de la fenêtre de réception. En théorie, à chaque segment reçu, le récepteur doit envoyer un ACK à l'émetteur. Cela sécurise la connexion mais ralentit le débit. Dans un souci de pouvoir augmenter la vitesse, on peut avec ce champ signifier à l'émetteur qu'il ne recevra un ACK que tous les N octets de donnée. Les 16 bits permettent de monter à une fenêtre de 64Ko. Pour augmenter encore cette fenêtre, on peut positionner une option – dans la fameuse zone optionnelle entre l'offset 160 et le début des données – afin de multiplier la taille de cette fenêtre et de l'augmenter jusqu'à 1Go. Cette action peut avoir un effet de bord non nul. En effet l'émetteur va envoyer des octets jusqu'à atteindre la taille de cette fenêtre afin de recevoir un ACK. S'il a besoin d'envoyer 312 octets et que la fenêtre est à 32Ko, il va devoir envoyer 32455 octets de « bourrage ». Il faut donc utiliser cette possibilité avec modération.
- « *Checksum* » : comme son nom l'indique, il permet d'envoyer une somme de contrôle concernant l'entête et les données.
- « *Urgent* » permet de désigner le début d'une zone de données urgentes – si le flag URG est mis à 1 – dans la zone de données du segment. Ce pointeur est un offset par rapport au numéro de séquence.

Enfin nous avons éventuellement la zone des options qui sera terminée par un bourrage à coup de zéro afin de finir sur des frontières de 32 bits.

Voila pour l'en-tête TCP. Avant de continuer l'analyse de notre connexion HTTP, regardons rapidement aussi les entêtes IP et ethernet.

2.2.2 En-tête IP

L'en-tête IP ressemble pas mal à l'en-tête TCP comme on peut le voir Figure 6.

Il tient lui aussi sur 20 octets, plus un champ « Options » facultatif. Les divers champs sont les suivants :

- « *Version* » : version d'IP donc 4 en général. L'entête IPv6 est différent.
- « *IHL* » pour Internet Header Length. Indique la taille du header en mots de 32 bits et permet de rajouter les options. Même mode de fonctionnement que le champ « Data offset » du header TCP.
- « *DSCP* » pour Differentiated Services Code Point : zone utilisée par les applications ayant besoin de temps réel, comme la VoIP par exemple. Pour plus d'informations voir la RFC 2474.
- « *ECN* » pour Explicit Congestion Notification. Permet de gérer les problèmes de congestion aussi au niveau IP. Les deux interlocuteurs doivent être capables de l'utiliser et d'accord pour le faire.
- « *Total Length* » indique la taille total du paquet, header plus données. Sachant qu'un en-tête fait minimum 20 octets, la taille maximum des données en IP sera de 65515 octets, cependant la plus petite taille qui doit être supportée par un équipement est de 576 octets (raisons historiques), header compris. Cet aspect des choses peut mener à des besoins de fragmenter les datagrammes IP afin qu'ils passent les routeurs.
- « *Identification* » permet d'identifier de façon univoque un paquet IP. A chaque envoi de datagramme en provenance du même émetteur ce champ sera incrémenté. Il est très utile lors du ré-assemblage des paquets fragmentés.
- « *Flags* » sur trois bits :
  - Bit 0 non utilisé, toujours à 0 ;
  - Bit 1 : Don't fragment (DF) ;
  - Bit 2 : More fragments (MF) sous entendu « are coming ».

Si le bit DF est mis à 1 et qu'un équipement requiert une fragmentation, le paquet est purement et simplement droppé. Ce flag est utilisé principalement par des outils comme traceroute.

Pour les paquets non fragmentés, le bit MF est à zéro ainsi que le champ « Fragment Offset ». Pour les paquets fragmentés, tous les fragments ont le bit MF à 1 sauf le dernier. La différence est faite grâce au champ « Fragment Offset » qui n'est pas à zéro dans ce cas.

- « *Fragment Offset* » indique l'offset du fragment en cours par rapport au premier octet du premier fragment.
- « *TTL* » pour Time To Live. Ce champ permet d'éviter qu'un paquet mal routé tourne sur le réseau pour l'éternité. La valeur du TTL est décrémentée par chaque routeur que le paquet traverse. Quand la valeur tombe à zéro le paquet est droppé et le routeur avertit l'émetteur via un message ICMP de type « Time Exceeded ».
- « *Protocol* » : dans cette zone on trouve le numéro du protocole sous-jacent tel que défini par l' « Internet Assigned Numbers Authority ».

Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Version				IHL				DSCP						ECN		Total Length															
4	32	Identification																				Flags			Fragment Offset							
8	64	Time To Live								Protocol								Header Checksum															
12	96	Source IP Address																															
16	128	Destination IP Address																															
20	160	Options (if IHL > 5)																															

Fig. 6 : En-tête IP

Par exemple TCP aura le numéro `0x06`, ICMP le numéro `0x01`, etc. La liste complète est disponible sur wikipedia : http://en.wikipedia.org/wiki/List_of_IP_protocol_numbers

- « *Header Checksum* » : somme de contrôle du header qui est contrôlée et recalculée par chaque routeur. En effet le fait de décrémenter le TTL change le checksum. Si la vérification est négative, le paquet est droppé.
- « *Source Address* » et « *Destination Address* » sont les adresses IP de l'émetteur et du récepteur. Elles sont susceptibles de changer en cas de NAT[3] par un routeur.

Ensuite viennent les options, si elles existent, puis les données. Ces dernières sont constituées dans le cadre de notre exemple, par l'en-tête TCP plus les données réelles. On a là la première vraie encapsulation.

2.2.3 En-tête ethernet

Terminons cette étude des en-têtes par celui que rajoute ethernet. Dans les faits ethernet rajoute un header et un trailer. Pour une meilleure lecture, la trame ethernet a été découpée en trois parties sur la Figure 7 mais elles sont bien entendu collées les unes aux autres dans la réalité.

Le header est constitué de cinq champs obligatoires (*Preamble*, *Start of frame delimiter*, *MAC destination*, *MAC source* et *Ethertype*) au milieu desquels peuvent s'insérer des champs optionnels dont nous avons un exemple ici avec le champ 802.1Q tag. Viennent ensuite les données et les deux champs du trailer.

- « *Preamble* » : le préambule est une succession de sept octets de valeur `0x55`. `0x55` hexa donne 0b10101010 en binaire ce qui aura pour effet de générer un signal carré sur le câble indiquant ainsi une tentative de communication.
- « *Start of Frame Delimiter* » (SFD) est là pour casser le rythme volontairement et il vaut `0x5D` soit 0b10101011.

Preamble	Start of frame delimiter	MAC destination	MAC source	802.1Q tag (optional)	Ethertype (Ethernet II) or length (IEEE 802.3)
7 octets	1 octet	6 octets	6 octets	(4 octets)	2 octets
Payload					
1500 octets					
Frame check sequence (32-bit CRC)				Interpacket gap	
4 octets				12 octets	

Fig. 7 : Trame ethernet avec header, payload et trailer

- « *MAC destination* » et « *MAC source* » sont les adresses MAC de destination et source et nous voyons enfin l'intérêt de la requête ARP décortiquée plus haut.
- « *Ethertype* » : ces deux octets servent ici à indiquer quel est le protocole encapsulé dans le champ « *Payload* ». Dans le cas d'un paquet IPv4 la valeur prendra `0x0800`, dans le cadre d'une requête ARP la valeur sera `0x0806`, s'il s'agit de Wake On Lan on trouvera `0x0842`, du DECNet engendrera une valeur de `0x6003`, etc. Ces valeurs sont données par convention et vous pourrez en trouver les principales sur wikipedia ici : http://en.wikipedia.org/wiki/Ethertype. Ceci est valable dans le cas de l'utilisation de la norme ethernet II, ce qui est notre cas, et aura une signification différente si l'on utilise une autre norme ethernet comme le 802.3 bien connue de ceux qui ont fait du réseau Novell.

Le champ optinnel ajouté ici à titre d'exemple est nommé « *802.1Q tag* » et est destiné aux réseaux virtuels (VLAN). Il sert principalement à stocker l'identifiant du VLAN (VLAN tagging) et possède un en-tête particulier qui permet de le distinguer. Nous avons là un exemple du fait qu'un header ethernet est à géométrie variable. Il est parfaitement possible de rajouter ou d'enlever des bouts en prenant moult précautions, du moment que les cinq champs principaux y sont.

Après le « *Payload* », ethernet rajoute deux champs, une somme de contrôle sous la forme d'un CRC et un séparateur de paquets destiné à faire une pause dans la transmission. Cette pause était minimum de 9.6us en 10Mbps elle n'est plus que de 96ns en gigabit.

Voilà pour la théorie, passage ardu mais indispensable pour comprendre la suite de notre exemple.

3 Retour à la pratique

3.1 Création d'une trame TCP/IP à la main

Avec ce que nous savons maintenant, pourquoi ne pas essayer de construire nous-mêmes à la main les paquets IP et TCP ? Nous pourrons ensuite comparer avec ce que *Wireshark* a récupéré sur le réseau. Il est bien évident que nous n'allons pas tomber exactement sur les mêmes paquets que ceux capturés car de nombreux paramètres sont dépendant du passé des échanges réseau et d'autres paramètres que nous ne maîtrisons pas.

3.1.1 En-tête TCP

Commençons donc par établir la requête HTTP correspondant à notre demande d'url http://172.20.0.200/main/main.html. Voici ce que Firefox a envoyé au serveur :

```
GET /main/main.html HTTP/1.1\r\n
Host: 172.20.0.200\r\n
User-Agent: Mozilla/5.0 (X11; Linux x86_64;
rv:22.0) Gecko/20100101 Firefox/22.0\r\n
Accept: text/html,application/
xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
Accept-Language: fr,en-us;q=0.7,en;q=0.3\r\n
Accept-Encoding: gzip, deflate\r\n
DNT: 1\r\n
Connection: keep-alive\r\n
\r\n
```

ce qui nous fait une zone de données de 313 octets.

[3] Network Address Translation. Votre firewall en fait en permanence :)

Ajoutons maintenant l'en-tête TCP en reprenant dans l'ordre les champs à remplir.

- Le port source sera celui déjà utilisé lors du SYN/ACK soit le 51235 ou **0xC823**.
- Le port destination sera bien entendu le 80 ou **0x0050**.
- Le numéro de séquence est relatif au premier paquet envoyé par le client lors du SYN/ACK, ce paquet étant le paquet numéro zéro. Dans notre cas nous allons démarrer avec un compteur à zéro et donc le champ vaudra **0x00000001**.
- Le numéro d'acknowledgement est aussi relatif et nous attendons un segment de réponse dont le premier octet de données sera le premier octet « utile » de cette transmission, il portera donc le numéro 1. Là encore nous supposons le compteur à zéro, ce champ vaudra donc lui aussi **0x00000001**.
- Le « *Data offset* » et les flags. Commençons par ces derniers. Nous sommes ici dans une transmission en cours – même si elle vient de débuter – et donc nous devons mettre le flag SYN à 0. Le flag ACK est à mettre à 1 car notre zone « *Acknowledgement number* » est significative ce qui sera le cas durant toute la durée de la communication. Laissons les trois flags de gestion de la congestion à 0, il n'y a rien de particulièrement urgent dans nos données donc URG est à 0. Nous envoyons un paquet de données provenant de la couche 5 à destination de la couche 5 du serveur, il serait donc souhaitable que la livraison se fasse, aussi mettons le flag PSH à 1. Nous ne réinitialisons (*reset*) pas la communication donc RST est à 0 et il ne s'agit pas d'une fin de communication, FIN est donc aussi à 0. Si on récapitule cela nous donne pour les flags :

Reserved			NS	CWR	ECE	URG	ACK	PSH	RST	SYN	FIN
0	0	0	0	0	0	0	1	1	0	0	0

Soit la valeur hexa sur 12 bits **0x018**.

Nous n'allons pas rajouter d'options, ne compliquons pas tout. La taille de l'en-tête sera donc de 20 octets soit encore 5 mots de 32 bits. La zone « *Data offset* » prendra donc la valeur **0x5**. Tout cela combiné, les deux premiers octets de ce champ valent donc **0x5018**.

- « Window size » : ce champ est complètement dynamique et recalculé à chaque segment envoyé. Sa valeur d'origine dépend principalement de la vitesse du réseau et du temps de ping entre les deux machines. Pour notre exemple nous allons le mettre à **0x0000** ce qui, dans la vraie vie, ne devrait pas se rencontrer, sauf à avoir une des deux machines carrément poussive. Pour ceux qui sont intéressés, une très bonne explication est disponible ici (en Anglais) : http://ithitman.blogspot.fr/2013/02/understanding-tcp-window-window-scaling.html.
- « *Checksum* » : le calcul de la somme de contrôle est relativement simple mais assez rébarbatif, nous allons donc, toujours dans le cadre de cet exercice, le laisser à **0x0000**.
- « *Urgent pointer* » n'est pas pertinent ici car le flag URG est à 0. Nous laissons là aussi le champ à **0x0000**.

Voici ce que donnera donc notre segment une fois tout converti en hexadecimal :

C8 23 00 50 00 00 00 01 00 00 00 01 50 18 00 00 00 00 00 00	En-tête
47 45 54 20 2F 6D 61 69 6E 2F 6D 61 69 6E 2E 68 74 6D 6C 20 48 54 54 50 2F 31 2E 31 0D 0A 48 6F 73 74 3A 20 31 37 32 2E 32 30 2E 30 2E 32 30 30 0D 0A 55 73 65 72 2D 41 67 65 6E 74 3A 20 4D 6F 7A 69 6C 6C 61 2F 35 2E 30 20 28 58 31 31 3B 20 4C 69 6E 75 78 20 78 38 36 5F 36 34 3B 20 72 76 3A 32 32 2E 30 29 20 47 65 63 6B 6F 2F 32 30 31 30 30 31 30 31 20 46 69 72 65 66 6F 78 2F 32 32 2E 30 0D 0A 41 63 63 65 70 74 3A 20 74 65 78 74 2F 68 74 6D 6C 2C 61 70 70 6C 69 63 61 74 69 6F 6E 2F 78 68 74 6D 6C 2B 78 6D 6C 2C 61 70 70 6C 69 63 61 74 69 6F 6E 2F 78 6D 6C 3B 71 3D 30 2E 39 2C 2A 2F 2A 3B 71 3D 30 2E 38 0D 0A 41 63 63 65 70 74 2D 4C 61 6E 67 75 61 67 65 3A 20 66 72 2C 65 6E 2D 75 73 3B 71 3D 30 2E 37 2C 65 6E 3B 71 3D 30 2E 33 0D 0A 41 63 63 65 70 74 2D 45 6E 63 6F 64 69 6E 67 3A 20 67 7A 69 70 2C 20 64 65 66 6C 61 74 65 0D 0A 44 4E 54 3A 20 31 0D 0A 43 6F 6E 6E 65 63 74 69 6F 6E 3A 20 6B 65 65 70 2D 61 6C 69 76 65 0D 0A 0D 0A	Données

3.1.2 En-tête IP

- « *Version* » : nous sommes en IPv4 donc **0x4**.
- « *IHL* » ou Header Length : 20 octets, soit 5 mots de 32 bits. La valeur est donc de **0x5**.
- « *DSCP* » : inutilisé ici, donc 0.
- « *ECN* » : pas de gestion de la congestion donc 0.
- « *Total Length* » : nous savons que nous avons 313 octets de données, 20 octets d'entête TCP et 20 octets d'entête IP soit un total de 353 octets, ou encore **0x0161**.
- « *Identification* » : ce numéro est censé suivre celui utilisé dans le dernier envoi dans le cadre du SYN/ACK qui était dans notre capture égal à **0xAC47**. Mettons donc **0xAC48**.
- « *Flags* » : ce champ occupe 3 bits sur 16. Laissons le bit « Reserved » à 0 et vu que notre segment n'est pas gros, demandons à ne pas fragmenter en posititionnant le flag « Don't Fragment » à 1 et logiquement le flag « More Fragments » sera à 0.
- « *Fragment Offset* » : il occupe les 12 bits restant. Comme on n'est pas fragmenté, on le met à 0.

  En regroupant ces deux champs pour former le mot de 16 bits, cela nous donne la valeur de 0b0100 0000 0000 0000 soit **0x4000**.
- « *Time To Live* » : mettons une valeur standard de 64 soit **0x40**.
- « *Protocol* » : le protocole encapsulé est TCP donc **0x06**.
- « *Header Checksum* » : laissons-le à 0 pour notre exemple, soit **0x0000**.

- « *Source IP Address* » : 172.20.1.10 soit **0xAC14010A**.
- « *Destination IP Address* » : 172.20.0.200 soit **0xAC1400C8**.

et voilà. Cela va donc nous faire la série suivante :

```
45 00 01 61 AC 48 40 00 40 06 00 00 AC 14 01 0A AC 01 00 C8
```

3.2 Vérifions...

Maintenant reprenons la capture effectuée et regardons si nous avons des erreurs en dehors des checksums et autres champs que nous avons positionné de manière arbitraire. Voici la séquence capturée pour l'en-tête TCP :

```
C8 23 | 00 50 | 59 29 95 2F | DB CB 8F 81 | 80 | 18 | 00 E5 | 5B 5A | 00 00 | 01
01 08 0A 13 50 39 6A 28 E2 EF 5A
```

et la nôtre

```
C8 23 | 00 50 | 00 00 00 01 | 00 00 00 01 | 50 | 18 | 00 00 | 00 00 | 00 00
```

déjà on n'a pas la même longueur... Argh ou pas argh ? Regardons ce que *Wireshark* nous a décodé :

```
Transmission Control Protocol
    Source port: 51235 (51235)
    Destination port: http (80)
    Sequence number: 1    (relative sequence number)
    Acknowledgment number: 1    (relative ack number)
    Header length: 32 bytes
    Flags: 0x018 (PSH, ACK)
        000. .... .... = Reserved: Not set
        ...0 .... .... = Nonce: Not set
        .... 0... .... = Congestion Window Reduced (CWR): Not set
        .... .0.. .... = ECN-Echo: Not set
        .... ..0. .... = Urgent: Not set
        .... ...1 .... = Acknowledgment: Set
        .... .... 1... = Push: Set
        .... .... .0.. = Reset: Not set
        .... .... ..0. = Syn: Not set
        .... .... ...0 = Fin: Not set
    Window size value: 229
    Checksum: 0x5b5a [validation disabled]
    Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
        No-Operation (NOP)
            Type: 1
                0... .... = Copy on fragmentation: No
                .00. .... = Class: Control (0)
                ...0 0001 = Number: No-Operation (NOP) (1)
        No-Operation (NOP)
            Type: 1
                0... .... = Copy on fragmentation: No
                .00. .... = Class: Control (0)
                ...0 0001 = Number: No-Operation (NOP) (1)
        Timestamps: TSval 324024682, TSecr 685961050
            Kind: Timestamp (8)
            Length: 10
            Timestamp value: 324024682
            Timestamp echo reply: 685961050
```

Déjà on remarque que nous avons des options dans la vraie vie, cela explique peut-être la différence de longueur.

Reprenons dans l'ordre :

- Ports source et destination : 51235 (**0xC823**) et 80 (**0x0050**). Jusque là ça colle.
- *Sequence number* : **0x00000001** pour nous et **0x5929952F** pour *Wireshark*. Aucune importance, il s'agit comme déjà expliqué, de numéros relatifs. On suppose donc que le segment d'avant portait le numéro **0x5929952E**, ce que valide notre capture d'écran de la Figure 8.
- *Acknowledgement number* : **0x00000001** pour nous, **0xDBCB8F81** pour *Wireshark*. Même problème que ci-dessus, il s'agit toujours d'une numérotation relative.
- *Data offset* : **0x5** (20 octets) pour nous et **0x8** (32 octets) pour *Wireshark*. Nous pouvons vérifier que la version de Wireshark comporte 12 octets d'options ce qui amène bien la taille de l'en-tête TCP à 32 octets, CQFD.
- *Flags* : **0x18** pour tout le monde.

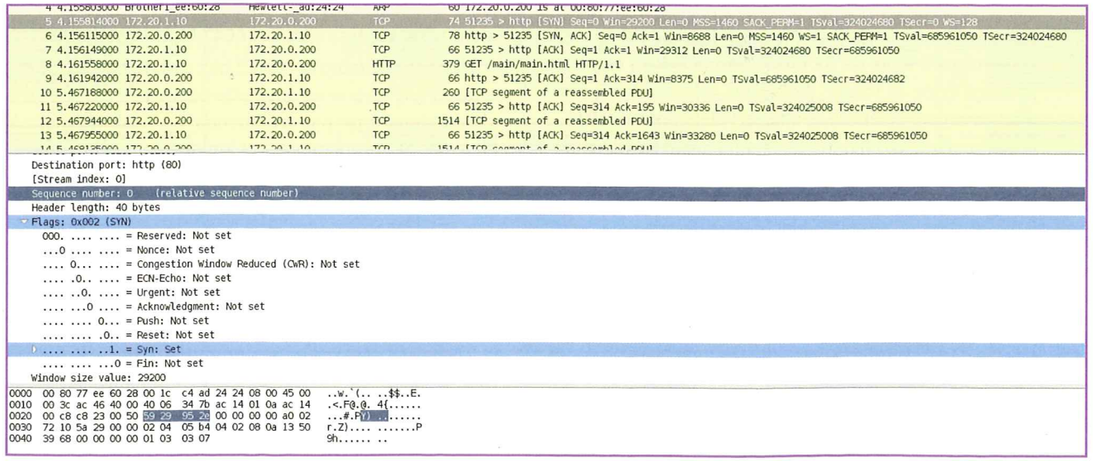


Fig. 8 : Valeurs des champs SEQ et ACK dans la vraie vie

- *Window size* : **0x00E5** soit 229 pour *Wireshark*, nous l'avions laissé volontairement à zéro.
- *Checksum* : **0x5B5A** pour *Wireshark*, on va le croire sur parole, nous avions là aussi laissé zéro volontairement.

Viennent ensuite les douze octets d'options TCP, nous n'avions rien mis.

En résumé nous pouvons dire que nous n'avons pas fait d'erreur avec les données que nous possédions.

Regardons maintenant l'en-tête IP :

```
45 | 00 | 01 6D | AC 48 | 40 00 | 40 | 06 | 33 48 | AC 14 01 0A | AC 14 00 C8
```

et la nôtre

```
45 | 00 | 01 61 | AC 48 | 40 00 | 40 | 06 | 00 00 | AC 14 01 0A | AC 01 00 C8
```

C'est déjà beaucoup plus ressemblant. Regardons aussi champ par champ :

- *Version* : **0x4** pour tout le monde.
- *IHL* : **0x5**
- *DSCP/ECN* : **0x00**
- *Total Length* : **0x0161** pour nous et **0x016D** pour *Wireshark*. 0x016D-0x0161=0x0C soit 12 soit la longueur rajoutée par les options dans l'en-tête TCP. Tout va bien.
- *Identification* : **0xAC48** mais là on avait triché et regardé sur la capture :)
- *Flags/Fragment offset* : **0x4000** ok
- *TTL* : **0x40**, TTL « standard »
- *Protocol* : **0x06** soit TCP
- *Checksum* : **0x3348** pour *Wireshark* mais il aurait été différent pour nous. Nous l'avions laissé à zéro volontairement.
- *Source* et *Destination IP address* : identiques, on sait encore convertir du décimal en hexa, chouette !

Donc au final, ce petit exercice s'est plutôt bien déroulé. Nous avons pu mettre l'accent sur la difficulté d'obtenir certaines informations car nous ne sommes pas directement immergé dans le flux des données mais le principe est là. Nous n'allons pas nous appesantir sur ce que rajoute la couche ethernet, il s'agit principalement comme nous l'avons vu des adresses MAC correspondant aux adresses IP ainsi qu'un checksum (encore un...) et quelques options éventuelles.

4 Retour sur les numéros de séquence et d'acknowledge, suite de la transmission

La notion de numérotation relative n'est pas facile à comprendre. Pour ce faire, un petit schéma vaut souvent mieux qu'un grand discours et la Figure 9 regroupe tout le début de communication avec indication de la valeur – absolue et relative – des champs SEQ et ACK ainsi que la taille des données.

Dans le schéma ci-dessous, les valeurs hexadécimales à rallonge ont été écrites avec des espaces entre les octets pour faciliter leur lecture mais il s'agit bien de mots de 32 bits et non de quatre octets sans lien entre eux ce qui, vu par *Wireshark*, donne la capture visible en Figure 10 (trames de 5 à 13).

Au commencement de la transmission, la trame SYN envoyée par le client indique aux deux protagonistes que les numéros de séquence (SEQ#) et d'acknowledge (ACK#) doivent être considérés comme étant à 0. Or l'on voit bien que le nombre stocké dans la trame est très grand, **0x5929952E** (1495897390 décimal) comme numéro de séquence côté client et **0xDBCB8F80** (3687550848 décimal) côté serveur. Ces deux nombres vont être considérés ensuite comme le zéro de la transmission et les valeurs de SEQ# seront calculées par rapport à ces nombres de base. Par exemple, si on prend le SEQ# côté client après le

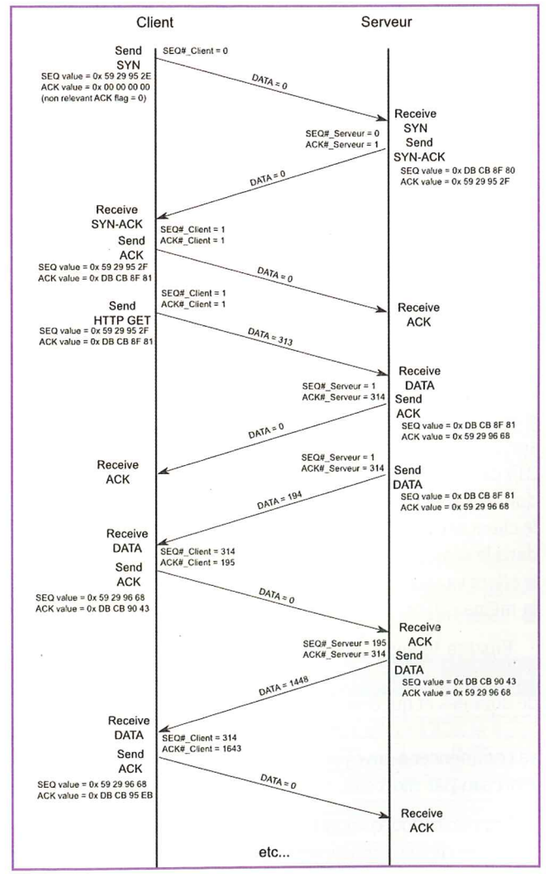


Fig. 9: Vue d'ensemble d'un échange TCP avec valeurs absolues et relatives des champs SEQ et ACK

No.	Time	Source	Destination	Protocol	Length	Info
5	4.155814000	172.20.1.10	172.20.0.200	TCP	74	51235 > http [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=324024680 TSecr=0 WS=128
6	4.156115000	172.20.0.200	172.20.1.10	TCP	78	http > 51235 [SYN, ACK] Seq=0 Ack=1 Win=8688 Len=0 MSS=1460 WS=1 SACK_PERM=1 TSval=685961050 TSecr=324024680
7	4.156149000	172.20.1.10	172.20.0.200	TCP	66	51235 > http [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=324024680 TSecr=685961050
8	4.161558000	172.20.1.10	172.20.0.200	HTTP	379	GET /main/main.html HTTP/1.1
9	4.161942000	172.20.0.200	172.20.1.10	TCP	66	http > 51235 [ACK] Seq=1 Ack=314 Win=8375 Len=0 TSval=685961050 TSecr=324024682
10	5.467188000	172.20.0.200	172.20.1.10	TCP	260	[TCP segment of a reassembled PDU]
11	5.467220000	172.20.1.10	172.20.0.200	TCP	66	51235 > http [ACK] Seq=314 Ack=195 Win=30336 Len=0 TSval=324025008 TSecr=685961050
12	5.467944000	172.20.0.200	172.20.1.10	TCP	1514	[TCP segment of a reassembled PDU]
13	5.467955000	172.20.1.10	172.20.0.200	TCP	66	51235 > http [ACK] Seq=314 Ack=1643 Win=33280 Len=0 TSval=324025008 TSecr=685961050
14	5.468135000	172.20.0.200	172.20.1.10	TCP	1514	[TCP segment of a reassembled PDU]
15	5.468149000	172.20.1.10	172.20.0.200	TCP	66	51235 > http [ACK] Seq=314 Ack=3091 Win=36096 Len=0 TSval=324025008 TSecr=685961050
16	5.468380000	172.20.0.200	172.20.1.10	TCP	1514	[TCP segment of a reassembled PDU]
17	5.468393000	172.20.1.10	172.20.0.200	TCP	66	51235 > http [ACK] Seq=314 Ack=4539 Win=39040 Len=0 TSval=324025008 TSecr=685961050
18	5.468601000	172.20.0.200	172.20.1.10	TCP	1514	[TCP segment of a reassembled PDU]
19	5.468614000	172.20.1.10	172.20.0.200	TCP	66	51235 > http [ACK] Seq=314 Ack=5987 Win=41856 Len=0 TSval=324025008 TSecr=685961050
20	5.469162000	172.20.0.200	172.20.1.10	TCP	1514	[TCP segment of a reassembled PDU]

Fig. 10 : Capture Wireshark de l'échange des données

premier envoi de données par le serveur, il est à 314. Le nombre stocké dans la trame à ce moment là est **0x59299668**. **0x59299668**-**0x5929952E** = 314. Youpi on est bon. Cette arithmétique se retrouve à chaque trame pour les SEQ# et les ACK#, chaque machine tenant son compteur à jour.

Nous remarquons aussi dès la trame SYN-ACK (la deuxième) que les deux machines récupèrent le SEQ# de l'autre pour le mettre dans son ACK# en incrémentant un coup car, même s'il n'y a pas eu de données transmises, seule la première trame (SYN) positionne le zéro.

A la suite de cela, les SEQ# et ACK# vont évoluer en fonction des données envoyées par les deux machines. Par exemple la requête GET occupe 313 octets, les deux compteurs sont à la valeur relative 1, donc le serveur – qui n'a toujours rien envoyé comme données – va garder son SEQ# à 1 mais passer son ACK# à 314 car le premier octet des prochaines données potentiellement envoyées par le client sera le 314e de la transmission dans le sens client -> serveur. De même le client va passer son SEQ# à 314 pour la même raison.

Ensuite le serveur, après sa trame d'acknowledge qui ne comporte pas de données et qui donc n'influe pas sur SEQ# et ACK# ni d'un côté ni de l'autre, va commencer à envoyer la page HTML morceau par morceau.

Le premier bout comportera 194 octets et donc le client va conserver son SEQ# à 314 (il n'a toujours pas envoyé de données depuis le **GET**) et passer son ACK# à 195.

Le client envoie sa trame d'acknowledgement, puis le serveur envoie 1448 octets de données. Donc le SEQ# du client va rester à 314 mais son ACK# va passer à 1643 soit 1448 (cette trame) +194 (trame précédente) +1 (premier octet du futur envoi du serveur). Côté serveur le SEQ# sera à 195 (numéro du premier octet de données) et l'ACK# sera toujours à 314 car le client n'a pas envoyé de données.

Cette séquence se répétera ainsi jusqu'à la fin de transmission. Le principe à retenir est le suivant :

- SEQ# = numéro du premier octet de la zone de données que je viens d'envoyer.
- ACK# = numéro du premier octet de la zone de données que j'attends éventuellement de la part de mon correspondant.

Ces deux numéros sont stockés dans les trames sous la forme d'un compteur sur 32 bits qui ne repasse à zéro que par overflow.

5 That's all folks

Pour signaler la fin d'une transmission, il existe là encore un protocole d'échange assez simple visible Figure 11 basé sur les flags.

Le serveur met le flag FIN à 1 (en plus des autres), le client acknowledge le FIN en revoyant ACK à 1 et FIN à 0. Dans cet état le client ne peut plus recevoir de données du serveur mais il peut encore en envoyer. Pour clore complètement la connexion, le client envoie FIN à 1 auquel le serveur répond avec juste ACK à 1.

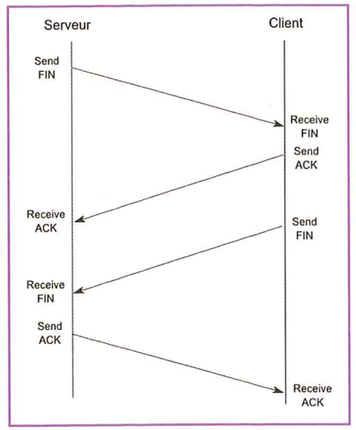


Fig. 11 : Echange de fin de transmission TCP

Une autre possibilité de plus en plus répandue consiste pour le client à renvoyer simultanément FIN et ACK à 1 ce à quoi le serveur répondra en envoyant juste ACK à 1 comme le montre la capture de la Figure 12.

En même temps que la fin des données HTTP (ligne 26) le serveur a renvoyé le flag FIN à 1, ce à quoi le client à répondu avec FIN/ACK, ce qui fut validé par le ACK du serveur.

6 Et si on creuse encore

Nous sommes arrivés maintenant à pouvoir transformer une requête HTTP simple en suite d'octets tout en respectant les protocoles et leurs encapsulations. Mais que se passe-t'il au niveau hardware ? Comment tout cela se déroule ?

26	5.478070000	172.20.0.200	172.20.1.10	HTTP	1145 HTTP/1.0 200 OK (text/html)
27	5.478298000	172.20.1.10	172.20.0.200	TCP	66 51235 > http [FIN, ACK] Seq=314 Ack=11217 Win=53504 Len=0 TSval=324025011 TSecr=685961100
28	5.479171000	172.20.0.200	172.20.1.10	TCP	66 http > 51235 [ACK] Seq=11217 Ack=315 Win=8374 Len=0 TSval=685961100 TSecr=324025011

Fig. 12 : Capture d'un échange de fin de transmission réduit au FIN+ACK/ACK

Le protocole ethernet utilise pour accéder au réseau la technique dite de Carrier Sense Multiple Acces with Collision Detection ou CSMA/CD. A l'époque des dinosaures, quand le réseau plafonnait à 10Mbps et que l'on utilisait du câble coaxial pour communiquer, toutes les stations d'un même brin réseau était physiquement « ensemble » sur le câble. Quand l'une d'elles voulait parler, elle commençait par écouter si une conversation était déjà en cours. Si oui elle attendait un « certain temps » puis réessayait. Si tout était calme elle se mettait à émettre. Cependant, il se pouvait que deux stations se mettent à émettre à la même fraction de seconde ce qui générait une collision. Celle-ci était détectée par la lecture de la valeur de la tension continue sur le câble. Si celle-ci était nettement supérieure (en valeur absolue) à +/- 0.85 volts, une collision était détectée. Dans ce cas, la station qui détectait la première ce phénomène renforçait la collision en envoyant un motif de 32 bits bien spécifique afin d'être sûre que toutes les stations connectées ont compris le problème. Ceci fait, les stations attendaient un délai aléatoire avant de recommencer.

Maintenant que nous travaillons en étoile avec des paires torsadées, il suffit à la station émettrice de constater qu'elle reçoit des données avant même d'avoir fini d'émettre pour conclure à une collision.

Contrairement au coaxial, la transmission sur paires torsadées se fait en différentiel, c'est à dire que la différence de tension est prise entre le fil + et le fil – et non pas entre le fil et la masse. Cette amélioration diminue fortement la sensibilité aux signaux parasites qui sont eux toujours référencés à la masse.

Les différentes versions d'ethernet (10Base-2, 10Base-T, 100Base-TX, etc.) ont vues aussi leurs protocoles de communication évoluer en ce qui concerne le mode d'encodage sur le fil. Par exemple le 10Base-T est encodé en code Manchester avec le bit le moins significatif en premier. A l'oscilloscope ça donne la courbe de la Figure 13.

Pour ceux qui veulent aller plus loin, le net est rempli d'informations mais il y a aussi Tektronix qui a sorti une brochure d'aide à l'utilisation de leurs oscilloscopes pour debugger ethernet. Elle est disponible ici : http://www.testequity.com/documents/pdf/ethernet-debugging.pdf.

Pour terminer avec cet aperçu du hardware, un rappel des normes – oui il y en a deux, sinon ce ne serait pas drôle – de branchement des prises ethernet en paires torsadées est visible en Figure 14.

La norme la plus répandue, du moins par chez nous, est la 568B mais vous choisissez celle que vous voulez, le tout est de vous y tenir.

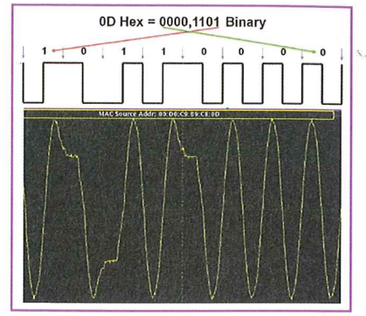


Fig. 13 : Signal ethernet 10Base-T vu à l'oscilloscope

Conclusion

Cet article ne prétend absolument pas être exhaustif concernant le protocole TCP/IP mais il est souvent bon de connaître le dessous des cartes et savoir ce qui se trame entre le kernel, la carte réseau et le câble mural. Ces connaissances permettent parfois de débrouiller des pannes qui font de la résistance. Vous voilà maintenant mieux armé pour comprendre ce que Wireshark ou tcpdump vous racontent quand vous chercherez à comprendre pourquoi cette $£%%*! de connexion ne s'établit pas comme elle devrait.

Pour ceux qui souhaitent aller plus loin, le net est rempli de sites plus ou moins bien faits et plus ou moins à jour mais qui offrent presque tous des informations intéressantes sur tel ou tel aspect de ce protocole assez tentaculaire. Maintenant il ne reste plus qu'à passer tout ça en IPv6, vaste programme... ■

RJ45 Pin #	Wire Color (T568A)	Wire Diagram (T568A)	10Base-T Signal 100Base-TX Signal	1000Base-T Signal
1	White/Green		Transmit+	BI_DA+
2	Green		Transmit-	BI_DA-
3	White/Orange		Receive+	BI_DB+
4	Blue		Unused	BI_DC+
5	White/Blue		Unused	BI_DC-
6	Orange		Receive-	BI_DB-
7	White/Brown		Unused	BI_DD+
8	Brown		Unused	BI_DD-

Norme 568A

RJ45 Pin #	Wire Color (T568B)	Wire Diagram (T568B)	10Base-T Signal 100Base-TX Signal	1000Base-T Signal
1	White/Orange		Transmit+	BI_DA+
2	Orange		Transmit-	BI_DA-
3	White/Green		Receive+	BI_DB+
4	Blue		Unused	BI_DC+
5	White/Blue		Unused	BI_DC-
6	Green		Receive-	BI_DB-
7	White/Brown		Unused	BI_DD+
8	Brown		Unused	BI_DD-

Norme 568B

Fig. 14 : Normes de câblage des paires torsadées en ethernet

SAMA5D3 XPLAINED : UN DEVKIT ARM CORTEX-A5 UN PEU ATYPIQUE

par Denis BODOR

Il existe aujourd'hui tant de devkits et de plateformes que l'amateur éclairé tout comme le professionnel aura bien du mal à s'y retrouver. Bien que l'ensemble du marché soit totalement dominé par ARM, la diversité est plus que jamais à l'ordre du jour. Parmi tous les compétiteurs, certains jouent la carte de la puissance et d'autres, plus rares, celles des fonctionnalités avec tantôt des choix surprenants.

La carte qui nous intéresse ici est la SAMA5D3 Xplained d'Atmel. Elle fait partie d'une série de devkits nommée « Xplained » permettant aux développeurs de prendre la main rapidement sur une plateforme, qu'il s'agisse d'un microcontrôleur AVR 8 bits (ATmega328P Xplained Mini), jusqu'au Cortex-A5 (SAMA5D3 Xplained) en passant par toute une gamme à base de cœur Cortex-M0 (SAM D21 Xplained par exemple) ou Cortex-M4 (SAM4* Xplained).

La bête qui nous occupe ici est un mélange étonnant difficile à classer dans une catégorie particulière :

- SoC SAMA5D36 à cœur ARM Cortex-A5 offrant un jeu d'instruction ARMv7-A Thumb2 et cadencé à 536 MHz,
- 256 Mo de mémoire DDR2,
- 256 Mo de flash NAND,
- un emplacement SD,
- deux ports USB hôtes,
- un port USB *device* avec connecteur micro-AB,
- une interface LCD,
- un port Gigabit Ethernet (10/100/1000),
- un port Ethernet (10/100),
- un connecteur JTAG,
- un port console,
- quelques leds et boutons,
- et un lot de connecteurs femelles formant un emplacement compatible Arduino R3 proposant GPIO, TWI (i2c), SPI, USART, UART, etc.

Ce dernier point est sans doute la partie la plus surprenante de la plateforme. Bien que l'intérêt de proposer un tel agencement de connecteurs soit évident, on peut se demander s'il y a ou non erreur sur le public visé par ce devkit. En effet, bien que la carte ne soit pas tolérante au 5V, une telle fonctionnalité permet d'utiliser une certaine partie des nombreux *shields* disponibles pour Arduino. Cependant, les autres caractéristiques de la SAMA5D3 Xplained, comme la flash NAND, le connecteur JTAG ou les deux ports Ethernet, ne semblent pas vraiment adresser les besoins des hobbyistes amateurs d'Arduino. Certes, l'approche consistant à équiper un devkit d'un emplacement compatible n'est pas nouveau, c'est un choix que même Intel semble considérer comme important pour sa Galileo (qui n'est pas forcément une référence en la matière, cf le connecteur jack RS232 pour la console série de la GEN1). Mais est-ce bien là quelque chose de techniquement logique ou d'une simple idée farfelue imposée par un « génie » du service marketing ? Mitigeons toutefois nos propos en rappelant que le but d'un devkit est l'évaluation et qu'historiquement ces produits sont très « chargés » en périphériques. De ce fait ils sont par définition coûteux pour le développeur. L'une des solutions pour rendre le produit financièrement plus accessible consiste à simplifier le devkit et à ajouter une connectique d'extension modulaire dont le choix découle du nombre d'extensions alors disponibles. Sur cette base, et sachant qu'au moment de la conception de cette carte Xplained il n'y avait pas de connecteur « HAT » (RPi), le choix d'un format Arduino est quelque chose qu'on peut comprendre, tout en ne l'appréciant pas (l'espace non standard entre les séries de connecteur 0 à 7 et 8 à 13 est tout bonnement détestable, et si un utilisateur ne veut pas faire l'effort de s'assurer de l'orientation d'un shield avant connexion, peut-être devrait-il se demander s'il veut vraiment s'investir dans le domaine).

Le paysage actuel, si l'on s'en tient au registre du budget par carte est, dans les grandes lignes, le suivant :

- ACME Arietta : 400 Mhz / 128 Mo = 20€
- Raspberry Pi A+ (toute récente) : 700 Mhz / 256 Mo = 21€
- Raspberry Pi B+ : 700 Mhz / 512 Mo = 40€
- Banana Pi : dual 1 Ghz / 1 Go = 50€
- BeagleBone Black : 1 Ghz / 512 Mo = 60€
- Olinuxino A20 (avec NAND) : dual 1 Ghz / 1 Go = 65€
- Galileo Gen2 : 400 MHz / 256 Mo = 70€
- SAMA5D3 Xplained : 536 MHz / 256 Mo = 80€
- Cubietruck : 1 GHz / 2 Go = 90€

Nous ne considérons, bien entendu, pas ici les fonctionnalités intégrées ou non (NAND, HDMI, GPU, bus, etc) mais nous nous en tenons simplement à la perception que peut avoir, justement, un hobbyiste débutant. Comme vous pouvez le constater, présenté de cette manière et en prenant en compte l'aspect promotionnel mis en oeuvre au delà des cercles des initiés, on comprend clairement pourquoi la Raspberry Pi s'est imposée aussi facilement et pourquoi la Banana Pi connaît actuellement un succès grandissant : on a plein de MIPS pour quelques euros !

Bien sûr, ce comparatif et ce classement ne sont valables qu'en faisant abstraction de la plupart des critères importants pour des utilisations spécifiques. Il est donc important de se souvenir qu'un devkit est avant toutes choses un kit d'évaluation permettant de juger de la pertinence d'un choix vis-à-vis d'un cahier des charges souvent bien précis. Ainsi, un certain nombre de points n'intéresse généralement pas l'utilisateur souhaitant avoir un usage générique du produit alors qu'il s'agit de fonctionnalités capitales dès lors qu'on souhaite réellement traiter d'embarqué (et non de nano-informatique).

La SAMA5D3 Xplained peut paraître être un choix peu judicieux pour un regard profane souvent ébloui par les gigaoctets de mémoire et la cadence des processeurs d'autres plateformes. Mais la présence de NAND, d'une interface Gigabit Ethernet, de deux bus CAN, d'une RTC intégrée, ou encore d'une interface LCD/TFT pour écran tactile ont une importance toute autre, selon que l'on cherche un nano-ordinateur ou une plateforme pour une solution de développement embarqué. Dans ce paradigme là, les rôles s'inversent et la très populaire Raspberry Pi perd vite son intérêt : pas de RTC, pas de NAND, pas assez de GPIO, pas de vrai Ethernet, pas de bootloader opensource, etc. Difficile alors de voir cette carte comme une plateforme d'évaluation pour le SoC BCM2835, chose qu'elle n'est de toute façon pas puisqu'il ne s'agit pas d'un devkit (« mais d'un jouet » diront certains).

La carte SAMA5D3 Xplained en revanche est clairement destinée à tester et évaluer le SAMA5D36 en offrant un maximum de facilités quant à la mise en oeuvre des différents périphériques. On s'étonne alors d'y trouver cet étrange connecteur Arduino dont le seul intérêt est de se voir équipé finalement d'un *shield* de prototypage qui offrira sans doute moins de souplesse qu'une simple double ligne de broches au pas de 2,54 mm comme on en trouve sur la BeagleBone Black par exemple. Force est de constater que nous n'avons pas trouvé de raison logique expliquant la présence de ce qu'on est tenté de qualifier de « standard de fait », mais issue d'un écosystème totalement différent.

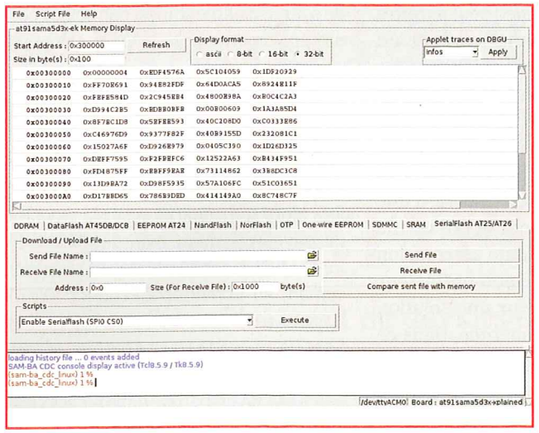
1 Mise en oeuvre rapide

Comme dit précédemment, nous avons à faire à une carte d'évaluation et dans cet esprit, le but est d'aller à l'essentiel le plus rapidement possible. Le connecteur JTAG présent sur la carte n'est ni la seule ni la première option permettant d'accéder à la NAND et au SoC. Le connecteur USB micro-AB faisant office de source d'alimentation par défaut permet également d'accéder à ces éléments.

La procédure est relativement simple puisqu'il suffit de retirer le cavalier en J2 puis de connecter la carte à un PC en USB. Une interface série CDC ACM (`/dev/ttyACM0` par exemple sous GNU/Linux) fera alors son apparition sur la machine de développement avec les VID/PID `03eb:6124` correspondants respectivement au idVendor/idProduct `Atmel Corp` et `at91sam SAMBA bootloader`. SAM-BA, pour *SAM Boot Assistant*, se compose d'un ensemble d'éléments : un moniteur SAM-BA embarqué dans la carte, une application disponible pour Windows et GNU/Linux et, liant les deux, un protocole utilisable sur une liaison USB ou au travers du port série DBGU (J23 sur la carte).

Le démarrage du moniteur SAM-BA dépend de la stratégie de démarrage du SoC, déterminée soit en raison d'un choix explicite de l'utilisateur, soit comme conséquence de tentatives infructueuses de trouver un support de démarrage adapté. Ainsi la carte est en mesure de démarrer aussi bien sur la NAND intégrée que sur la carte SD mais également sur une flash SPI ou TWI (i2c). Dans notre cas, la façon la plus simple d'obtenir un accès au moniteur SAM-BA est de retirer le cavalier en J2 connectant la ligne /CS de la NAND (/CE en réalité dans la doc Micron Tech). En l'absence d'accès à la NAND, la carte n'a d'autre choix que de proposer SAM-BA.

Il vous faudra ensuite vous rendre sur http://www.atmel.com/tools/ATMELSAM-BAIN-SYSTEMPROGRAMMER.aspx pour récupérer l'archive `sam-ba_2.13.zip` contenant les binaires 32 et 64 bits (compilés statiquement) de l'outil SAM-BA ainsi qu'un ensemble de scripts Tcl/Tk 8.4 (*oldschool* comme on aime). Afin que l'application puisse avoir accès au port CDC ACM, assurez-vous que l'utilisateur actif dispose des droits adéquates (utilisateur dans le groupe `dialout` avec Debian GNU/Linux par exemple). Pour lancer l'outil, placez-vous simplement dans le répertoire créé après désarchivage du Zip et utilisez `./sam-ba` ou `./sam-ba_64` selon votre architecture 32/64.



L'application vous demandera alors quel port utiliser et le type de carte à gérer (ici *at91sam5d3x-xplained*). L'interface qui se présente alors à vous est divisée en 6 parties distinctes :

- Une barre de menu sobre et classique,
- une interface de gestion de la mémoire,
- une série d'onglets permettant l'accès à différents types de supports/espaces,
- une interface de gestion lecture/écriture avec un support pour l'exécution de scripts,
- un shell Tcl,
- et une barre d'état résumant les informations de connexion à la carte.

Grâce à cette application il vous est possible d'accéder en lecture comme en écriture à l'ensemble des espaces mémoires intégrés ou optionnels (DDRAM, NAND, flash SPI, EEPROM, SD/MMC, etc). Il est important, à ce propos, de prendre pour habitude, juste après le boot, de systématiquement replacer le cavalier J2 et donc de permettre l'accès à la NAND. Ceci vous évitera de vous demander pourquoi l'accès à la mémoire ne fonctionne subitement plus (un bouton poussoir eu été préférable à un cavalier pour désactiver la ligne /CE de la NAND Micron Technology).

L'interface graphique offre des fonctionnalités très pratiques mais l'application est surtout entièrement scriptable en Tcl. Ainsi le site communautaire at91.com propose un ensemble de ressources et d'images binaires permettant de rapidement démarrer avec plus d'une demi douzaine de cartes d'évaluations utilisant des Atmel AT91SAM*.

Dans notre cas, la carte étant accompagnée d'un écran LCD 4,3 pouces TM4300B de marque PDA Inc, nous nous sommes plutôt dirigés vers le serveur FTP (ftp://www.at91.com/pub/demo/) afin d'y récupérer le fichier **linux4sam_4.4/linux-4sam-poky-qte-sama5d3_xplained-4.4.zip** contenant des images binaires incluant une application de démonstration Qt faisant usage du *touchscreen* de 480×272 pixels. L'archive ainsi récupérée créé un répertoire **linux4sam-poky-sama5d3_xplained-4.4/** au désarchivage, incluant entre autres choses :

- **sama5d3_xplained-nandflashboot-uboot-3.6.2.bin** : le bootloader de second niveau AT91Bootstrap qui initialise le matériel et charge U-Boot,
- **u-boot-sama5d3_xplained-v2013.07-at91-r2.bin** : le très connu bootloader U-Boot qui charge le noyau Linux,
- **ubootEnvtFileNandFlash.bin** : l'environnement U-Boot,
- **zImage-sama5d3_xplained.bin** : le noyau Linux,
- **atmel-xplained-demo-image-sama5d3_xplained.ubi** : le système de fichiers racine au format UBIFS,
- et trois fichiers DTB correspondants aux blobs du *device tree* respectivement pour la carte seule, avec un écran 4 pouces et 7 pouces de PDA Inc.

L'archive comprend également un script shell **demo_linux_nandflash.sh** contenant :

```
#!/bin/sh
sam-ba /dev/ttyACM0 at91sama5d3x-ek demo_linux_nandflash.tcl > logfile.log 2>&1
cat logfile.log
```

L'utilitaire **sam-ba**, attendu comme étant dans le **$PATH**, est appelé en ligne de commandes en spécifiant le port CDC ACM, le modèle de carte et un script Tcl. On remarquera la redirection vers un fichier de log qui est ensuite *dumpé* via **cat** (heu... et **tee** alors ?).

Plutôt que d'utiliser le script shell, on préférera recopier directement le SAM-BA dans le répertoire pour ensuite appelé l'outil directement, en précisant le bon script TCL, **demo_linux_nandflash_pda4.tcl** :

```
## Files to load
set bootstrapFile       "sama5d3_xplained-nandflashboot-uboot-3.6.2.bin"
set ubootFile           "u-boot-sama5d3_xplained-v2013.07-at91-r2.bin"
set kernelFile          "zImage-sama5d3_xplained.bin"
set rootfsFile          "atmel-xplained-demo-image-sama5d3_xplained.ubi"

## board variant
set boardFamily "at91-sama5d3_xplained"
set board_suffix "_pda4"

## now call common script
source demo_script_linux_nandflash.tcl
```

Il ne s'agit là que d'un ensemble de paramètres de configuration, le travail est effectué par **demo_script_linux_nandflash.tcl** dont la lecture s'avère très intéressante. En lançant ainsi la commande :

```
% ./sam-ba /dev/ttyACM0 at91sama5d3x-ek \
demo_linux_nandflash_pda4.tcl 2>&1 | \
tee my.log
```

on peut assister au fonctionnement complet du système :

- SAM-BA se connecte au port spécifié,
- s'assure de l'identité du SoC,
- charge **applet-lowlevelinit-sama5d3x.bin** pour initialiser le matériel,
- charge **applet-extram-sama5d3x.bin** pour accéder à la RAM externe,
- passe le relais à **demo_linux_nandflash_pda4.tcl**,
- qui charge **applet-nandflash-sama5d3x.bin** pour accéder à la NAND (effacement et écriture des données reçues),
- configure le PMECC (*Programmable Multi-bit ECC Controller*) intégré au SAMA5D36 qui sert d'interface pour l'accès à la NAND tout en fournissant un mécanisme de correction d'erreurs (ECC) aussi bien pour les NAND de type SLC (*Single-Level Cell*) comme la MT29F2G08ABAEA présente sur la carte et MLC (*Multi-level Cell*), plus coûteuses,
- efface la NAND,
- envoie et écrit l'image d'AT91Bootstrap à **0x0**,
- envoie et écrit U-Boot à **0x40000**,
- envoie et écrit l'environnement U-Boot à **0xC0000**,
- envoie et écrit le blob du Device Tree à **0x180000**,
- envoie et écrit le noyau Linux à **0x200000**,
- envoie et écrit le rootfs à **0x800000**,
- et finalement termine avec la mention **=== DONE. ===** sans faire de reset de la carte.

Les binaires des *applets* utilisés se trouvent dans le répertoire **tcl_lib/** de SAM-BA et majoritairement dans **tcl_lib/at91sama5d3x-ek**. Excellente surprise, les sources sont livrées dans **applets/sama5d3x/sam-ba_applets**. À condition de disposer d'un environnement de compilation *baremetal* (**arm-none-eabi-*** anciennement disponible via github.com/esden/summon-arm-toolchain mais maintenant bien davantage maintenu sur launchpad.net/gcc-arm-embedded), il sera très facile de recompiler ces éléments d'un simple **make**.

Note : voyez-vous, c'est précisément pour ce genre de choses très agréables, qu'il est préférable d'opter pour un vrai devkit offrant de vrais avantages aux développeurs (sources, datasheet, développement *mainline*, pérennité, etc), plutôt que de simplement raisonner sur la base de la simple formule Mhz/€.

Comme vous pouvez le constater, même en dehors du fait que le binaire **sam-ba** lui-même ne soit pas open source, un grand nombre d'éléments logiciels est à notre disposition et nous permet d'explorer relativement simplement la plateforme et ses qualités.

2 Compilation des éléments « vitaux »

La première chose qui vient à l'esprit lorsqu'on met la main sur une nouvelle carte, et ce après l'essai d'une ou plusieurs images binaires de démonstration, est le désir de reconstruire les éléments étape par étape en commençant de préférence au plus près du matériel. Nous ne nous attarderons pas ici sur le sujet puisque, comme vous allez le constater, tout ceci est relativement classique.

Dans le cas qui nous occupe, il s'agit donc de commencer par le bootloarder de second niveau, AT91Bootstrap, appelé par le code BootROM dans le SoC :

```
% git clone git://github.com/linux4sam/at91bootstrap.git
% cd at91bootstrap
% find board/ -type d | sort | cut -d '/' -f2
[...]
sama5d3xek
sama5d3_xplained
sama5d4ek
sama5d4_xplained

% find board/sama5d3_xplained/ -name "*_defconfig" | cut -d '/' -f3
sama5d3_xplainedsd_linux_uimage_dt_defconfig
sama5d3_xplainedsd_uboot_defconfig
sama5d3_xplainednf_linux_zimage_dt_defconfig
sama5d3_xplainednf_uboot_defconfig
sama5d3_xplainedsd_linux_zimage_dt_defconfig
sama5d3_xplainednf_linux_uimage_dt_defconfig

% make sama5d3_xplainednf_uboot_defconfig
#
# configuration written to .config
#
# make dependencies written to .auto.deps
[...]

% make CROSS_COMPILE=arm-linux-gnueabi-
[...]
  LD       sama5d3_xplained-nandflashboot-uboot-3.7.1.elf
Size of sama5d3_xplained-nandflashboot-uboot-3.7.1.bin is 20600 bytes
[Succeeded] It's OK to fit into SRAM area
[Attention] The space left for stack is 44936 bytes
```

Les sources sont récupérées directement sur le GitHub maintenu par le site communautaire at91.com et après quelques commandes permettant de déterminer les configurations disponibles adaptées à notre carte, il nous suffit d'invoquer **make** pour définir la configuration appropriée et procéder à la compilation. Celle-ci vise à produire une version du bootloader passant le relais à un U-Boot placé en NAND. Le résultat, sous la forme du fichier **sama5d3_xplained-nandflashboot-uboot-3.7.1.bin** pourra directement être réutilisé avec SAM-BA.

On passe ensuite à U-Boot avec une approche similaire :

```
% git clone https://github.com/linux4sam/u-boot-at91.git
% cd u-boot-at91

% grep  -e "xplained" boards.cfg | awk '{print $7}'
sama5d3_xplained_mmc
sama5d3_xplained_nandflash
sama5d4_xplained_mmc
sama5d4_xplained_nandflash
sama5d4_xplained_spiflash

% make sama5d3_xplained_nandflash_config
% make ARCH=arm  CROSS_COMPILE=arm-linux-gnueabi-
[...]
  LD      u-boot
  OBJCOPY u-boot.bin
  MKIMAGE u-boot.img
  OBJCOPY u-boot.srec
```

Là encore le résultat, **u-boot.bin**, est destiné à supporter une utilisation en NAND et on pourra, dans la foulé, produire une image de l'environnement U-Boot à partir d'un simple fichier texte que nous appellerons **uboot.txt** (oui, les \ en fin de ligne, ça marche avec **mkenvimage**) :

```
bootdelay=1
baudrate=115200
stdin=serial
stdout=serial
stderr=serial
bootargs=console=ttyS0,115200 \
  mtdparts=atmel_nand:\
  256k(bootstrap)ro,\
  512k(uboot)ro,\
  256k(env),\
  256k(env_redundant),\
  256k(spare),\
  512k(dtb),\
  6M(kernel)ro,-(rootfs) rootfstype=ubifs \
  ubi.mtd=7 root=ubi0:rootfs rw
bootcmd=nand read 0x21000000 0x00180000 0x00005271;\
  nand read 0x22000000 0x00200000 0x0033C7B0; \
  bootz 0x22000000 - 0x21000000
```

Notez cependant que ceci n'est utile que si vous souhaitez enregistrer à la main l'environnement en NAND. En effet, le script Tcl de SAM-BA produit à la volé un binaire correspondant à partir d'informations contenues dans **demo_script_linux_nandflash.tcl**. Dans le même ordre d'idées, les sources du *Device Tree* (DTS) incluent un **bootargs** (via **chosen**) pour le noyau, mais cet argument est écarté/écrasé au bénéfice des paramètres **bootargs** passés par U-Boot.

On construit ensuite les outils et on génère l'image **ubootEnvtFileNandFlash.bin** :

```
% make CROSS_COMPILE=arm-linux-gnueabi- tools

% tools/mkenvimage -r -p 0x00 -s 0x20000 \
-o ubootEnvtFileNandFlash.bin uboot.txt
```

Marquons là une pause pour ceux qui ont peut-être remarqué la syntaxe utilisée pour **bootcmd**. **nand read** prend comme toujours en argument l'adresse de chargement en RAM, l'adresse en NAND des données à charger et leurs tailles (qu'il faudra adapter en conséquence si l'on n'utilise pas le script Tcl). La particularité vient de **bootz** qui prend non pas un argument mais deux. Le premier est classiquement l'adresse du noyau et le second celui du bloc *Device Tree* utilisé par le noyau.

A propos du noyau, il est temps de le construire également :

```
% git clone git://github.com/linux4sam/linux-at91.git
% cd linux-at91
% make ARCH=arm  CROSS_COMPILE=arm-linux-gnueabi- sama5d3_xplained_
defconfig
% make ARCH=arm  CROSS_COMPILE=arm-linux-gnueabi- menuconfig
% make -j10 ARCH=arm  CROSS_COMPILE=arm-linux-gnueabi- zImage
% make -j10 ARCH=arm  CROSS_COMPILE=arm-linux-gnueabi- dtbs
```

Pas de grandes surprises ici en dehors du fait qu'on utilise les sources provenant des développeurs d'at91.com. Celles-ci sont, par définition, plus à jour que celles du noyau *mainline* même si les ajouts/corrections sont rapidement remontées *upstream*. Au terme de la compilation et de l'exécution de ces commandes nous retrouvons les binaires utiles dans **arch/arm/boot/** et **arch/arm/boot/dts**, respectivement **zImage** et **at91-sama5d3_xplained.dtb** (ou **at91-sama5d3_xplained_pda4.dtb** dans notre cas pour l'écran LCD 4 pouces).

Il nous suffit alors de regrouper l'ensemble des fichiers afin de pouvoir dupliquer/personnaliser le script SAM-BA :

```
at91bootstrap/binaries/sama5d3_xplained-nandflashboot-uboot-3.7.1.bin
u-boot-at91/u-boot.bin
linux-at91/arch/arm/boot/zImage
linux-at91/arch/arm/boot/dts/at91-sama5d3_xplained.dtb
linux-at91/arch/arm/boot/dts/at91-sama5d3_xplained_pda4.dtb
```

Ceci ne forme que la base d'un système puisque nous n'avons absolument pas abordé l'aspect *userland*. Il est, bien entendu, possible de construire un rootfs « artisanalement » (façon LFS) mais ceci implique plus de problèmes que d'avantages :

- Construction de chaque élément à la main,
- composition intégrale et manuelle du système d'un fichier racine cohérent,
- gestion des dépendances quasi-inexistante,
- pour des projets très complets, l'utilisation de scripts ou de **Makefile** devient rapidement ingérable.

Il existe de longue date quelques systèmes de construction de système embarqué fort respectables et efficaces parmi lesquels OpenWRT, PTXdist, OpenEmbedded ou encore Buildroot. Ce dernier à fait l'objet d'un long article dans Open Silicium 10, suivi d'un complément dans le numéro 11. Nous n'allons donc pas revenir sur le sujet (du moins dans l'immédiat) mais plutôt nous pencher sur OpenEmbedded et plus exactement le système en vogue, Poky, du projet Yocto, reposant sur OpenEmbedded. Nous arrêtons donc là le présent article pour abordez cet important sujet dans le suivant... ■

UTILISATION DE YOCTO/POKY AVEC LA SAMA5D3 XPLAINED

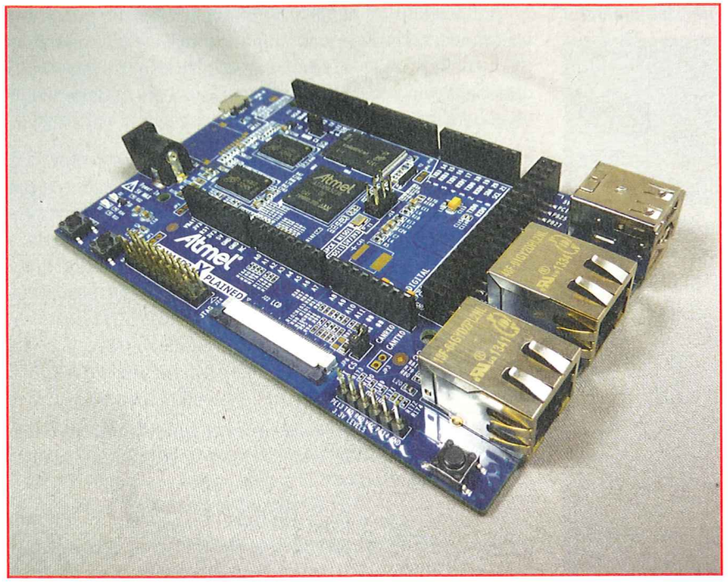
par Denis BODOR

La construction d'un système complet était, fût un temps, l'affaire de quelques téléchargements, scripts et Makefiles accompagnés d'une bonne maîtrise de l'architecture d'un système GNU/Linux. Alors que ceci est sans doute encore vrai pour certaines réalisations, comme un firmware de netcam chinoise basé sur uClinux, force est de constater que la complexité et la richesse grandissante des plateformes et des systèmes rend cela presque impossible pour une majorité de projets. Pour orchestrer la construction de système embarqué, on utilise alors des ensembles que l'on peut qualifier d'environnements de construction. Poky du projet Yocto est l'un des environnements qui tend actuellement à croître rapidement en popularité.

Construire un système cohérent complet ne nécessite pas forcément un système de construction. C'est tout simplement un choix raisonnable basé sur le fait qu'on souhaitera ou non maintenir le système et le faire évoluer. Derrière cette remarque sarcastique se cache une réalité évidente. Si vous comptez créer un produit unique, bas de gamme et sans autre but que de rapidement « avoir quelque chose qui marche à peu près », peut-être que la construction artisanale et manuelle de son firmware peut passablement être envisagé comme une option. En revanche, dès lors que vous comptez maintenir le produit, en assurer l'évolution et la qualité, vous vous retrouvez devant deux options : développer un système de construction ou en choisir un existant. Dans les deux cas le but est de disposer d'un outil adapté à votre projet et surtout à son avenir.

Majoritairement et historiquement, deux principaux systèmes communautaires se partagent le « marché ». Nous avons d'une part Buildroot, qui est simple d'utilisation, très facile à appréhender et parfaitement adapté au développement d'un firmware pour un système unique. Buildroot sera un excellent choix pour une *appliance* avec un cahier des charges fixe et invariable. Il atteindra cependant rapidement ses limites si votre projet est massif, complexe et susceptible de concerner, par exemple, une gamme de produits aux fonctionnalités et caractéristiques disparates. Un exemple hypothétique peut être une gamme de liseuses électroniques avec des modèles proposant différentes ressources pour l'utilisateur final : plusieurs tailles d'écrans *e-paper*, des déclinaisons LCD, avec ou sans 3G ou Wifi, avec ou sans technologies *contactless*, et pourquoi pas, tout simplement, plusieurs SoC de marques et de modèles différents.

Dans ce genre de situations, si vous souhaitez optimiser vos développements en factorisant un maximum d'éléments, la solution proposée par Buildroot revient à maintenir plusieurs projets parallèles en fonction des modèles. Le principe de base d'un système de construction voulant que le développement des composants se fasse en dehors de l'environnement ne règle qu'une partie du problème. Vous êtes toujours obligés de gérer les spécificités propres à chaque plateforme dans plusieurs « branches » de développement. Il en va de même s'il s'agit d'avoir une certaine modularité dans le système final. Buildroot peut techniquement supporter un système de *packaging* pour offrir une composition *in vivo* du système mais il n'est pas conçu initialement dans ce but. Buildroot construit des rootfs et non des distributions GNU/Linux.



L'autre système offrant des fonctionnalités similaires à Buildroot est OpenEmbedded (ou OE pour les intimes). Existant de longue date, OE gagne depuis quelques temps en popularité, en particulier suite à l'annonce par la Linux Foundation du projet Yocto. L'implémentation de référence du projet Yocto, nommée Poky, utilise l'architecture OE et plus exactement l'OpenEmbedded-Core (ou OE-Core) issue d'un effort collaboratif avec le projet Yocto. Ok, à ce stade, cela commence à ressembler à une sitcom anglaise où on ne sait plus qui flirt avec qui... Retenez simplement quelques points importants :

- Le projet Yocto est le groupe de travail de la Linux Foundation qui développe Poky,
- OE-core (OpenEmbedded-Core) est le *framework* développé par OpenEmbedded et un élément de Poky,
- Poky est l'implémentation de référence que tout le monde (ou presque) utilise comme base (Poky inclus OE-core),
- Angstrom est une autre « distribution » également basée sur OE-core mais ne faisant pas partie du projet Yocto.

Le projet Yocto attire l'attention des industriels depuis quelques temps en raison de la souplesse de son architecture et son adaptabilité. Le support de la part de la *Linux Foundation* est également un élément décisif. Les choix techniques, que nous allons découvrir, ont été fait depuis le début du projet pour assurer un juste équilibre entre structure et personnalisation. Chaque intervenant dans la conception d'un système embarqué peut aisément ajouter le support de son matériel, des déclinaisons spécifiques, les éléments logiciels, des applicatifs, etc. Le système final se construit brique par brique et celles-ci peuvent être, si elles sont judicieusement conçues et utilisées, interchangeables. Basculer une suite logicielle d'une plateforme à une autre demande ainsi bien moins de travail qu'avec un autre système de construction. Bien entendu, nous le verrons plus loin dans l'article, ces « facilités » ont bien entendu un prix et les utilisateurs de longue date d'OpenEmbedded le savent : la construction d'un système prend un temps et des ressources non négligeables (sensiblement plus qu'avec Buildroot). Et, disons-le franchement, bien que Yocto/Poky soit moins « usine à gaz » qu'OpenEmbedded ne l'était fût un temps, il reste tout de même plus difficile à appréhender dans son ensemble que Buildroot.

1 Architecture et organisation de Yocto & Poky

Par définition, le projet Yocto est un ensemble de *templates*, d'outils et de procédures permettant de créer une distribution pour système embarqué personnalisée basée sur GNU/Linux. Le système de référence, Poky, qui repose sur OE-core, est un ensemble de *layers* qu'il faut voir comme des profils plutôt que comme de réelles couches. Nous conserverons ici le terme *layer* utilisé dans la documentation officiel et nous ferons de même pour la majorité des autres éléments même si certains d'entre eux sont plus facilement traduisibles.

Les layers sont des ensembles de *recipes* (recettes) décrivant des tâches à effectuer comme le téléchargement, la configuration, la compilation, l'installation et/ou la création de paquets ou d'images. OE-core fourni un ensemble de layers commun à tous les systèmes basés et construits sur OpenEmbedded. Poky ajoute une collection d'outils permettant d'initier (ou initialiser) une distribution embarquée avec des composants élémentaires indispensables.

Pour interpréter et « exécuter » les configurations et les recipes qui forment les layers, le système utilise un moteur de construction appelé BitBake écrit en Python. Son principe de fonctionnement est assez similaire à celui de GNU Make et est en mesure de tirer avantage des architectures PC actuelles via une parallélisation des tâches. Cependant, un certain nombre de spécificités sont propres à BitBake :

- Les tâches sont exécutées en fonction de métadonnées et non de simples *Makefiles*. Les métadonnées prennent la forme de recipes, de configurations et de classes et

ont pour objectif d'indiquer à BitBake non seulement la manière dont il doit exécuter la tâche mais également quelles sont les dépendances entre les tâches.

- BitBake intègre une bibliothèque de téléchargement permettant la récupération des sources des éléments depuis plusieurs types de services dont FTP, HTTP(S), Git, SVN, etc.

- BitBake peut fonctionner en mode client/serveur et donc, en plus d'être utilisé localement en ligne de commandes, peut être « piloté » via XMLRPC. Ceci rend possible l'utilisation d'interfaces de gestion, y compris en mode graphique (Hob utilisant GTK+/PyGTK) ou via une interface Web (API Toaster).

Nous baserons nos explications sur Yocto/Poky 11.0.1, alias Daisy 1.6.1, annoncé en version stable en mars dernier. Une version plus récente, 12.0.0, alias Dizzy 1.7 disponible depuis fin octobre est également proposée au téléchargement mais les layers utilisés pour notre carte servant de base d'expérimentation (SAMA5D3-Xplained) n'ont pas encore été validés à cette date. Les différences entre les deux versions tiennent en particulier en des mises à jours de logiciels (GCC, Glibc, etc) et en l'application de correctifs de sécurité (Bash, OpenSSH, FFmpeg, Python, etc), mais l'architecture et la philosophie restent totalement identiques en ce qui concerne l'objet de cet article.

La démarche d'initialisation d'une distribution repose en premier lieu sur la récupération de Poky. On complétera ensuite l'installation avec des layers spécifiques à la plateforme qui nous occupe afin d'obtenir un BSP pour *Board Support Package*, ainsi que ceux utilisés pour l'aspect applicatif du projet. Tous ces layers seront utilisés de concert pour produire le système final. Ceci implique une règle à respecter quoi qu'il arrive : IL N'EST PAS QUESTION DE MODIFIER POKY ! JAMAIS !

Pour récupérer la base du système, deux options s'offrent à vous. La première consiste à utiliser Git et à cloner le dépôt officiel en spécifiant la branche qui vous intéresse. Ici :

```
% git clone -b daisy git://git.yoctoproject.org/poky.git poky
Clonage dans 'poky'...
remote: Counting objects: 250941, done.
[...]
Vérification de la connectivité... fait.
Checking out files: 100% (4737/4737), done.
```

Afin d'éviter toutes manipulations hasardeuses et regrettables, il est fortement recommandé d'immédiatement créer et basculer sur une branche locale particulière :

```
% cd poky
% git checkout daisy -b my_branch
Basculement sur la nouvelle branche 'my_branch'

% git branch
  daisy
* my_branch
```

L'autre solution consiste à récupérer sur https://www.yoctoproject.org/downloads une archive d'une version stable : **poky-daisy-11.0.1.tar.bz2**. Après désarchivage, vous obtiendrez un répertoire **poky-daisy-11.0.1** dans lequel vous pourrez travailler.

De manière générale, les développements se font EN DEHORS du système de construction et sont intégrés par l'intermédiaire de dépôts séparés, utilisés par les recipes qui composent un layer. Bien entendu, ces layers sont ajoutés dans le système de construction mais ne font pas partie de Poky.

Après récupération par Git ou sous forme d'archive, vous obtenez la structure suivante :

- **bitbake/** : contient l'ensemble des scripts qui forment BitBake et la commande **bitbake** elle-même,
- **documentation/** : les sources de l'indispensable documentation de référence incluant les différents guides et manuels,
- **meta/** : c'est le layer OE-Core. Ce répertoire contient donc les recipes spécifiques à cette partie du système.
- **meta-selftest/** : un layer de test généralement utilisé pour le développement de Yocto/Poky lui-même et normalement pas intégré à vos projets.
- **scripts/** : regroupe les scripts pour configurer l'environnement, les outils de développement et utilitaires pour déployer les images (flash).
- **meta-skeleton/** : regroupe un ensemble de recipes de démonstration pour l'intégration de ses propres éléments (générique, noyau, bibliothèque, etc).
- **meta-yocto/** : contient la configuration pour le système de référence (Poky).
- **meta-yocto-bsp/** : la configuration pour les BSP des plateformes matérielles (ou émulées) de référence du projet Yocto,
- **oe-init-build-env** : le script de configuration/initialisation de l'environnement de construction. Celui-ci n'est pas destiné à être exécuté mais doit être « sourcé » avec **source oe-init-build-env** ou **. oe-init-build-env**. Son travail, entre autres choses, consiste à initialiser un certain nombre de variables d'environnements et vous placer dans le répertoire de construction désigné en argument ou **build** par défaut.
- **oe-init-build-env-memres** : une version spécifique de l'environnement installant un serveur BitBake résident en mémoire.

Vous l'avez compris, **oe-init-build-env** va modifier l'environnement de votre shell. Lors du premier « appel » si le répertoire de construction n'existe pas il sera créé et vous y serez placé. Vous pouvez spécifier le nom de répertoire qui vous plaira en évitant toutefois les espaces dans le nom

utilisé (par définition, les espaces dans les noms de répertoires et de fichiers sont le mal absolu, quand bien même les environnements graphiques poussent les utilisateurs à en faire usage, ces environnements sont une composante du mal. Le bien c'est la ligne de commandes, tout le monde le sait).

Le nom **build** est utilisé par défaut mais il est recommandé de spécifier quelque chose de plus explicite. En effet, l'une des caractéristiques très avantageuses de Yocto/Poky est de permettre la construction de plusieurs systèmes sur la base d'un seul et même environnement de construction. Vous l'avez sans doute également compris, la racine du système de construction restera inchangée, l'ensemble des opérations, des téléchargements, des compilations et des éléments propres à la construction du système résident dans le répertoire de construction (**build**). Il en va de même pour la configuration du système que vous souhaitez construire. Le choix des layers à utiliser et les spécificités (architecture, type de compilation, choix du système de gestion de packages, etc) n'existent que dans un répertoire de construction initialisé avec **oe-init-build-env**.

Bien entendu, entre deux sessions de travail, il vous suffira de réitérer la commande **. oe-init-build-env repertoire** pour retrouver les éléments tels qu'ils auront été laissés précédemment. Le système de construction lui-même gère son état. Ainsi, il est parfaitement possible d'interrompre une construction, d'éteindre l'ordinateur pour ensuite revenir à cette activité. Le déclenchement d'une nouvelle tentative de construction ne concernera que les tâches encore en instance et il ne sera pas nécessaire de repartir de zéro. Il en va de même en cas d'échec de la construction en cas de défaut ponctuel (manque de mémoire disponible, d'espace disque, etc).

2 Un peu de pratique

Nous nous sommes maintenant suffisamment penché sur les parties théories et il est temps de passer à quelque chose de plus concret. Nous allons donc explorer Yocto/Poky en construisant un système complet pour notre Atmel SAMA5D3-Xplained. Nous partirons du principe que vous avez d'ors et déjà téléchargé (Git ou archive) Poky dans sa version Daisy et vous êtes placé dans le répertoire en question.

La première chose à faire avant d'entamer les réjouissances est de compléter l'installation. A ce stade nous n'avons que la base du système de référence (Poky) et n'avons aucun support pour le SoC Atmel. De base, le BSP proposé par Poky ne propose que les cibles (*target*) suivantes :

```
% ls meta-yocto-bsp/conf/machine/*.conf
beaglebone.conf
edgerouter.conf
genericx86-64.conf
genericx86.conf
mpc8315e-rdb.conf
```

auxquelles s'ajoutent celles fournies par OE-Core :

```
% ls meta/conf/machine/*.conf
qemuarm.conf
qemumips64.conf
qemumips.conf
qemuppc.conf
qemux86-64.conf
qemux86.conf
```

Dans l'état, nous pouvons déjà construire un système basique pour l'une de ces cibles en une paire de commandes (ne le faites pas, c'est juste pour illustrer) :

```
% . oe-init-build-env
[...]
### Shell environment set up for builds. ###

You can now run 'bitbake <target>'

Common targets are:
    core-image-minimal
    core-image-sato
    meta-toolchain
    adt-installer
    meta-ide-support

You can also run generated qemu images
with a command like 'runqemu qemux86'

% bitbake core-image-minimal
```

La sortie de cette commande hypothétique nous permet de décrire un peu davantage le fonctionnement de Yocto/Poky. En effet, la première étape de la construction consistera à analyser les recipes :

```
Parsing recipes: 100% |##################| Time: 00:00:18
Parsing of 862 .bb files complete (0 cached, 862 parsed).
1221 targets, 38 skipped, 0 masked, 0 errors.
NOTE: Resolving any missing task queue dependencies
```

Dans la configuration actuelle, qui correspond à Poky au sens le plus primaire du terme et à la production de l'image **core-image-minimal**, nous avons 862 recipes qui définissent 1221 cibles (éléments logiciels à créer). Il est important à ce stade de signaler que la terminologie est quelque chose de très important avec Yocto. « recipe » par exemple peut désigner la création d'un paquet... ou pas. Inversement, une recipe peut également créer plusieurs paquets... ou pas. Il est donc important de ne pas désigner les « recipes » par le terme « package ». Ici nous voyons apparaître le nom « targets ». Ceci désigne littéralement une cible sur laquelle porte une action, le plus souvent la cible est le nom d'une recipe et l'action est sa construction (**do_build**). Si nous ajoutons à cela d'autres éléments perturbants comme le fait que les répertoires contenant les layers soient préfixés de **meta-** alors que les metadonnées (*metadata*) sont des informations qui prennent la forme de « variables » stockées dans des recipes (**.bb**, **.conf**, **.inc**, **.bbclass**), on peut en arriver à vraiment regretter la simplicité et la logique de Buildroot.

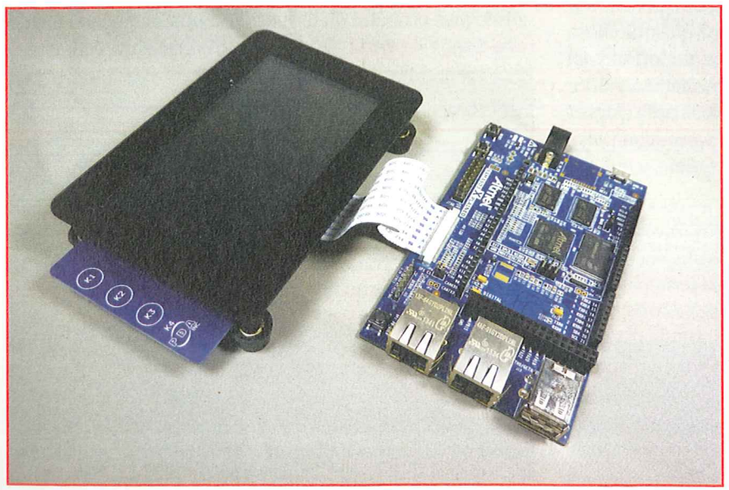
Quoi qu'il en soit, peu après cette analyse, le système va vous afficher un résumé de la configuration qui sera utilisée :

```
Build Configuration:
BB_VERSION        = "1.22.0"
BUILD_SYS         = "i686-linux"
NATIVELSBSTRING   = "Debian-7.0"
TARGET_SYS        = "i586-poky-linux"
MACHINE           = "qemux86"
DISTRO            = "poky"
DISTRO_VERSION    = "1.6.2"
TUNE_FEATURES     = "m32 i586"
TARGET_FPU        = ""
meta
meta-yocto
meta-yocto-bsp    = "my_branch:[...]
```

On reconnaît ici différents numéros de versions et informations concernant le système hôte ainsi que l'architecture cible. Nous n'avons touché à aucun fichier de configuration et la cible par défaut sera une machine désignée par **qemux86**, une émulation x86 par Qemu. Apparaissent également les layers utilisés.

Suite à cet affichage, les choses sérieuses commencent avec la préparation de la *runqueue* ou en d'autres termes, la liste des tâches que le système de construction devra accomplir pour arriver au résultat final. *runqueue* qui sera ensuite méthodiquement dépilée, tâche par tâche :

```
NOTE: Preparing runqueue
NOTE: Executing SetScene Tasks
NOTE: Executing RunQueue Tasks
Currently 16 running tasks (31 of 1833):
0: libtool-native-2.4.2-r6.1 do_fetch (pid 15127)
1: m4-native-1.4.17-r0 do_fetch (pid 15124)
[...]
13: binutils-cross-2.24-r0 do_fetch (pid 15307)
14: libmpc-native-1.0.2-r0 do_fetch (pid 15428)
15: quilt-native-0.61-r0 do_install (pid 16005)
```



Nous avons ici 1833 tâches à accomplir. Celles-ci vont du téléchargement des sources (**do_fetch**), à la génération d'un paquet (**do_package_write_rpm** par exemple) en passant par la compilation (**do_compile**), l'installation (**do_install**) ou encore la copie des fichiers obtenus de manière à ce que les autres recipes puissent y avoir accès (**do_populate_sysroot**). La tâche **do_build** est celle par défaut de chaque recipe et dépend d'une succession d'autres tâches nécessaires à la construction effective.

Ce système d'interdépendances peut sembler un peu complexe et plus difficile à gérer qu'un simple ensemble de *Makefiles* mais permet d'obtenir un niveau d'abstraction intéressant. En effet, d'un point de vue utilisateur/développeur, la construction de la recipe **core-image-minimal** n'est pas différente de celle d'une recipe décrivant un autre élément logiciel comme, par exemple, **ethtool**. Le même principe s'applique, avec la même logique que vous utilisiez **bitbake core-image-minimal** ou **bitbake ethtool**, seule la chaîne de dépendance et donc la *runqueue* nécessaire sera différente.

Si vous souhaitez avoir une vision relativement exacte du coût en complexité de ce genre de facilités qui vous sont « offertes », vous pouvez utiliser l'option **-g** sous la forme **bitbake core-image-minimal -g** et obtenir ainsi trois diagrammes Graphwiz représentant les interdépendances. Utilisés avec la commande **dot task-depends.dot -Tpdf > graph.pdf** par exemple, vous pourrez alors littéralement voir la complexité en oeuvre et en déduire la raison pour laquelle deux CPU 4 cœurs avec 6 Go de mémoire donnent l'impression de réellement avoir du mal à produire une image basique de système embarqué. Encore une fois, pour que les choses soient tout à fait claires, pour un système simple avec quelques composants, Buildroot reste le choix le plus adapté mais dès lors qu'il s'agit d'une vaste collection de composants, doublée d'un besoin de personnalisation avancé, Yocto/Poky sera un meilleur choix. Ceci non en terme de performance lors de la construction mais de souplesse au moment de la configuration et de l'association des éléments.

3 Beaucoup plus de pratique

Arrêtons de tourner autour du pot et retroussons nos manches (ou plutôt celles du PC). Nous avons une structure standard pour construire le système mais ceci n'est pas suffisant. Nous allons à présent ajouter les layers nécessaires à notre plateforme d'expérimentation. Plaçons-nous à la racine de l'arborescence et intégrons en premier lieu le layer OpenEmbedded :


```
% git clone -b daisy  git://git.openembedded.org/meta-openembedded
[...]
Résolution des deltas: 100% (25660/25660), done.
Vérification de la connectivité... fait.
```

Ceci vient en complément d'OE-Core et ajoute un ensemble de recipes permettant de construire une vaste collections d'outils, de bibliothèques et d'applicatifs. Ces recipes sont organisées en layers (**meta-openembedded/meta***), fournissant les recipes « en lots ». Nous retrouvons ainsi, par exemple, les EFL (bibliothèques Enlightenment), des éléments pour les serveurs web (Apache, PHP, NGinx, XDebug), ou encore des composants multimédias (VLC, Sox, XBMC, UPNP).

Pour construire une image de démonstration pour la plateforme SAMA5D3 Xplained, nous devons également disposer du Toolkit Qt5 sur lequel repose l'interface graphique présentant les différents programmes exemples faisant usage de l'écran LCD tactile de PDA Inc. Ajoutez le layer approprié est un jeu d'enfant :

```
% git clone git://github.com/meta-qt5/meta-qt5.git
[...]
Résolution des deltas: 100% (2164/2164), done.
Vérification de la connectivité... fait.
```

Nous avons à présent l'ensemble des layers contenant les recipes qui forment les briques nécessaires pour toute la partie *user* du système mais nous n'avons pas de BSP pour notre plateforme. Atmel au travers du site communautaire at91.com a réalisé ce travail et il nous suffit, là encore, d'intégrer simplement un nouveau layer :

```
% git clone http://github.com/linux4sam/meta-atmel
[...]
Résolution des deltas: 100% (719/719), done.
Vérification de la connectivité... fait.
```

Ce qui nous donne au final :

```
% ls -F
bitbake/
documentation/
LICENSE
meta/
meta-atmel/
meta-openembedded/
meta-qt5/
meta-selftest/
meta-skeleton/
meta-yocto/
meta-yocto-bsp/
oe-init-build-env*
oe-init-build-env-memres*
README
README.hardware
scripts/
```

L'étape suivante consiste à initialiser le répertoire de construction. Comme nous aimons suivre nos propres recommandations, nous le désignerons sous le nom **build-SAMA5D3x**, ce qui nous donne la commande **. oe-init-build-env build-SAMA5D3x**, et nous nous retrouvons dans **poky/build-SAMA5D3x** avec un ensemble de variables d'environnement définies et/ou modifiées pour nous (vous pouvez avoir un aperçu de l'environnement en utilisant **bitbake -e**, attention il y a plus de 17000 lignes).

Notre répertoire de construction se compose ainsi :

```
% tree
.
ÐÐÐ conf
    ÐÐÐ bblayers.conf
    ÐÐÐ local.conf
    ÐÐÐ templateconf.cfg
```

Comme vous pouvez le voir, tout ceci est relativement vide. Nous avons là un environnement encore non personnalisé et la première étape consiste à spécifier les layers que nous comptons utiliser. Éditez alors le fichier **conf/bblayers.conf** contenant actuellement :

```
BBLAYERS ?= " \
  /home/denis/EMB/POKY/poky/meta \
  /home/denis/EMB/POKY/poky/meta-yocto \
  /home/denis/EMB/POKY/poky/meta-yocto-bsp \
  "
```

en

```
BBLAYERS ?= " \
  /home/denis/EMB/POKY/poky/meta-atmel \
  /home/denis/EMB/POKY/poky/meta-qt5 \
  /home/denis/EMB/POKY/poky/meta \
  /home/denis/EMB/POKY/poky/meta-yocto \
  /home/denis/EMB/POKY/poky/meta-yocto-bsp \
  /home/denis/EMB/POKY/poky/meta-openembedded/meta-oe \
  /home/denis/EMB/POKY/poky/meta-openembedded/meta-networking \
  /home/denis/EMB/POKY/poky/meta-openembedded/meta-ruby \
  "
```

Bien entendu, vous devez adapter les chemins absolus (ici **/home/denis/EMB/POKY**) en fonction de votre propre installation de manière à renseigner précisément les emplacements des layers. Ceci aura pour effet de rendre disponible les recipes provenant de ces layers pour votre *build*. N'hésitez pas à aller fouiller dans ces répertoires pour vous imprégner de l'architecture et des relations entre layers et recipes.

A ce stade, vous avez accès à bien plus de cibles ***image*** et à celles par défaut fournies par le layer **meta/** (OE-Core), **core-image-minimal**, **core-image-sato**, **meta-toolchain**, **adt-installer** et **meta-ide-support**, s'ajoutent celles proposées par **meta-atmel/** :

```
% ls ../meta-atmel/recipes-core/images/*.bb
atmel-demo-image.bb
atmel-xplained-demo-image.bb
atmel-xplained-lcd-demo-image.bb
```

Ces fichiers **.bb** décrivent des recipes, comme toutes les autres. Celles-ci sont généralement organisées de manière

logique et nous avons ici celles concernant la production d'images du système final. Là encore, le parcours de ces répertoires et fichiers est très intéressant pour comprendre la philosophie de l'ensemble (j'insiste mais la meilleure source de documentation est incontestablement la lecture des fichiers **.bb** et **.bbclass**, dixit le *Yocto Project Development Manual*). Ainsi, la recipe **atmel-xplained-lcd-demo-image.bb** contient :

```
DESCRIPTION = "An image for SAMA5D3 Xplained with screen."
LICENSE = "MIT"
PR = "r1"

require atmel-demo-image.inc

IMAGE_FEATURES += "splash"

IMAGE_INSTALL += "\
    packagegroup-base-3g \
    packagegroup-base-usbhost \
    fb-test \
    tslib \
    tslib-conf \
    tslib-tests \
    tslib-calibrate \
    "
```

Afin de simplifier l'écriture, un mécanisme d'inclusion permet d'appeler un contenu commun à plusieurs recipes sous la forme d'un fichier **.inc**. Ici, **atmel-demo-image.inc** est donc intégré à notre recipe et ajoute les recipes nécessaires :

```
IMAGE_INSTALL = "\
    packagegroup-core-boot \
    packagegroup-core-basic \
    packagegroup-base-wifi \
    packagegroup-base-bluetooth \
    packagegroup-base-usbgadget \
    kernel-modules \
    lrzsz \
    setserial \
    opkg \
    iperf \
    linux-firmware \
    i2c-tools \
    dosfstools \
    mtd-utils \
    iproute2 \
    iptables \
    bridge-utils \
    canutils \
    python-pyserial \
    python-smbus \
    python-ctypes \
    python-pip \
    python-setuptools \
    python-pycurl \
    gdbserver \
    usbutils \
    wget \
    ${CORE_IMAGE_EXTRA_INSTALL} \
    "

inherit core-image
[...]
```

Remarquez la dernière ligne nous permettant de renseigner **IMAGE_INSTALL** avec la liste des composants que nous souhaitons intégrer dans le système final tout en héritant de ceux provenant de **core-image**. Ce fichier **.inc** est assez typique de la construction d'une recipe pour une image personnalisée. C'est une base de système composée des éléments décrits plus **core-image**, qui est une recipe de groupe de paquets (voyez cela comme une pseudo-recipe). Dans la recipe qui **require** ce fichier, nous complétons la liste des éléments avec **IMAGE_INSTALL +=** pour ajouter d'autres éléments (notez la différence **=** vs **+=**).

Nous pouvons également regarder du côté de **meta-atmel/recipes-qt/images** pour découvrir deux recipes permettant d'obtenir les images de démonstration Qt4e et Qt5.

Tout ceci est bien sympathique, mais le simple ajout de layers ne fait qu'ajouter des recipes, nous n'avons à ce stade choisi aucune plateforme. Si présentement nous tentons d'exécuter la recipe **atmel-xplained-lcd-demo-image** par exemple, nous obtiendrons une erreur. Pire encore, en utilisant **bitbake core-image-minimal** nous produirons une image pour x86 qui n'a rien à voir avec nos chers nouveaux layers.

Pour corriger cela, il faut nous pencher sur la configuration locale définissant les paramètres de construction. Ceci est entièrement configuré dans le fichier **conf/local.conf**. Mais avant de procéder à des modifications dans ce fichier, il est CAPITAL de comprendre les syntaxes utilisées et en particulier en ce qui concerne les assignations :

- **VAR = "pouet"** : initialise **VAR** avec la chaîne « pouet »,
- **VAR ?= "pouet"** : fait de même mais uniquement si **VAR** c'est pas encore défini,
- **VAR ??= "pouet"** : idem mais de plus faible « priorité » car évalué en dernier. Même si vous utilisez cette syntaxe AVANT un **VAR ?= "toto"**, **VAR** ne contiendra PAS la chaîne « pouet ». On parle de *lazy/weak assignment*.
- **VAR += "pouet"** : ajout au bout de **VAR** avec un espace (*append*),
- **VAR =+ "pouet"** : ajout à l'avant de **VAR** avec un espace (*prepend*),
- **VAR .= "pouet"** : ajout au bout de **VAR** sans espace (*append*),
- **VAR_append = "pouet"** : idem,
- **VAR =. "pouet"** : ajout à l'avant de **VAR** sans espace (*prepend*),
- **VAR = "${AUTRE_VAR} blabla"** : interprétation de **AUTRE_VAR** lors d'une référence à **VAR**,
- **VAR := "${AUTRE_VAR} blabla"** : interprétation de **AUTRE_VAR** à l'assignation (*immediate variable expansion*).

Comme bien d'autres choses avec Yocto/Poky, ceci est à bien se graver dans le cerveau car à l'issue des lignes suivantes :

```
VAR ??= "titi"
VAR ?= "toto"
VAR ?= "tata"
```

VAR contiendra **toto** et non **titi** car l'assignation avec **??=** est faible et sera testée/effectuée en dernier. Dans le cas du fichier de configuration **local.conf** par exemple, nous avons :

```
[...]
MACHINE ??= "qemux86"
[...]
```

MACHINE permet de spécifier le nom de la plateforme visée. Ceci se présente sous la forme de fichiers **.conf** (métadonnées de configuration) dans **meta-atmel/conf/machine** :

```
% ls ../meta-atmel/conf/machine
at91sam9m10g45ek.conf
at91sam9rlek.conf
at91sam9x5ek.conf
sama5d3xek.conf
sama5d3-xplained.conf
sama5d4ek.conf
sama5d4-xplained.conf
```

Ainsi,

```
MACHINE ??= "sama5d3-xplained"
MACHINE ??= "qemux86"
```

construira une image pour **qemux86** (ou générera une erreur dans le cas de **atmel-xplained-lcd-demo-image**). Mais,

```
MACHINE ??= "qemux86"
MACHINE ?= "sama5d3-xplained"
```

construira pour **sama5d3-xplained**, tout comme,

```
MACHINE ??= "qemux86"
MACHINE = "sama5d3-xplained"
```

ou encore,

```
MACHINE ??= "qemux86"
MACHINE ??= "sama5d3-xplained"
```

Ceci peut être TRÈS problématique car, en n'étant pas suffisamment attentif, il est aisé de démarrer un processus très long pour finalement obtenir un résultat inutilisable. Prenez l'habitude de toujours utiliser l'option **-n** de **bitbake** afin de simuler (*dry run*) la construction et vérifier le résumé qui vous est affiché, avant de procéder à l'opération effective.

MACHINE nous permet donc de spécifier la plateforme visée mais ce n'est pas le seul élément que nous pouvons configurer. Yocto est un projet de générateur de distribution incluant une gestion de package « à l'exécution » (comme une distribution pour PC). Cependant, le choix du système de gestion de paquets est laissé à la discrétion du développeur. Ceci se fait en définissant **PACKAGE_CLASSES** :

- **package_rpm** : type RPM (Fedora/RedHat),
- **package_ipk** : type OPKG (OpenWrt par exemple),
- **package_deb** : type Debian/Ubuntu.

La documentation du layer **meta-atmel** sur GitHub recommande l'utilisation de **package_ipk**.

Nous devons également nous pencher sur deux variables TRÈS IMPORTANTES :

- **BB_NUMBER_THREADS** : détermine le nombre de tâche que BitBake va lancer en parallèle (tâches aux sens Yocto du terme, il ne s'agit pas nécessairement de compilation). Par défaut dans **local.conf**, cette variable est initialisée avec **${@oe.utils.cpu_count()}** qui correspond au nombre de cœurs détectés sur le système hôte (**/proc/cpuinfo**)
- **PARALLEL_MAKE** : correspond à l'option **-j** de **make** permettant de spécifier le nombre de processus que GNU Make peut faire fonctionner en parallèle lors d'une tâche de compilation. Là encore c'est le nombre de cœurs présents qui défini la valeur par défaut utilisée.

Si vous pensez qu'un bi-Xeon 5520 accompagné de 6 Go de mémoire (ma machine de travail) est un peu *overkill* pour construire une distribution relativement simple comme celle de démonstration d'Atmel utilisant Qt et l'écran LCD, détrompez-vous. 16 tâches BitBake avec potentiellement 16 compilations parallèles fonctionnent à merveille... jusqu'au moment où il s'agit de compiler Qt (**qtwebkit_5.3.2.bb** pour être précis). Pour peu que vous ayez un navigateur avec quelques onglets ouverts et pourquoi pas un LibreOffice qui traîne et vous comprendrez votre malheur, et l'intérêt de ces deux variables :

```
| virtual memory exhausted: Cannot allocate memory
| virtual memory exhausted: Cannot allocate memory
| make[2]: *** [.obj/svg/SVGAllInOne.o] Error 1
| make[2]: *** Waiting for unfinished jobs....
[...]
| ERROR: oe_runmake failed
[...]
Summary: 1 task failed:
  /home/denis/EMB/POKY/poky/meta-qt5/
  recipes-qt/qt5/qtwebkit_5.3.2.bb, do_compile
[...]
```

Ceci après 9961 secondes (2h46) d'exécution et donc sous forme de surprise matinale si vous avez eu la mauvaise (ou bonne) idée de lancer la construction avant d'aller vous coucher ou en quittant le bureau. Il n'y a malheureusement pas de solution clé en main et bien que la situation était bien pire avec les versions précédentes de BitBake, ceci reste un problème important. En fonction des éléments à construire, de

votre (ou vos) CPU, de la mémoire disponible et d'éventuelles autres applications en fonction, vous rencontrerez potentiellement quelques problèmes de consommation de ressources et devrez alors revoir ces deux variables à la baisse. Ceci est fort heureusement compensé par le fait que la construction se fait de manière incrémentale et que si une première tentative échoue, il vous suffit d'éditer **local.conf** et réitérer la commande.

En ce qui concerne la plateforme SAMA5d3 Xplained et la démonstration Qt basée sur le *framebuffer* Linux, il sera également nécessaire d'ajouter deux lignes supplémentaires au **local.conf** :

```
LICENSE_FLAGS_WHITELIST += "commercial"
SYSVINIT_ENABLED_GETTYS = ""
```

Ceci afin de prendre en compte la double licence de Qt ainsi que le fait que la démonstration utilise le périphérique *framebuffer* qui n'est donc normalement pas utilisé pour une console (**fbcon**). Il n'y a en principe pas de terminal virtuel et donc pas de **getty** lancé au démarrage.

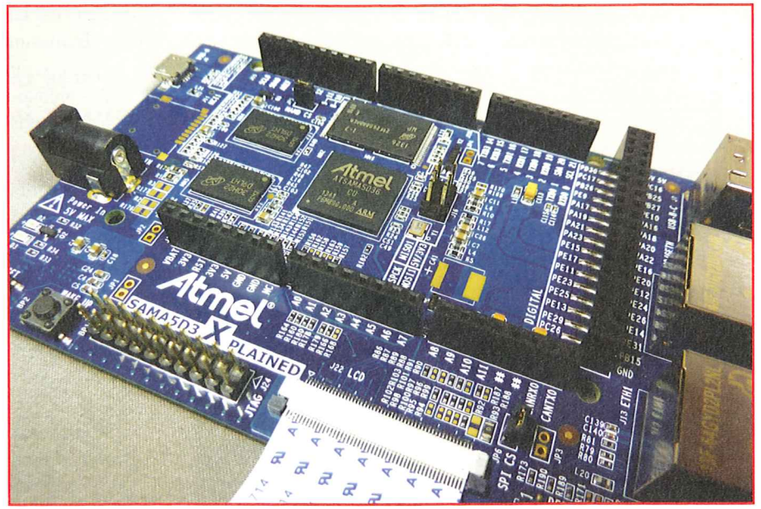
Après ces modifications, vous pourrez construire soit les images basiques proposées par Poky (comme **core-image-minimal**) ou celles proposées par le layer **meta-atmel** avec :

```
% bitbake atmel-xplained-lcd-demo-image
```

ou

```
% bitbake atmel-qt4e-demo-image
```

Selon votre configuration, vous pouvez partir boire quelques cafés, manger quelques pizza ou même faire un marathon StarWars ou LoTR avec des amis. Ce après quoi, normalement, si tout s'est bien déroulé, vous aurez le plaisir de constater que le sous-répertoire **tmp/** du répertoire de construction (**build-SAMA5D3x** ici) se retrouve généreusement peuplé. Le répertoire de construction lui-même est également passé d'une poignée de kilooctets et une



paire de fichiers à quelques 38 Go au terme de la construction de **atmel-qt4e-demo-image** (oui, vous devrez peut-être supprimer **~/pr0n** ou investir quelques 50€ pour 1 To supplémentaire) !

Peu intuitivement (encore !) ce que vous chercherez inévitablement à ce moment se trouve dans **tmp/** et plus particulièrement **tmp/deploy/images/sama5d3-xplained** :

```
% ls -F tmp/deploy/images/sama5d3-xplained
at91bootstrap.bin@
at91bootstrap-sama5d3_xplained.bin@
atmel-qt4e-demo-image-sama5d3-xplained-20141119104551.rootfs.manifest
atmel-qt4e-demo-image-sama5d3-xplained-20141119104551.rootfs.tar.gz
atmel-qt4e-demo-image-sama5d3-xplained-20141119104551.rootfs.ubi
atmel-qt4e-demo-image-sama5d3-xplained-20141119104551.rootfs.ubifs
atmel-qt4e-demo-image-sama5d3-xplained.manifest@
atmel-qt4e-demo-image-sama5d3-xplained.tar.gz@
atmel-qt4e-demo-image-sama5d3-xplained.ubi@
BOOT.BIN@
modules--3.10+0+35158dd80a-r5-sama5d3-xplained-20141119102702.tgz
modules-sama5d3-xplained.tgz@
README_-_DO_NOT_DELETE_FILES_IN_THIS_DIRECTORY.txt
sama5d3_xplained-nandflashboot-uboot-3.7.1.bin*
u-boot.bin@
u-boot-sama5d3-xplained.bin@
u-boot-sama5d3-xplained-v2014.07-at91-r1.bin*
zImage@
zImage--3.10+0+35158dd80a-r5-at91-sama5d3_xplained-20141119102702.dtb
zImage--3.10+0+35158dd80a-r5-at91-sama5d3_xplained_pda4-20141119102702.dtb
zImage--3.10+0+35158dd80a-r5-at91-sama5d3_xplained_pda7-20141119102702.dtb
zImage--3.10+0+35158dd80a-r5-sama5d3-xplained-20141119102702.bin
zImage-at91-sama5d3_xplained.dtb@
zImage-at91-sama5d3_xplained_pda4.dtb@
zImage-at91-sama5d3_xplained_pda7.dtb@
zImage-sama5d3-xplained.bin@
```

On y retrouve le bootloader AT91Bootstrap, U-Boot, le noyau Linux, les DTB ou encore le rootfs UBIFS, exactement comme livrés par les images de démonstration utilisables avec SAM-BA (cf article précédent). Mais **tmp/** contient également :

- **tmp/buildstats/** : un très grand nombre d'informations concernant le processus de construction aussi bien pour l'ensemble de l'opération que pour chaque élément. Ce répertoire contient une arborescence composée du nom de la cible et de la *MACHINE* puis de sous-répertoires horodatés. Ceci permet de retrouver les informations sur une construction en plusieurs fois (vous savez, plus de RAM, plus d'espace libre, etc). Chaque recipe est décomposée en tâches (**do_compile**, **do_configure**, **do_install**, **do_patch**, etc) et chacune d'elles possède un fichier éponyme dans le répertoire avec les informations statistiques correspondantes. On peut ainsi apprendre que la fameuse compilation de **qtwebkit-5.3.2-r0** à elle seule aura durée 1624,06 secondes (27 minutes) et l'installation 15,02 secondes.
- **tmp/work/** : c'est, au sens large du terme, le répertoire de travail de BitBake. C'est là que sont décompressés, configurés et construits les éléments logiciels. On y retrouve donc, en toute logique les fichiers objets et les sources, patchés si besoin.
- **tmp/sysroots/** : se retrouve ici les éléments comme les fichiers d'entête et les bibliothèques partagées nécessaires pour la compilation et la construction des éléments, aussi bien à destination de la plateforme cible que de la machine hôte de développement.

Chose très intéressante, comme l'ensemble du système de construction repose sur un principe incrémental, le fait d'avoir construit **atmel-qt4e-demo-image** et d'avoir réussi cette opération vous donne de l'avance pour la construction d'une autre cible. Ainsi, en vous pliant d'un **bitbake atmel-xplained-lcd-demo-image** qui correspond à une version légère de l'image système ne faisant que mettre à disposition **fb-test** et quelques autres outils, seule une partie des tâches doit être exécutée. Une masse énorme d'opérations de compilations par exemple ne sont plus nécessaires et directement sautées pour passer à l'installation et la création de paquets. Le résultat est une construction en quelques 241,37 secondes (4 minutes) d'une nouvelle image **atmel-xplained-lcd-demo-image-sama5d3-xplained-20141119164553.rootfs.ubi** de quelques 114 Mo contenant une poignée de paquets en moins et/ou en plus.

En parlant de paquets justement. Comme Yocto/Poky est un constructeur de distribution, celle-ci s'accompagne d'un dépôt contenant les paquets installables sur la plateforme cible. Celui-ci est automatiquement créé et, dans le cas présent comme nous avons choisi le format IPK, nous retrouvons tout cela dans **tmp/deploy/ipk/**. Notez que ce répertoire se décline en sous-répertoires en fonction de la plateforme concernée. On retrouve ainsi :

- **all/** pour les applications génériques comme les firmwares de périphériques, les certificats des AC SSL/TLS, ou les simples données (type **tzdata**),
- **cortexa5t2hf-vfp/** contenant les paquets propres à l'architecture, ici un Cortex A5 avec VFP, comme la libC, les Toolkits, les outils systèmes et tout ce qui est lié au jeu d'instructions spécifique à un type de CPU,
- **sama5d3_xplained/** regroupant les paquets propres à la plateforme elle-même comme les modules noyaux, le bootloader, etc.

Là encore, l'aspect *multi-build* et/ou multi-plateforme est très présent. Le contenu de **all/** par exemple est uniquement dépendant de la distribution et non de l'architecture. De même les paquets dans **cortexa5t2hf-vfp/** seront adaptés pour toutes les architectures de ce type et non uniquement un SoC Atmel. Tout ceci n'aura donc pas besoin d'être reconstruit pour une autre plateforme ARM Cortex A5 similaire, pas plus que **all/** changera en passant d'ARM à MIPS. Voilà précisément le type de problèmes et de besoins adressés par le projet Yocto.

Conclusion

Que tirer finalement de tout cela ? Personnellement, je reste profondément attaché à Buildroot et son caractère proche de la philosophie qu'on peut retrouver dans la logique de construction/compilation du noyau Linux, et surtout sa simplicité (j'ai beaucoup joué avec OpenWrt par le passé). L'approche adoptée par le projet Yocto, faisant intervenir des outils Python et des concepts qu'on pourrait qualifier de plus haut niveau d'abstraction peut effectivement être bénéfique selon le type de système à obtenir. Certains des avantages généralement mis en avant, comme le fait de naturellement pouvoir supporter plusieurs plateformes cibles sont, à mon goût, quelque peu surévalués. Cela n'est, en effet, pas sans rappeler les discussions quant à la transition du système d'init de la plupart des distributions GNU/Linux vers Systemd. Là aussi, l'avantage trop souvent mis en avant concerne le temps de démarrage du système qui pourtant n'a pas vraiment d'intérêt puisqu'on parle en réalité d'un gain de 0,1 % sur une journée de travail de 12h (on ne passe normalement pas son temps à redémarrer la machine, en tout cas sous GNU/Linux).

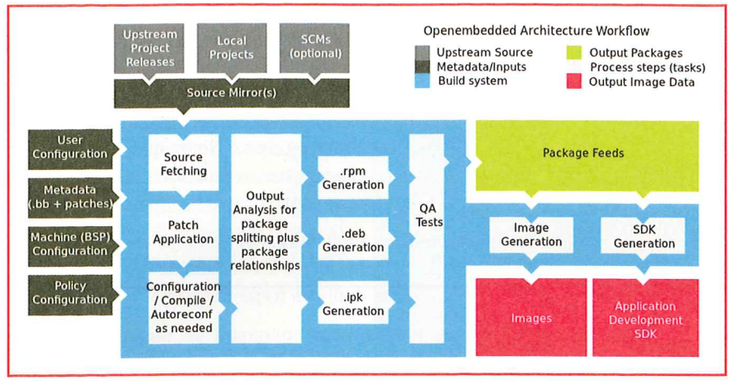


Schéma générale de l'architecture et du workflow OpenEmbedded/Yocto

L'argument concernant la construction pour plusieurs plateformes cibles doit donc être pondérer, de préférence au bénéfice de l'aspect modulaire du projet Yocto. Voilà un point qui séduit le monde industriel et pousse les constructeurs à faire la promotion de cette initiative. Il en va de même pour des éléments qui, pour un développeur GNU/Linux expérimenté, n'ont strictement aucun sens. Je pense par exemple au plugin pour l'IDE Eclipse destiné à supporter *Application Development Toolkit* (ADT) et donc permettre le développement, l'exécution et le test d'applications directement dans Eclipse. Développer une application pour système embarqué dans un IDE Java et donc un environnement graphique gourmand en ressources ne semble pas cohérent face à une démarche classique reposant sur un bon éditeur et CMake/autotools/Make par exemple. Cependant, ceci permet de fédérer des développeurs applicatifs n'ayant pas nécessairement de connaissances concernant les systèmes de développement issus du monde UNIX/Linux. L'histoire nous a montrer que certains choix contre-intuitif, comme celui de développer des applications pour smartphone en Java, peuvent conduire au succès d'une architecture logicielle.

Le réel bénéfice du projet Yocto réside davantage dans l'approche humaine que technique : le fait de fournir à l'industrie l'outil qu'elle désir, dans lequel elle peut facilement intégrer sa brique tout en attirant des développeurs de tous horizons (y compris ceux souhaitant développer dans Eclipse).

Et le réel problème du projet Yocto est la difficulté immédiate à assimiler la logique globale de l'ensemble malgré des points furieusement trompeur : les layers nommés **meta**, les *metadata* décrivant des recipes, des groupes de paquets qui regroupent en fait des recipes qu'il vaut mieux éviter d'appeler *package*, et tout un tas d'autres choses déroutantes comme (**atmel-xplained-lcd-demo-image.bb**) :

```
IMAGE_FEATURES += "splash"
```

alors que dans **meta/classes/image.bbclass** (pas **core-image.bbclass**) :

```
# Define some very basic feature package groups
SPLASH ?= "psplash"
FEATURE_PACKAGES_splash = "${SPLASH}"
```

Si ce n'est pas un apeau à typos ça...

Je comprends parfaitement l'intérêt, puisqu'on peut ainsi écraser **SPLASH** dans la recipe de l'image à produire afin d'utiliser autre chose que **psplash** mais lorsqu'il s'agit de retrouver un élément par l'exploration, cela peut littéralement se transformer en chasse au trésor tant les possibilités de personnalisation sont importantes et diversifiées ! Et cela commence par la simple question « je bitbake quoi pour obtenir l'image que je veux? » (et non, je ne trouve pas que 15s de `bitbake -s | grep image` soit une réponse adaptée, pas plus que `rgrep "^IMAGE_FEATURES" meta*`). La simple lecture de la page « Yocto Project Introduction » de elinux.org tend à montrer qu'appréhender Yocto est tout sauf une tâche facile et intuitive...

Le projet Yocto se décrit lui-même comme un générateur de distributions et ceci inclus une fonctionnalité de gestion de paquets embarquée. Buildroot est un générateur de rootfs ou de firmware complet. Vos besoins, et le marché, dictent le choix du système de génération. La préférence « sentimentale » envers l'un ou l'autre est secondaire, il faut simplement utiliser le bon outil pour la bonne tâche.

Enfin, il apparaît clairement qu'on entre en Yocto comme on entre en religion. Il faut envisager cela comme un tout et généraliser son utilisation en une seule et même « instance ». Le coût d'investissement en temps et en ressource « intellectuelle » est trop important pour utiliser Yocto de manière ponctuelle et/ou sous la forme d'une série d'installations dans des répertoires distincts. Non, la logique consiste à disposer d'une installation de Yocto et d'intégrer progressivement chaque nouveau projet, d'ajouter des layers et de multiplier éventuellement les répertoires de construction. Ainsi, si vous vous mettez à utiliser Yocto pour votre activité et non simplement pour un ou deux projets, là l'investissement vaut vraiment la peine.

En conclusion, qu'on ait ou non des affinités avec Yocto ou Buildroot, le point important à relever est en premier lieu le fait qu'on dispose de deux systèmes de construction ouverts, open source, fonctionnels, collaboratifs, et activement maintenus. En d'autres termes, réjouissons-nous déjà que le choix nous soit accessible... ■

UN SED ÇA VA, TROIS SED...

par Yann GUIDON – des bits, des électrons, des photons et aussi de l'aspirine...

C'est un des piliers historiques du monde UNIX, un des outils de base que l'administrateur et le développeur « doivent connaître », mais aussi un de ces programmes qui donnent à UNIX/Linux sa mauvaise réputation, son aura d'impénétrabilité, qui font penser que le système ne sera jamais vraiment, totalement accessible au commun des mortels.
Pire que vi, plus sournois que brainfuck, voilà le retour de sed.

Pourtant j'avais été échaudé lorsque j'ai conçu le compacteur de code JavaScript **[1]**. J'avais dû abandonner rapidement car les limitations sont très vite apparues et le code source devenait ignoble, donc ingérable sur le long terme. Mais j'ai bien vite tourné la page et j'ai cru qu'un algorithme simple échapperait à la règle. Au final, j'ai perdu beaucoup de temps et ma foi en l'humanité.

1 Un problème trivial

À la base, c'était quand même très simple. J'avais déjà codé l'algorithme en C et en Javascript, c'était très facile, digne d'un exercice de programmation de débutant, réfléchi, écrit et testé en trois minutes.

L'objectif consiste à encoder un flux binaire en hexadécimal. Chaque octet est converti en deux caractères, et lorsqu'un octet se répète, les répétitions suivantes sont représentées par un seul point. Par exemple la chaine hexadécimale `00 11 22 22 22 33 33 44 55 55 55 55 66` est écrite ainsi : `00 11 22 . . 33 . 44 55 . . . 66` (on peut même enlever les espaces mais là n'est pas la question).

C'est la gestion des répétitions qui est si facile à coder avec un langage normal, et si alambiquée à coder en **sed**... L'algorithme normal fonctionne ainsi : une variable conserve la valeur précédemment envoyée. Cette variable est comparée avec la valeur lue suivante. Si les deux valeurs sont égales, alors on écrit un point en sortie, sinon on remplace la variable avec la nouvelle valeur, qu'on écrit aussi.

La variable temporaire est initialisée à une valeur impossible au début, afin d'éviter que la comparaison soit vraie dès le départ, mais à part ce détail, il n'y a aucun piège. Cela prend au plus quelques lignes de code. Par exemple en pseudo-C :

```
int tmp = -1;

int c = (int)lit_valeur();
if (c == tmp)
  ecrit('.');
else {
  tmp = c;
  ecrit(c);
}
```

Dans certains cas j'ai besoin de réaliser ceci sans compiler de programme, pour gagner de la place et du temps. Il existe d'ailleurs un programme qui fait presque ce que je veux : **od** est un tout petit utilitaire standard de « *dump octal* », un peu comme **hexdump**. On peut ajuster le format de sortie, lui fournir un fichier binaire en entrée, tout baigne.

```
od -v -w16 -t x1 -An nom_du_fichier
```

- **-w16** indique qu'on veut traiter 16 octets par ligne.
- **-t x1** indique que les données sont affichées par octets hexadécimaux.
- **-An** demande de ne pas afficher les numéros de lignes.

Par exemple :

```
$ od -v -w16 -t x1 -An /dev/urandom
...
 5e 73 11 f2 3c cc 15 c6 33 ec 59 55 a7 a5 a2 f9
 bb 2b b2 1a 6f 5d 74 42 42 d1 db 9d a0 3f 44 87
 65 08 dc 33 47 93 e1 2f f1 62 57 b6 f0 c2 42 43
...
```

C'est parfait jusque là. Il reste juste un petit détail : l'option **-v** demande à **od** de ne pas indiquer les lignes dupliquées avec un astérisque. Si on l'enlève, on obtient ceci :

```
$ od  -w16 -t x1 -An /dev/zero
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
*
^C
```


En résumé, on peut dire que **od** convient pour la conversion de binaire à hexadécimal, mais la gestion des répétitions ne convient pas, car il n'y a aucune indication du nombre de répétitions. Qu'à cela ne tienne, on peut tuyauter sa sortie dans un autre programme qui va s'en charger comme on le souhaite.

2 Sed c'est dien...

On prend donc la sortie de **od**, dont la ligne de commandes est un petit peu modifiée (pour écrire un seul octet par ligne) car l'idée est de traiter les octets les uns après les autres, afin de gérer des lignes consécutives.

```
$ od -v -w1 -t x1 -An /dev/urandom
 bd
 3a
 1b
 08
 fd^C
```

Maintenant, il doit bien y avoir un moyen de comparer deux lignes, non ?

sed semble être la solution indiquée car il fonctionne sur la base de lignes lues successivement et traitées par flux. Il dispose de deux espaces de stockage : l'espace des motifs, sur lequel s'effectuent tous les traitements, et un espace auxiliaire (« *hold buffer* »).

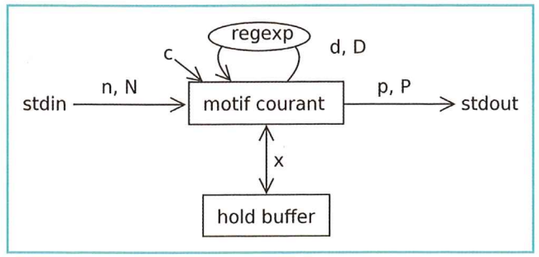


Fig. 1: La structure interne de sed

Dans un sens, cela évoque une machine de Turing, ou un ordinateur doté d'un seul registre accumulateur et d'un registre temporaire. La différence étant qu'au lieu de traiter des nombres, l'unité de traitement est une ligne (une suite de caractères terminés par **\n**). Nos grands-pères ont marché sur la Lune avec moins que ça donc je devrais m'en sortir haut la main.

C'est Internet qui m'a donné l'espoir d'y arriver rapidement et facilement, car j'ai trouvé un petit script **[2]** qui semble proche de la fonction que je veux réaliser :

```
# élimine les lignes consécutives identiques d'un fichier (émulation "uniq").
# La première ligne d'un ensemble de lignes identiques consécutives
# est retenue, les autres éliminées
sed '$!N; /^\(.*\)\n\1$/!P; D'
```

C'est court, rapide et ça marche. C'est ce qui fait qu'on utilise sed, on ne comprend pas cette série de caractères abscons mais les incantations magiques sont bien pratiques quand on les connait. « *Il ne reste plus qu'à* » modifier ce *one-liner* pour qu'il écrive un point au lieu d'effacer la ligne. Facile.

Spoiler alert : j'ai failli abandonner après y avoir perdu plus d'une nuit... Toutes les modifications triviales et fonctionnelles du script aboutissent à un de ces deux résultats :

- Soit le point est correctement écrit mais des répétitions multiples sont transformées en répétitions de paires valeur-point. La transformation fait échouer la comparaison avec la ligne qui suit, donc on ne peut pas obtenir de séquences de points consécutifs.

```
AA          AA
AA  donne   .
AA          AA
AA          .
```

- Soit les séquences de points sont correctement écrites mais la valeur initiale est écrite à la fin :

```
AA          .
AA  donne   .
AA          .
AA          AA
```

Au secours.

3 Les entrailles de la bête

Après avoir (un peu mais quand même trop) tourné en rond, j'ai réalisé qu'il n'était pas possible d'y arriver sans comprendre mieux ce que je faisais (au lieu de changer des trucs pour voir ce que ça faisait) et j'ai donc creusé dans les documentations disponibles. Une page web **[3]** contient heureusement des informations synthétiques, pratiques et utiles, même si j'ai dû beaucoup interpréter pour dénicher ce qui était pertinent pour mon cas. Jean-François Champollion aurait été fier de moi !

Il faut se rendre à l'évidence : **sed** n'a pas été conçu pour être un langage de programmation, c'est juste un utilitaire qui a mal tourné en cédant au *feature-creep*. Tout d'abord, une première révélation : les deux « espaces » de travail internes ne peuvent pas être comparés. Il n'y a pas d'instruction pour cela. Alors comment le *one-liner* fonctionne-t-il (et pourquoi marche-t-il) ? Voici donc l'autopsie :

- **$!N** doit être compris comme la séquence **$! N** (avec un espace pour nous, les humains). Mais comme **sed** ignore les espaces à certains endroits (mais pas d'autres !) cela permet d'écrire comme des vrais *ha><orz*.
- Le tout début d'une commande peut commencer par une adresse, et dans ce cas, **$** indique la fin du fichier. À ne pas confondre avec l'*adresse* **/$/** qui, à cause des barres obliques, est une expression régulière.

- Le point d'exclamation **!** indique que l'adresse (ou, devrait-on dire, « *condition d'exécution des commandes qui suivent* » mais ce serait trop clair) est inversée.

- **N** concatène la ligne suivante dans l'espace de travail.

Ainsi, la séquence **$!N** ordonne d'ajouter la ligne suivante à la ligne en cours de traitement, si la fin du fichier n'a pas été atteinte. En fait, **N** fonctionne aussi bien si on ne précise pas la fin, mais c'est un indice bien utile pour plus tard. Si vous avez compris jusque là, accrochez-vous, la suite est plus corsée.

- **/^\(.*\)\n\1$/!** est une *adresse* qui indique que la commande suivante **P** ne sera pas exécutée si l'expression régulière est trouvée dans l'espace de travail.
- **P** est une instruction d'impression (*Print*) ou d'écriture sur la sortie. Il est bon de savoir que la même instruction en minuscule **p** écrit tout l'espace de travail, alors que **P** n'écrit que jusqu'au premier retour à la ligne (**\n**).
- L'impression est conditionnée par une expression régulière (entre les barres obliques), qui couvre tout l'espace de travail (encadré par **^** pour le début et **$** pour la fin de la ligne courante).
- La première partie de l'expression régulière **(.*)\n** indique « *tous les caractères jusqu'à un retour à la ligne* », et les parenthèses gardent ces caractères pour la suite.
- La deuxième partie **\1** fait justement référence à cette première suite de caractères.

En résumé, c'est la manière seddienne de dire que deux lignes consécutives sont identiques. Dans le cas contraire (**!**) on imprime la partie gauche de l'espace de travail.

La ligne se termine par **D**, la commande « *Delete* », qui efface la première ligne de l'espace de travail. L'effet de bord est que cela termine aussi la séquence de commandes, on ne peut rien mettre derrière et le code reprend au début de la séquence, pour la ligne suivante. Ça devait sembler logique aux concepteurs, même pratique, mais quand on sait qu'il n'y a quasiment pas d'autres commandes qui traite partiellement l'espace de travail, on se demande s'ils sont masochistes ou juste pervers.

4 Après l'analyse, la synthèse.

Ainsi, en **sed**, la plupart du travail effectif est réalisé par les expressions régulières, ultrapuissantes, assistées par une poignée de commandes qui bougent des morceaux de ligne ici et là. L'espace auxiliaire n'est même pas touché pour déterminer si deux lignes consécutives sont identiques.

De plus, une des nombreuses manières de coder un script « transparent » (qui ne fait rien) est la suivante :

```
sed 'N;P;D;'
```

En résumé : lire la ligne suivante, l'imprimer et l'effacer.

En filigrane, l'espace de travail contient toujours la ligne suivante, prête à être traitée, mais pas encore affichée. C'est à partir de cette base que je commence à écrire la suite.

On connait aussi l'*adresse* qui correspond à une ligne dupliquée : **/^\(.*\)\n\1$/** et on peut la transformer en expression de substitution. L'idée étant de remplacer un des deux éléments par un point :

```
s/^\(.*\)\n\1$/\1\n./
```

ou bien

```
s/^\(.*\)\n\1$/.\n\1/
```

Lorsqu'elle est placée juste avant l'instruction P, cette substitution génère un des deux comportements déjà expliqués : soit les points successifs *précèdent* la valeur, soit un seul point est écrit.

Las, j'ai tenté de contourner le problème. Il faut que le point soit après la première valeur, donc je le déplace après celle-ci, avec une expression régulière (il n'y a pas d'autre moyen). Ce doit être effectué sur la ligne du début, car sinon on ne pourra pas comparer les lignes suivantes ensuite. L'ensemble du script de conversion commence à ressembler à cela :

```
od -v -w1 -t x1 -An $fichier |sed '
N
s/^\(.*\)\n\1$/\1.\n\1/
P
D'
```

Les commandes sont sur des lignes individuelles mais rien ne vous oblige à le coder ainsi, si vous voulez vous entrainer pour l'IOCCC. En tout cas, ce script est déjà un début, les octets dupliqués sont marqués par le point voulu, bien que ce soit sur la ligne précédente...

```
AA          AA.
AA  donne   AA.
AA          AA.
AA          AA
```

Mais ce contournement n'a fait que déplacer le problème, sans le résoudre : il faut un autre script **sed** pour transformer les lignes ainsi modifiées. Comme on effectue encore du traitement multiligne, on repart de la fameuse séquence neutre **'N;P;D;'**.

J'ai bien tenté de scripter une opération qui élimine un nombre quand la ligne précédente finit par un point, mais les effets de bord étaient trop importants, j'ai donc *encore* divisé le problème en deux :

- Un premier script transforme une ligne terminée par un point, en deux lignes, la première sans le point, la deuxième avec le point. Une autre façon de déplacer le problème.


```
sed 's/[.]$/\n./g'
AA.          AA
AA.  donne   .
AA.          AA
AA           .
             AA
             .
             AA
```

- Le script suivant s'inspire du script initial, qui élimine des lignes si une condition est remplie. Dans ce cas, quand on détecte un point, on élimine la ligne suivante, ce qui se fait sans substitution :

```
sed '/[.]/ { N ; c .
}'

AA          AA
.           .
AA  donne
.           .
AA
.           .
AA
```

Ici, la ligne suivante est lue avec N mais l'ensemble est purement remplacé par un point, avec l'instruction c (change). Celle-ci a la manie de prendre tout le reste de la ligne, ce qui oblige à des acrobaties syntaxiques qui ajoutent au charme si désuet de **sed**...

Une fois que nous disposons d'une série de lignes dont les répétitions sont correctement marquées, ce n'est plus qu'une formalité d'arranger les informations et **sed** est ici tout à fait adapté. D'abord, on réunit autant de lignes que possible dans l'espace de travail, en répétant l'instruction N. Nous avons vu précédemment que l'adresse $! permettait de filtrer cette instruction, ce qui évite des erreurs de formatage en fin de flux. Ensuite, on efface tous les retours à la ligne et les espaces, et c'est fini.

```
sed '
$!N;$!N;$!N;$!N;$!N;$!N;$!N;
$!N;$!N;$!N;$!N;$!N;$!N;$!N;
s/[ \n]//g'
```

La séquence de scripts fonctionne mais le résultat est loin d'être stellaire car on se retrouve avec quatre **sed** tuyautés les uns aux autres. Ce code horrible est beaucoup plus lent que le script initial sans la détection des octets identiques, ce qui réduit beaucoup l'intérêt de tous ces efforts !

5 Je vous en rajoute encore un peu ?

La nuit porte conseil... quand elle n'est pas blanche et quand on dort. Je m'étais juré d'en rester là mais **sed** a ce charme particulier des casse-têtes : on n'est pas satisfait tant qu'on n'a pas trouvé **la** solution.

J'ai mentionné au début qu'un autre espace temporaire existait dans **sed**, et on peut basculer de l'un à l'autre avec l'instruction **x**. Or il ne m'était pas venu à l'idée jusqu'à maintenant d'y mémoriser les séquences de points. Une fois l'idée en tête, il n'est plus possible de résister, je dois réparer l'affront. L'algorithme est alors le suivant :

- Si les deux lignes sont identiques, alors ajouter un point à l'espace auxiliaire et terminer.
- Si les lignes sont différentes, alors imprimer l'espace auxiliaire puis l'espace de travail.

Mais il n'y a pas d'instruction **if** avec **sed**. Il faut ruser avec les blocs (marqués par des accolades) et des instructions comme **d** ou **D** qui terminent le bloc et recommencent l'exécution depuis le début.

La première partie est codée avec l'adresse-expression-régulière qu'on connait déjà, et exécute un bloc qui échange les espaces, ajoute le point et ré-échange les espaces, pour finir avec le **D** qui permet de reboucler.

```
sed '$!N;
/^\(.*\)\n\1$/ {
  x         # échange
  s/$/\n./  # ajoute le point à l'espace auxiliaire
  x         # ré-échange
  D         # efface le début de l'espace de travail
}
```

Remarque

La substitution à base d'expressions régulières pose un problème à long terme car on risque de tomber sur une exécution en $\Omega(n^2)$ mais comment le savoir si on ne connait pas parfaitement les entrailles du logiciel ? Cela n'est pas gênant avec quelques points dans l'espace auxiliaire mais quand la chaîne grandira, tous les points précédemment ajoutés risquent d'être re-balayés, donc il faut éviter de traiter des fichiers avec des milliers de répétitions...

La deuxième partie (une sorte de **else**) est exécutée à la suite du bloc **x;s;x;D** puisque cela ne correspond pas à l'adresse voulue. Les instructions suivantes effectuent ceci :

```
P       # impression du début de la ligne
x       # échange des espaces
p       # imprime l'espace (les points)
s/.*//  # efface l'espace
x       # échange des espaces
D       # efface et reboucle
```

On y est ! Ou presque, puisque rien n'est jamais vraiment gagné avec **sed**. En effet, un autre effet de bord apparaît : l'instruction **p** ajoute un saut à la ligne même quand la ligne est vide, ce qui ajoute une ligne vide entre deux lignes différentes.

Tant qu'on y est, on pourrait tuyauter un autre **sed** pour effacer les lignes vides avec un autre *one-liner*.

```
sed '/^$/ d'
```

Mais plus on tuyaute, plus l'ensemble est lent ! Enfin, normalement, non ?

Et **sed** réserve encore d'autres surprises : on peut ajouter des adresses à l'intérieur d'un bloc (on ne peut plus vraiment parler d'adresse alors, mais admettons), ce qui permet une exécution conditionnelle de l'impression.

D'autres retours à la ligne sont ajoutés inconditionnellement lorsqu'on ajoute un point au buffer avec `\n.`, on peut enlever celui du début avec une simple substitution. Ce qui donne le script suivant :

```
sed '$!N;
/^\(.*\)\n\1$/ {
  x
  s/$/\n./
  x
  D
}
P
x
/^$/! {    # ne pas imprimer si l'espace auxiliaire est vide
  s/^\n//  # enlever le premier retour à la ligne
  p
  s/.*//
}
x
D'
```

6 Tout ça pour ça ?

Avouez quand même que ce n'est pas trivial et on est loin, très loin de l'algorithme original en langage de haut niveau. On pourrait argumenter que la version compactée ne tient que sur trois lignes :

```
sed '$!N;/^\(.*\)\n\1$/{x;s/$/\n./;x;D}
P;x;/^$/!{s/^\n//;p;s/.*//;}
x;D'
```

Mais qui voudrait maintenir du code trop lent qui ressemble à un mot de passe root ? Et puisqu'on parle de vitesse, n'oublions pas de la mesurer.

```
$ dd if=/dev/urandom of=rnd.bin bs=1000 count=1000 status=noxfer
1000+0 enregistrements lus
1000+0 enregistrements écrits

$ time od -v -w16 -t x1 -An rnd.bin > out.txt
real    0m7.690s
user    0m7.480s
sys     0m0.020s

$ time ./bin2bin.v1.sh rnd.bin > out.txt
real    1m40.291s
user    1m37.690s
sys     0m0.910s

$ time ./bin2hex.v2.sh rnd.bin > out.txt
real    1m36.729s
user    1m34.790s
sys     0m0.590s
```

Non seulement le code *amélioré* n'est pas significativement plus rapide malgré la réduction du nombre de tuyaux (sans oublier les risques de ralentissement pathologique en $\Omega(n^2)$) mais en plus il est 12 fois plus lent que **od** seul, soit seulement 10K octets par seconde sur un CPU à 700MHz.

Conclusion

sed est un outil à connaitre, ne serait-ce que pour comprendre comment ne pas écrire un programme ou concevoir un interpréteur. Il peut être utile pour des tâches très simples mais d'après vous, pourquoi a-t-on créé **awk**, **perl** ou **bash** ?

Je suis conscient que le code que j'ai écrit n'est probablement pas la meilleure solution et un expert de **sed** trouvera certainement une approche plus performante. Mais c'est justement là que se situe le problème : à quoi bon créer et utiliser un langage aussi limité et limitant ? Pour le sport ? Je n'en reviens toujours pas d'avoir écrit autant d'explications pour coder l'équivalent de huit lignes de code en « C trivial » et d'avoir perdu autant de temps dessus.

Si vous ne développez pas, vous ne perdez rien, retenez juste que vous avez intérêt à éviter **sed** (à part dans des cas de force majeure, comme infecter l'ordinateur central d'un vaisseau extraterrestre avec une bombe logique, puisque **sed** est Turing complet). Si les langages ésotériques vous attirent, essayez plutôt **Fractran**.

En pratique, il vaut mieux apprendre des langages vraiment flexibles, puissants et utiles. **C**, **JavaScript** ou **Python** sont plus complexes mais vous ne vous arracherez pas autant les cheveux pour réaliser une fonction toute bête. Dans un registre similaire, **bash** a de nombreuses capacités insoupçonnées, bien que souvent enfouies dans son manuel, mais au moins on dispose de variables à volonté, de multiprocessing et même de calcul presque décent.

Donc si jamais vous avez un petit truc à coder vite fait, oubliez **sed**. Si vous ne trouvez pas la solution immédiatement dans un recueil de *one-liners*, passez vite à un autre langage. C'est un piège ! ■

Références

[1] Comme la mémoire s'évapore en trois ans ! Guidon, Yann « Compactez votre site web pour le rendre léger et rapide : avec les outils UNIX » GNU/Linux Magazine France n°144, Décembre 2011, p. 78 à 87

[2] Internet propose de nombreuses ressources sur sed, en particulier les one-liners incontournables de http://sed.sourceforge.net/sed1line_fr.html mais ils n'expliquent pas vraiment pourquoi ça marche.

[3] C'est http://www.grymoire.com/Unix/Sed.html qui m'a le plus aidé en expliquant chaque commande mais on est encore loin d'une description limpide. Pourquoi n'ai-je jamais vu le dessin qui illustre cet article ?

LA RÉCEPTION DE SIGNAUX VENUS DE L'ESPACE PAR RÉCEPTEUR DE TÉLÉVISION NUMÉRIQUE TERRESTRE

par J.-M FRIEDT

Nous avions exprimé, dans un article précédent concernant GNURadio et l'utilisation des récepteurs de télévision numérique terrestre (DVB-T) [1], notre frustration à l'incapacité à recevoir les signaux de satellites en orbite polaire basse. Cette déficience sera ici corrigée, et de façon plus générale l'ambition de recevoir des signaux venus de l'espace au moyen de récepteurs de DVB-T nous fournira l'opportunité d'explorer divers concepts de traitements de signaux radiofréquences et en particulier par traitement logiciel des signaux : communication entre processus par pipe et analyse multicanaux.

La réception de signaux venant de l'espace va induire deux nouvelles contraintes par rapport à nos explorations passées de l'utilisation de récepteurs de télévision numérique terrestre (DVB-T) pour le traitement logiciel de signaux radiofréquences. D'une part, la faible puissance du signal radiofréquence reçu va nécessiter l'ajout d'un pré-amplificateur et ajuster quelque peu la géométrie de l'antenne à la bande de fréquence considérée. D'autre part, la fugacité des signaux nous encourage à analyser simultanément plusieurs bandes de fréquence adjacentes afin de ne pas rater le signal à détecter et devoir attendre un nouveau passage de satellite dont l'orbite ne l'amène que rarement au-dessus du site du récepteur. Ces concepts seront par ailleurs mis en œuvre sur des signaux terrestres, plus simples à recevoir pour déverminer les applications proposées, mais toujours dans l'optique d'être appliqués à des signaux venus de l'espace. Nous conclurons par l'accès aux signaux transmis par la constellation GPS, où cette fois tous les satellites transmettent sur la même fréquence mais la différenciation des sources se fera par décalage Doppler de la fréquence lié au mouvement de chaque satellite, et par le code transmis qui ne sera pas abordé ici.

1 Sensibilité des récepteurs

Nous avons à notre disposition trois récepteurs de télévision numérique terrestre (DVB-T) vendus pour une somme avoisinant les 10 euros, d'apparence externe similaire mais qui s'avèrent munis d'étages de traitement de signaux radiofréquences (*frontend*) différents. Il s'agit de l'Elonics E4000, du Rafael Micro R820T[1] et du Fiticomm FC0013. Tous ces frontends radiofréquences génèrent une paire de signaux analogiques I et Q qui sont numérisés sur 8 bits par un convertisseur Realtek RTL2832U. La réception de signaux venus de l'espace nécessite de connaître la sensitibilité de chaque récepteur, *i.e.* la capacité à déceler un signal de puissance faible à l'entrée d'antenne. Quatre bandes de fréquence vont nous intéresser : la bande de communication entre aéronefs, proche de la bande utilisée par les satellites en orbite polaire basse, de 137 MHz. La deuxième bande est la bande des signaux GPS autour de 1,5 GHz. La popularité des récepteurs DVB-T pour le décodage de signaux transmis entre avions et le sol (ADS-B) suggère d'analyser la bande autour de 1090 MHz. Finalement, une bande intermédiaire peut être intéressante à caractériser car n'étant pas soumise à une demande de régulation individuelle pour l'exploiter : 434 MHz. Pour caractériser la sensibilité des divers récepteurs dans ces diverses bandes, nous utilisons un synthétiseur de signaux radiofréquences professionnel - Rohde & Schwartz SMA100A - configuré en générateur de signal modulé en fréquence ou en amplitude autour d'une fréquence de porteuse parmi la gamme proposée ci-dessus. Nous ajustons chaque récepteur DVB-T sur le gain maximal de son frontend radiofréquence - tel qu'indiqué dans les sources de `rtlsdr` à https://github.com/steve-m/librtlsdr/blob/master/src/librtlsdr.c#L945-L956 - et abaissons la puissance du synthétiseur jusqu'à ce que le rapport signal à bruit du signal démodulé atteigne un seuil prédéfini (Fig. 1). Nous avons choisi de noter la puissance radiofréquence émise pour laquelle les blocs de démodulation logiciel AM et WBFM fournis par GNURadio génèrent un rapport signal à bruit de 10 dB (tel qu'observé sur un affichage de la FFT du signal démodulé). De cette façon, nous nous affranchissons des définitions différentes de gains des étages radiofréquences qui ne présentent pas tous la même architecture (gamme de gain et position des amplificateurs). Dans tous les cas, l'acquisition se fait à 2,3 MHz, un filtre passe-bas décime le flux d'un facteur 12 avant d'entrer dans le bloc de démodulation AM ou WBFM, qui lui même décime de 5. Le résultat est affiché dans le bloc FFT avec une bande passante de 2300/60 ≃ 38 kHz sur 1024 points, ou une résolution spectrale de 37 Hz.

Récepteur	137 MHz	434 MHz	1090 MHz	1500 MHz
E4000	-105	-105	-96	-95
R820T	-112	-112	-109	-102
FC0013	-112	-111	-76	-58

Récepteur	137 MHz	434 MHz	1090 MHz	1500 MHz
E4000	-109	-110	-101	-101
R820T	-117	-119	-113	-110
FC0013	-116	-115	-80	-62

Tableau 1: Puissance de la porteuse radiofréquence (en dBm) nécessaire pour atteindre un rapport signal à bruit de 10 dB sur le signal démodulé par chaque récepteur réglé sur son gain maximal (39 dB et 45 dB pour le gain RF et IF respectivement du E4000, 50 dB pour le gain RF du R820T et 20 dB pour le gain RF du FC0013). En haut le cas de la modulation FM (modulation sinusoïdale à 3 kHz pour une excursion de 5 kHz) et en bas le cas de la modulation AM (modulation sinusoïdale à 3 kHz pour une modulation à 30%).

Il apparaît que la définition du « meilleur » composant de traitement des signaux radiofréquences dépend de l'application envisagée. À basse fréquence (137 MHz pour les satellites en orbite polaire basse, ACARS ou tour de contrôle d'aéroport), le FC0013 semble un bon choix. Cependant, il devient catastrophique pour la réception des signaux GPS autour de 1500 MHz, et le Rafael R820T se comporte bien mieux à haute fréquence (au-dessus du GHz). Il n'est pas exclu que les sensibilités meilleures du R820T viennent du remplacement du connecteur d'antenne d'origine par une embase SMA, alors que les entrées antennes des deux autres récepteurs ont été remplacées par des embases BNC se comportant un peu moins bien à haute fréquence.

2 Satellites en orbite polaire basse

Nous avons déjà présenté dans ces pages divers concepts de physique et de traitement du signal liés à la réception d'images issues de satellites en orbite polaire basse, en particulier ceux exploités par la NOAA **[2]**[2]. Pour rappel, une porteuse autour de 137 MHz est modulée en fréquence à 2400 Hz, cette seconde porteuse (sous-porteuse) audiofréquence étant elle-même modulée en amplitude (AM) afin de retranscrire l'intensité lumineuse détectée par le capteur optique embarqué sur le satellite. Cette multitude de modulations peut paraître surprenante au premier abord, jusqu'au moment où on se rappelle que le vecteur de vol est en mouvement (le satellite le long de son orbite), et la fréquence du signal vu depuis le sol est décalée par effet Doppler. Dans nos précédentes investigations, le signal radiofréquence ne nous était pas accessible : seul le signal audio-fréquence était disponible en sortie d'une chaîne de traitement analogique pour enregistrement sur carte son et démodulation de la modulation AM.

Un récepteur DVB-T n'est pas conçu pour recevoir un signal d'un émetteur distant de plusieurs centaines de kilomètres - plus de 800 km pour les satellites en orbite polaire basse. Il est donc judicieux d'ajouter un amplificateur radiofréquence entre l'antenne et le récepteur. Il est classique d'utiliser le même câble coaxial pour porter la tension d'alimentation du préamplificateur et le signal radiofréquence reçu par l'antenne - la distinction entre les deux composantes se faisant par la bande de fréquence associée. En effet, la composante radiofréquence de fréquence f se verra coupée par une inductance L d'impédance $Z_L=2\pi*L*f$ tandis qu'un condensateur C ne présente qu'une impédance $Z_C=1/(2\pi C*f)$ d'autant plus faible que f

[1] http://superkuh.com/gnuradio/R820T_datasheet-Non_R-20111130_unlocked.pdf

[2] l'activité des satellites est résumée régulièrement à http://homepage.ntlworld.com/phqfh1/status.htm

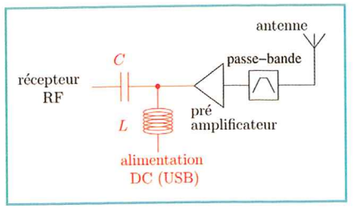


Figure 1: Schéma d'un T de polarisation (en rouge) permettant d'alimenter le préamplificateur situé après l'antenne réceptrice et le filtre basse-bande. Le condensateur laisse passer le signal radiofréquence et bloque la composante continue de l'alimentation. L'inductance bloque le signal radiofréquence et se comporte comme une résistance de valeur négligeable pour l'alimentation continue.

est élevée. En pratique, en choisissant L de l'ordre de quelques dizaines de microhenris et C de quelques nanofarads, on obtient un pont diviseur d'impédances $Z_L \simeq$ 10 kΩ et $Z_C \simeq$ 1 Ω : le signal radiofréquence passe préférentiellement à travers le condensateur plutôt que dans l'inductance. Un circuit nommé « T de polarisation » propose donc un pont diviseur qui vise à bloquer l'arrivée du signal radiofréquence dans l'alimentation par une inductance et laisser passer le signal radiofréquence vers le récepteur au travers d'un condensateur. Au contraire, une inductance se comporte comme une résistance de faible valeur pour l'alimentation continue, et le condensateur comme un circuit ouvert qui évite de détériorer le récepteur radiofréquence par l'application d'un potentiel continu sur son entrée (Fig. 1). Ce concept sera ré-utilisé pour la réception de signaux GPS qui nécessite une antenne active munie d'un pré-amplificateur.

Afin d'améliorer nos chances de réception et tenter de compenser en partie l'efficacité médiocre de l'antenne que nous proposerons (par soucis de compacité et de transportabilité plus que d'efficacité), nous avons ajouté un pré-amplificateur alimenté par le circuit que nous venons de décrire. Comme les bandes de fréquences autour de 137 MHz sont proches de la bande FM commerciale (88-108 MHz), nous nous inspirons de la pléthore de circuits d'amplification de cette bande disponibles sur le web. Nous sommes partis d'un des schémas disponibles sur le web[3] pour modifier les valeurs des condensateurs des filtres passe-bande en entrée et en sortie pour les décaler vers 137 MHz. Ce travail nécessite cependant de l'instrumentation radiofréquence spécialisée qui n'est que rarement accessible à l'amateur, en particulier du fait de l'utilisation commune des inductances à noyau d'air, de réalisation peu reproductible. Une alternative plus reproductible nous semble une approche d'assemblage de briques aux performances connues et stables, à savoir filtres à ondes élastiques de surface (SAW) et amplificateur monolithique. En ce sens, nous avons acquis sur Ebay à un tarif dérisoire (moins de 10 euros) des filtres passe-bande Epcos (désormais TDK) B3607[4] centrés sur 140 MHz et présentant environ 10 dB de pertes afin de réduire la bande passante du circuit (60 dB de rejection hors bande), et un amplificateur Hittite (désormais Analog Devices) HMC478MP86 (tout autre modèle d'amplificateur monolithique large bande conviendra) qui a le bon goût de s'alimenter en 5 V depuis le port USB d'un PC.

La réduction de la bande passante en entrée de ces amplificateurs de bande très large est nécessaire d'une part pour ne pas être saturé par les signaux puissants proches de la bande qui nous intéresse (la bande FM n'est que 30 MHz sous la bande des signaux émis par les satellites), et pour réduire le bruit thermique intégré par l'amplificateur qui pourrait cacher le signal faible à détecter.

2.1 Approche SDR avec un récepteur DVB-T

La richesse de l'approche de radio définie par logiciel (SDR) est que, mis à part une transposition analogique initiale par mélange pour amener le signal radiofréquence dans la bande passante

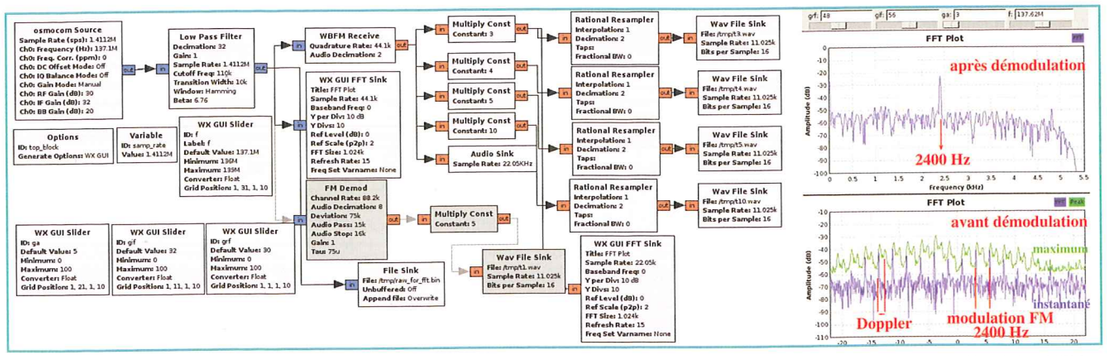


Figure 2: Gauche : blocs d'acquisition pour les images NOAA autour de 137 MHz, avec comparaison des démodulateurs WBFM et NFM, et divers gains audio. Droite : FFT du signal issu du mélangeur du DVB-T après filtrage passe-bas pour décimation. Ce signal présente les raies séparées de 2400 Hz représentatives de la modulation FM de la porteuse, raies qui se décalent par effet Doppler au cours du vol du satellite au-dessus du récepteur. En haut, le signal démodulé est quant à lui insensible à l'effet Doppler puisque toute dérive de la porteuse est compensée par le démodulateur : seul l'espacement entre les raies de modulation, et non leur position, importe.

[3] http://www.circuittrue.com/fm-booster-circuit-that-comprises-a-common-emitter-tuned-rf-preamplifier-wired-around-vhfuhf-transistor-2sc2570/
[4] http://www.epcos.com/inf/40/ds/mc/B3607.pdf, référence exacte B39141-B3607-Z510

de la chaîne de numérisation (convertisseur analogique-numérique et communication USB - bande passante d'environ 2,5 MHz), tous les traitements se font de façon logicielle. Nous avons donc sans effort accès à tous les signaux, et en particulier avant la démodulation FM pour l'extraction du signal audio à 2400 Hz.

Le décalage Doppler est un phénomène bien connu dans les fréquences audibles (la sirène de pompier plus aigue lorsque le camion s'approche et grave quand il s'éloigne), mais qui affecte tout phénomène ondulatoire, en particulier électromagnétique : si la source d'un signal périodique de fréquence f_0 se propageant à la célérité c, et le récepteur sont affectés d'une vitesse relative v, alors la fréquence du signal vu par le récepteur est $f=f_0*((c+v)/(c-v))$. Si $c>>v$, alors $c+v \simeq c$ et il reste $f \simeq f_0*(c/(c-v))$ ou plus classiquement $f-f_0 \simeq f_0*v/(c-v)$. On observe donc un décalage Doppler de la fréquence émise proportionnelle à la vitesse relative entre la source et le récepteur, le terme c−v étant pratiquement constant.

Un satellite orbitant autour de la Terre à r=870 km effectue un tour complet en 102 minutes (http://www.osd.noaa.gov/Spacecraft%20Systems/Pollar_Orbiting_Sat/NOAA_N_Prime/NOAA_NP_Factsheet.pdf - la faute d'orthographe dans le titre est d'origine). La vitesse linéaire du satellite (rayon terrestre R=40000/(2π)=6370 km) est donc de $2\pi*(R+r) \simeq$ 45500 km en 102 minutes ou 26700 km/h=7,43 km/s. En supposant que la Terre est plate[5], le satellite dépasse l'horizon (de la Terre plate ?!) avec une vitesse relative de +7,43 km/s, passe au-dessus de l'observateur pour finalement repartir vers l'horizon opposé avec une vitesse de -7,43 km/s. Seule la projection de la vitesse en direction du récepteur importe dans le calcul du décalage Doppler : l'angle entre le satellite et le récepteur au sol, toujours dans l'approximation d'une Terre plate, est une loi en *arctan()* du ratio de la position du satellite sur son orbite à son altitude. Lorsque le satellite est juste au-dessus du récepteur, la projection de la vitesse dans la direction de visée est nulle. Aux deux horizons, le décalage Doppler est asymptotiquement prédit comme $\pm$ 7430*(137*10⁶)/(3*10⁸)=$\pm$ 3400 Hz avec le dénominateur de 3*10⁸ m/s la célérité de la lumière. Nous prévoyons donc d'observer la porteuse radiofréquence décalée de $\pm$ 3400 Hz au cours du passage d'un satellite.

Un récepteur de télévision numérique terrestre (DVB-T) est principalement (pour vraiment simplifier !) un mélangeur radiofréquence pour transposer le signal reçu sur l'antenne vers une bande plus basse exploitable, et un circuit de numérisation. L'oscillateur local du récepteur DVB-T est configuré à une valeur fixe autour de 137 MHz, et l'acquisition s'effectue à environ 2 MHz, donc nous observerons les phénomènes dans la bande 137$\pm$ 1 MHz. Si seule la porteuse radiofréquence était émise, nous pourrions (sous certaines conditions) observer un signal initialement décalé de +3400 Hz, qui disparaîtrait progressivement (en suivant la loi en arctan mentionnée précédemment) lorsque le satellite passe au-dessus de notre tête (par annulation de la composante DC dans un récepteur tel que le DVB-T), et finalement réapparaître à -3400 Hz lorsque le satellite s'éloigne de nous. Heureusement, la modulation audio à 2400 Hz va nous donner un point de repère tout au long du passage du satellite. La modulation en fréquence au rythme de 2400 Hz d'une porteuse se traduit, dans le domaine spectral, par un peigne de raies éloignées de multiples de 2400 Hz de la porteuse. Si la porteuse se décale par effet Doppler, le peigne de raies va suivre ce phénomène (Fig. 3, droite). La tendance est bien observée, mais l'excursion de décalage Doppler lorsque le satellite passe d'horizon à horizon est un peu plus faible que prévu.

Afin d'affiner un peu le modèle, nous quittons l'approximation de la Terre plate, modèle rejeté par la majorité de la communauté scientifique depuis plus de 2000 ans[6], pour introduire un modèle de Terre sphérique qui fera l'affaire pour ce qui nous concerne (Fig. 3, gauche). Un récepteur situé au point P peut observer un satellite d'horizon à horizon HH'. Le satellite passe au-dessus de l'horizon en H et le vecteur vitesse $\vec{v}$ du satellite le long de l'orbite présente une projection $\vec{v}_{\parallel}$ vers P selon de la trigonométrie élémentaire : l'angle ϑ vérifie

$$\sin(\vartheta) = \frac{R}{R+r}, \text{ et } |\vec{v}_{\parallel}| = |\vec{v}| \cdot \cos(90-\vartheta) = |\vec{v}| \cdot \sin(\vartheta) = |\vec{v}| \cdot R/(R+r)$$

qui vaut donc 3000 Hz en H et -3000 Hz en H′.

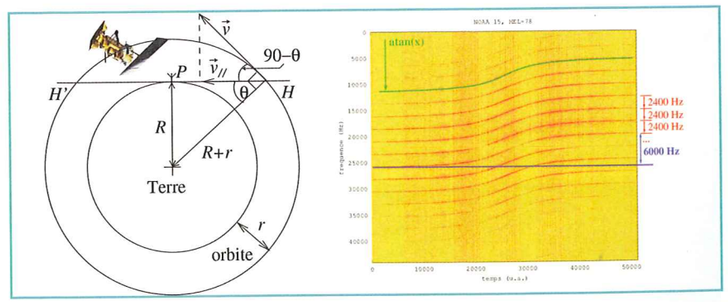


Figure 3: Gauche : schéma aidant à suivre le raisonnement de la projection du vecteur vitesse $\vec{v}$ du satellite le long de sa trajectoire en direction du récepteur P lors de son survol d'horizon à horizon HH'. L'illustration du satellite vient de http://www.oso.noaa.gov/poesstatus/. Droite : évolution dans le temps (axe des abscisses) des fréquences des signaux (ordonnée) acquis du satellite, avant démodulation. L'espacement de 2400 Hz est bien visible tout au cours du passage, tandis qu'une même raie se décale de 6000 Hz entre le passage du satellite au-dessus de l'horizon (point H sur le schéma de gauche) et son passage sous l'autre côté de l'horizon (point H') - décalage indiqué par les symboles en bleu. La fonction arctan(x) a été ajoutée sur une des traces pour guider l'œil, sans prétention d'un ajustement rigoureux des paramètres par minimisation de l'erreur.

[5] http://news.bbc.co.uk/2/hi/uk_news/magazine/7540427.stm et http://en.wikipedia.org/wiki/Flat_Earth_Society
[6] un monde plat soutenu par quatre éléphants sur une carapace de tortue ferait pourtant des images satellites intéressantes

Nous validons expérimentalement ces concepts en affichant le spectre du signal *brut* issu du récepteur DVB-T - avant toute démodulation (dont nous allons voir que le rôle est de justement s'affranchir de l'effet du décalage Doppler). Le transformée de Fourier du flux de données I/Q issu du récepteur traduit l'évolution du spectre du signal reçu en fonction de la position du satellite lors de son passage d'horizon à horizon au dessus du récepteur fixe au sol (Fig. 3, droite). En toute rigueur, cette analyse n'est valable que pour un satellite passant à une élévation de 90° au-dessus de l'observateur. En pratique, un satellite passant suffisament haut dans le ciel vérifie convenablement les résultats de ce calcul.

2.2 Résultats d'images démodulées

Le signal brut venant du satellite est démodulé de sa modulation en fréquence par les blocs GNURadio WBFM (FM large bande) ou NBFM (FM bande étroite). Nous n'avons pas observé de différence de performance de ces deux blocs, et tous les résultats présentés ci-dessous sont acquis sur les signaux sauvegardés dans un fichier au rythme de 11025 Hz tel que requis par `wxtomig` (http://www.wxtoimg.com), logiciel fermé mais gratuit de conversion des flux audiofréquences en images. Afin de respecter cette contrainte sur le débit du flux de sortie, l'acquisition s'effectue à `samp_rate`=11025*128= 1,411 MHz. Un premier filtre passe-bas décime le flux d'un facteur 32, ou 44100 Hz. L'évolution au cours du temps de la FFT de ce signal a été affichée pour générer le graphique de la Fig. 3, droite, tandis que ce même signal alimente le bloc de démodulation WBFM qui lui-même décime d'un facteur 2 pour atteindre 22050 Hz, la plus basse fréquence d'échantillonnage accessible pour la carte son d'un ordinateur portable Panasonic CF-19. Ce signal est envoyé sur la sortie audio pour fournir à l'opérateur un signal audiofréquence permettant d'ajuster l'orientation de l'antenne au cours du passage du satellite. Enfin, un filtre de décimation rejette encore un point sur 2 pour finalement atteindre les 11025 Hz du débit de données allant dans un fichier de sauvegarde au format WAV, 16 bits/échantillon.

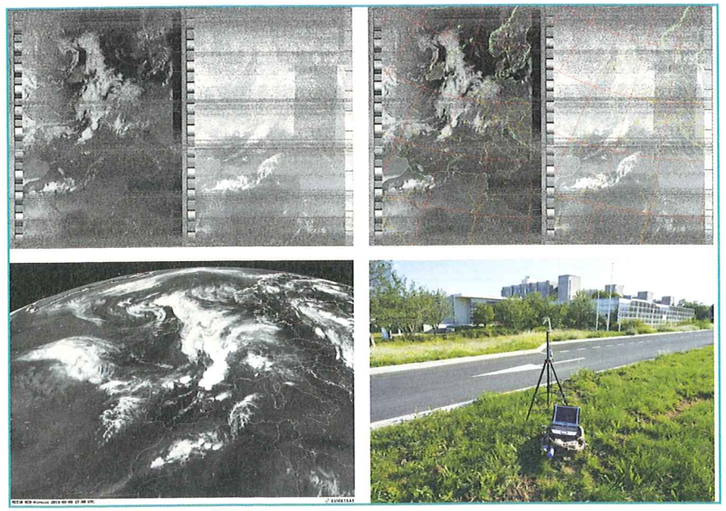


Figure 4: Capture d'image de NOAA 15 survolant Clermont-Ferrand avec une élévation maximale de 78°. Haut : l'image brute démodulée par GNURadio, sauvegardée dans un fichier audio au débit de 11025 Hz, puis décodé par wxtoimg, avec (droite) et sans (gauche) la superposition des frontières en post-traitement. Bas : gauche l'image Meteosat prise à la même date, et à droite le montage expérimental.

Une des captures résultantes, Fig. 4, compare avantageusement les images prises par notre montage et la photographie obtenue sur le web à www.wetterzentrale.de, de Meteosat10. L'image issue du récepteur DVB-T est aussi proposée avec la superposition des frontières pour mieux situer la géographie de la région observée. En particulier, noter la très mauvaise résolution de l'image Meteosat au-dessus de 60° N : les satellites géostationnaires sont peu appropriés pour observer les zones polaires au-dessus de 60° et en dessous de -60° **[3]**. Ce constat justifie l'intérêt des satellites en orbite polaire - les prévisions météorologiques en Europe étant fortement influencées par le comportement des masses d'air en région arctique par exemple **[4]** - et, puisque les données sont transmises en temps réel par ces satellites, la nécessité de placer des stations de réception à proximité de ces régions.

La contrainte d'emporter ce montage lors de nos déplacements - notamment en avion - définit la nature de l'antenne qui doit être démontable, compacte et légère. Nous avons donc sélectionné l'architecture du *crossed-dipole* dans laquelle deux dipôles taillés pour fonctionner à 137 MHz (chaque brin fait environ 55 cm de long) sont reliés par un câble coaxial de longueur $\lambda/4$ (ou, à 137 MHz, 36 cm pour tenir compte de la célérité d'une onde électromagnétique dans un cable coaxial de 66% de la célérité dans le vide). Cette architecture démontable tient dans un tube de 60 cm de long et 5 cm de diamètre, respectant les consignes des compagnies aériennes sur les dimensions de bagages en soute.

2.3 Comparaison avec RIG RX2

Nous avions auparavant **[2]** présenté la réception de signaux issus de satellites en orbite polaire basse au moyen d'un récepteur commercialisé en kit par le Remote Imaging Group (RIG) anglais, qui ne semble plus disponible aujourd'hui.

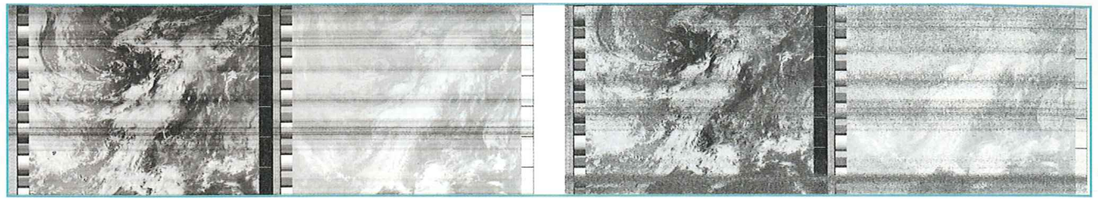


Figure 5: Comparaison des signaux acquis par un RIG RX-2 (gauche) et un récepteur DVB-T (droite) : le même signal radiofréquence est divisé vers les deux récepteurs pour traitement simultané. Bien que la structure générale des images soit similaire, le rapport signal à bruit est meilleure pour le RX2, effet particulièrement visible lorsque le satellite est près de l'horizon et propage un signal significativement atténué par l'atmosphère.

Ce récepteur reste notre référence par l'excellente qualité des images acquises.

La sortie d'antenne, avant le préamplificateur, est divisée en deux signaux. Une fraction de la puissance incidente est dirigée vers le RX2, tandis que le complément passe dans le circuit de préamplification+passe-bande et arrive au récepteur DVB-T. Les signaux audiofréquences générés par GNURadio et acquis sur l'entrée de la carte son reliée à la sortie du RX2 sont acquis simultanément. Les deux fichiers WAV sont alors traités avec `wxtoimg` qui adapte au mieux son gain en fonction du rapport signal à bruit observé sur chaque jeu de données (Fig. 5).

La bande passante réduite du récepteur RX2 (dédié uniquement aux satellites en orbite polaire basse) et l'accès pour traitement logiciel uniquement au signal démodulé audiofréquence ne répond pas aux exigences de cet article, mais le traitement du signal analogique hétérodyne proposé par le RX2 fournit à terme une sensibilité nettement meilleure qui compense la médiocrité de l'antenne dipôle utilisée pour ces expériences. Les deux images se comparent cependant avantageusement et la qualité moins bonne de l'image acquise par DVB-T ne dégrade pas significativement le résultat (Fig. 5). Nous reviendrons sur ces point plus bas lors de l'analyse multicanaux des signaux reçus de satellites en orbite polaire (section 5.3) - résultat uniquement accessible par DVB-T que le RX2 n'autorise pas du fait de sa chaîne de traitement analogique qu'il faudrait dupliquer par le nombre de canaux à traiter simultanément.

3 Bilan de liaison

Il peut paraître *a priori* surprenant qu'un récepteur de télévision numérique terrestre réussisse à capturer un signal issu d'un satellite. Devons nous réellement nous en étonner ?

Dans un premier temps, notons que la propagation en espace libre d'ondes électromagnétiques (radiofréquences) entre le satellite et le sol s'approche du cas idéal : pas de rebond ni d'atténuation sur des obstacles, pas d'interaction de l'onde électromagnétique avec le sol dans la zone dite de Fresnel. Le satellite envoie une onde sphérique qui se propage uniformément jusqu'à atteindre le récepteur. Dans ces conditions, la seule limitation à la portée d est la conservation de l'énergie qui nous informe que la puissance émise depuis le satellite se distribue uniformément sur la sphère de rayon d donc de surface $4\pi d^2$. Ceci revient à dire que la puissance décroît comme la classique loi en $1/d^2$. Si l'antenne réceptrice a une taille de l'ordre de la longueur d'onde λ, alors l'intersection de la sphère sur laquelle se distribue la puissance émise et l'antenne porte sur une surface $\lambda^2/(4\pi)$ (le terme 4π correspond à la normalisation par la surface de la sphère imaginaire qui entoure l'antenne supposée isotrope), et la fraction de puissance reçue par l'antenne réceptrice est $\lambda^2/(4\pi d)^2$. Puisque la longueur d'onde est le ratio de la célérité de l'onde c sur sa fréquence f, nous avons donc une fraction de la puissance émise arrivant au récepteur égale à $c^2/(4\pi fd)^2$. Il s'agit là des pertes de propagation en espace libre ou *free space propagation loss*[7]. Si nous passons en décibels en prenant $10\log_{10}$ de cette expression (qui facilite les calculs en transformant les produits en sommes), alors $10\log_{10}(c^2/(4\pi fd)^2) = 20\log_{10}(c/(4\pi)) - 20\log_{10}(d) - 20\log_{10}(f)$ avec le premier terme qui vaut, pour $c=3*10^8$ m/s, +148 dB. Un satellite POES de la NOAA émet une puissance de 10 W (http://en.wikipedia.org/wiki/Automatic_Picture_Transmission - pour comparaison, Radio Campus à Besançon émet 1 kW pour une portée annoncée de 25 km) ou +40 dBm (noter que le dBm est référencé à 1 mW donc 1 W qui fait 1000 mW vaut +30 dBm et 10 W valent +40 dBm), donc la puissance atteignant le sol à 870 km du satellite (λ vaut 2,2 m pour une fréquence de f=137,5 MHz) est de $+40-148+20\log_{10}(870*10^3)+20\log_{10}(137{,}5*10^6) = -94$ dBm. Nous insérerons un amplificateur d'environ 15 dB entre l'antenne et le récepteur : la puissance reçue est alors de l'ordre de -80 dBm. Cette valeur peut sembler ridiculement faible (une dizaine de pico-watt), mais le plancher d'un récepteur de type DVB-T est observé (section 1) autour de -110 dBm. Le signal reçu du satellite est donc clairement visible avec 30 dB de signal au-dessus du bruit, en accord avec nos observations expérimentales. Les raies horizontales sur les images que nous produisons (Fig. 5) sont interprétées comme des directions dans laquelle l'antenne fonctionne mal (*nuls* du diagramme de rayonnement) et pour lesquelles le bilan de liaison est donc dégradé.

[7] http://en.wikipedia.org/wiki/Free-space_path_loss

4 Connexion par pipe

Des outils de traitement logiciel du signal ont longtemps existé, bien avant l'explosion de la mode de la SDR, et en particulier à l'époque où la sortie audio d'un récepteur radiofréquence large bande (*scanner*) était connectée à l'entrée d'une carte son. Parmi les plus connus,multimon permet de décoder de nombreux modes numériques de bande passante réduite, déterminée par la fréquence d'échantillonnage de la carte son. La question consiste donc, à l'instar d'Alexandru Csete sur http://www.youtube.com/watch?v=GBmli8Vflig, de connecter la sortie de GNURadio vers un autre programme externe chargé de la conversion du flux de données audiofréquences en trame de données numérique. Sous réserve que ce dernier accepte un flux de données brutes (*i.e.* ne nécessitant pas les points d'accès d'une carte son par exemple), nous connectons la sortie du flux de données issues de GNURadio à un fichier. Cependant, le fichier sera en réalité un *pipe* nommé, c'est-à-dire une FIFO qui transmet les données qui y entrent vers un programme qui prend en entrée la sortie de la FIFO. Il s'agit donc, sous forme d'un pseudo-fichier, d'obtenir la même fonction que le | d'unix. La seule subtilité à identifier est qu'un programme faisant appel à un *pipe* nommé est bloqué tant que le second côté du tuyau n'est pas connecté à un puits. D'autre part, les données attendues par **multimon** sont celles qui viendraient d'une carte son, donc format d'entiers codés sur 16 bits et avec un débit de 22050 Hz. Nous convertissons donc, après homothétie, les flottants issus d'un bloc de démodulation vers un flux de *shorts* qui est redirigé vers un fichier. Ce pseudo fichier a été créé par la commande **mkfifo fichier**. Finalement, **multimon** se connecte à la FIFO par **multimon -t raw fichier**.

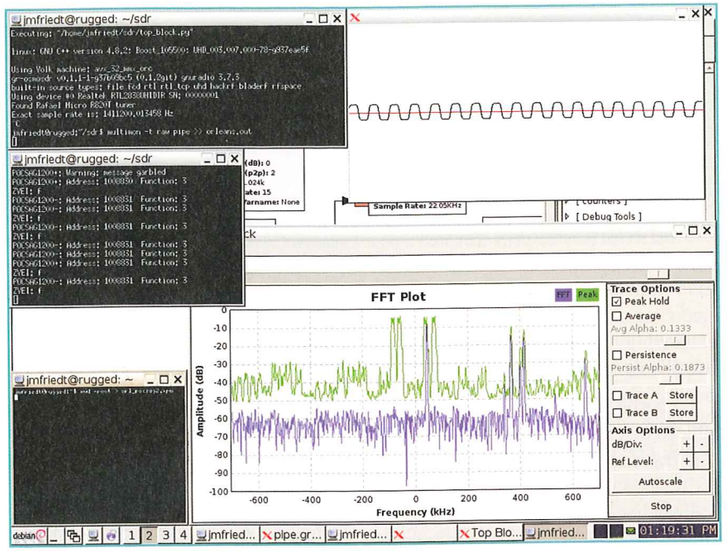


Figure 7: En haut à droite : la sortie graphique (SCOPE) de multimon présente l'évolution temporelle des bits en cours de décodage, issus de la démodulation de fréquence (FSK) du signal radiofréquence incident. En bas, le spectre affiché par gnuradio dans la bande de fréquence portant les signaux POCSAG : en bleu, la bande de fréquence servant à la transmission du signal en cours de décodage, et en vert la puissance maximale observée pour chaque point de la FFT (historique). Terminal de gauche : affichage des messages décodés par multimon ayant identifié un protocole POCSAG, ici des messages de maintenance du réseau de communication, la loi nous interdisant de diffuser des messages plus pertinents (tel que par exemple les avis de décès transmis lors des interventions de pompiers).

Notre intérêt pour **multimon** porte en particulier sur sa capacité à décoder le protocole d'encodage des signaux émis depuis le sol et retransmis vers le sol par la Station Spatiale Internationale (ISS) autour de 145,800 MHz (ou 145,825 MHz). L'encodage le plus commun pour ce mode de communication numérique amateur, issu de AX25, est APRS. La modulation est AFSK1200, c'est-à-dire des états de bits encodés sur deux fréquences audio. Cependant, l'ISS passe rarement, et l'horaire n'est pas toujours approprié pour expérimenter avec un nouveau concept.

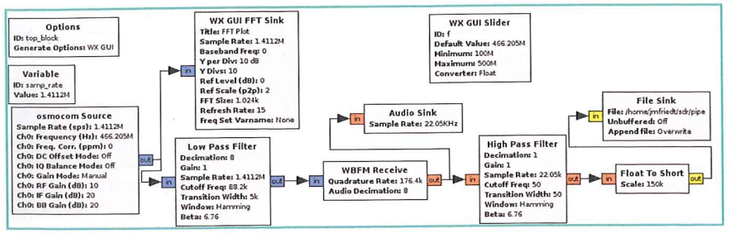


Figure 6: Traitement d'un signal modulé en fréquence puis transmis à multimon au travers d'un pipe nommé pour décodage du flux de données numériques.

4.1 Décodage de POCSAG (pagers)

Afin de tester la capacité à exploiter le *pipe* d'unix pour faire communiquer GNURadio avec un logiciel de traitement numérique des trames démodulées, nous devons accéder à une source continue de messages qui soit plus reproductible que les brefs passages de la Station Spatiale Internationale (ISS) et son transpondeur ARISS. Heureusement, le successeur des vénérables Tamtam et autres Tatoo existe encore sous le nom de e*Message et continue à diffuser, à des fins professionnelles, des messages sur les ondes électromagnétiques autour de 466 MHz[8]. Noter le commentaire publicitaire de l'exploitant *« Choisir e*Message, c'est bénéficier de nombreux avantages : un réseau indépendant et sécurisé »* - la notion de sécurité est toute relative lorsque les informations sont transmises dans un format numérique trivial à décoder ... Nous allons donc caler le récepteur DVB-T sur les fréquences autour de 466 MHz et configurer GNURadio pour émettre un flux de données continu décodable par le vénérable **multimon** (disponible en paquet précompilé sous Debian/GNU Linux).

Bien que la loi nous interdise de diffuser les messages décodés, le rapport signal à bruit des raies liées aux canaux de communication numérique visibles sur Fig. 7 est excellent et laisse présager d'un décodage aisé. Par ailleurs la vue *scope* de **multimon** permet de facilement se familiariser avec le flux de données et associer deux fréquences de modulation aux deux états possibles des bits transmis. Notre expérience depuis divers sites de Besançon est qu'il se passe rarement plus de 30 secondes sans qu'un message ne soit diffusé au format POCSAG, le plus souvent des messages automatiques de maintenance, et sinon des messages liés aux interventions de pompiers, maintenance informatique ou état de diverses installations hospitalières ou prise de fonction des personnels soignants. Noter sur la Fig. 6 un filtre passe haut entre la sortie du démodulateur FM et le convertisseur flottant en entiers : il s'agit d'une tentative timide de compenser tout biais de l'oscillateur local du récepteur par rapport à la porteuse émise par un *pager*, afin de centrer la courbe décodée par **multimon** sur 0. En effet en FM, un écart entre fréquences d'oscillateurs se traduit par un biais sur le signal décodé, et l'algorithme de détection de passage par 0 de **multimon** échoue si la courbe est trop éloignée de l'origine des ordonnées.

4.2 Décodage des liaisons numériques APRS

Avant d'être utilisé pour les transactions numériques au travers de l'ISS, l'APRS est avant tout utilisé pour les liaisons au sol. En Europe de l'ouest, la fréquence dédiée à l'APRS est 144.800 MHz, proche de la fréquence qui nous concerne pour l'ISS. Nous allons démontrer notre capacité à décoder l'APRS avec une puissance de signal inférieure à celle observée au cours d'une liaison vocale depuis l'ISS, laissant présager un décodage efficace si une transaction à travers ARISS se faisait.

La séquence ci-dessous est acquise sur une durée d'environ 30 minutes depuis le plateaux des Cézeaux sur le campus universitaire de Clermont-Ferrand, en réglant le récepteur DVB-T, connecté à l'antenne prévue pour les satellites en orbite polaire, sur 145,800 MHz.

```
jmfriedt@rugged:~/sdr$ multimon -t raw pipe
multimod  (C) 1996/1997 by Tom Sailer HB9JNX/AE4WA
available demodulators: POCSAG512 POCSAG1200 POCSAG2400 EAS AFSK1200 AFSK2400 DTMF ZVEI SCOPE
Enabled demodulators: POCSAG512 POCSAG1200 POCSAG2400 EAS AFSK1200 AFSK2400 DTMF ZVEI SCOPE
AFSK1200: fm F6FQI-12 to TU3RV6-0 via F6FQI-2,WIDE1-0,F5ZFC-3 UI  pid=F0
`yJ.oS<v/'"`:{}!4
AFSK1200: fm F4CGN-0 to APU25N-0 via F5SNV-3,F5ZFC-3,WIDE2-0 UI^ pid=F0
;F4CGN    *121247z4618.15N\00428.95E?PHG50304/Probe Enabled
AFSK1200: fm F6FQI-12 to TU3RW5-0 via F6FQI-2,WIDE1-0,F5ZFC-3 UI  pid=F0
`yJHmS v/'"`p}Bruno, 145.450 ou Relais local - U.Bat:14.3V  73's|4
AFSK1200: fm F5ZHR-4 to APFD39-0 via F5SNV-3,F5ZFC-3,WIDE2-0 UI^ pid=F0
!4658.33NV00553.19E#PHG3940 Digi APRS VHF du Mont Poupet Dept 39
AFSK1200: fm F1ZFJ-3 to APFD03-0 via F5ZFC-3,WIDE2-0 UI^ pid=F0
!4610.47N/00257.35E#  APRS DIGI ADRASEC 03 UIDIGI 1.9B3
AFSK1200: fm F4CGN-0 to APU25N-0 via F5SNV-3,F5ZFC-3,WIDE2-0 UI^ pid=F0
;F4CGN    *121247z4618.15N\00428.95E?PHG50304/Probe Enabled
AFSK1200: fm F5SNV-4 to APRS58-0 via F5ZFC-3,WIDE2-2 UIv pid=F0
AFSK1200: fm F5SNV-4 to APRS58-0 via F5ZFC-3,WIDE2-2 UIv pid=F0
=4706.44NV00307.33E#PHG3200/DIGI_NED Digi Urbain Nevers.
AFSK1200: fm F4CGN-0 to APU25N-0 via F5SNV-3,F5ZFC-3,WIDE2-0 UI^ pid=F0
;TRAFFIC  *121246z4618.15N\00428.95E?42 In 10 Minutes
AFSK1200: fm F4CGN-13 to APU25N-0 via F5SNV-3,F5ZFC-3,WIDE2-0 UI^ pid=F0
@121249z4618.15N/00428.95E_332/006g010t072r000P000p000h45b10158/Virtual Weather station / WMR928N {UIV32N}
AFSK1200: fm F4CGN-0 to APU25N-0 via F5SNV-3,F5ZFC-3,WIDE2-0 UI^ pid=F0
>120847zUI-View32 V2.03
AFSK1200: fm F6FQI-12 to TU3SP2-0 via F6FQI-2,WIDE1-0,F5ZFC-3 UI  pid=F0
`yJ`mfuv/'"`;C}!4
AFSK1200: fm F1ZFJ-3 to APFD03-0 via F5ZFC-3 UI^ pid=F0
!4610.47N/00257.35E#  DIGI APRS JN16LE LA BOSSE 03 ALLIER
AFSK1200: fm F6FQI-12 to TU3RX5-0 via F6FQI-2,WIDE1-0,F5ZFC-3 UI  pid=F0
`yJdnIEv/'"`;n}!4
AFSK1200: fm HB9MNC-9 to TV2XT8-0 via WIDE1-0,F5SNV-3,F5ZFC-3,WIDE2-0 UI  pid=F0
`l2Fr0Xj/'"8W}Bonjour!w_#!!3
AFSK1200: fm F6FQI-12 to TU3RX3-0 via F6FQI-2,WIDE1-0,F5ZFC-3 UI  pid=F0
`yJqn].v/'"`;v}Bruno, 145.450 ou Relais local - U.Bat:14.4V  73's|4
AFSK1200: fm F6FQI-12 to TU3RX1-0 via F6FQI-2,WIDE1-0,F5ZFC-3 UI  pid=F0
`yK3n*Zv/'"<=}!4
AFSK1200: fm F5SNV-3 to APRS58-0 via F5ZFC-3 UIv pid=F0
AFSK1200: fm F5SNV-3 to APRS58-0 via F5ZFC-3 UIv pid=F0
!4658.29NV00358.52E#PHG4700/DIGI_NED Digi APRS du MORVAN 818m.
AFSK1200: fm F1ZYB-3 to APFD01-0 via F5ZHR-4,F5SNV-3,F5ZFC-3,WIDE3-1 UI  pid=F0
>TT4 Digi Col du Berthiand 01
AFSK1200: fm F6IKY-5 to APU25N-0 via F5SNV-3,F5ZFC-3 UI^ pid=F0
@121253z4638.01N/00514.62E_008/003g006t071r000P000p000h55b10133Meteo Chateaurenaud 71500 {UIV32N}
AFSK1200: fm F6FQI-2 to BEACON-0 via F5ZFC-3 UI  pid=F0
=4534.22N100347.68E#PHG2625/Solar powered. Bat = 13.4V le 121254 UTC
```

Il est amusant de recevoir, depuis Clermont Ferrand, des messages issus du Mont Poupet dans le Jura, proche de Besançon où réside actuellement l'auteur.

[8] http://fr.wikipedia.org/wiki/POCSAG

Les messages comprennent principalement l'état des relais radiofréquences et des informations météorologiques, mais le plus important est de démontrer le bon fonctionnement de la réception et décodage de ce mode de transmission numérique.

4.3 Cas de l'ISS

Un passage de l'ISS est un évènement bref - la station passe d'un horizon à l'autre en environ 10 minutes - qui ne laisse pas le temps pour des réglages des paramètres du récepteur DVB-T. On validera donc le montage, la connexion du préamplificateur et de la configuration de GNURadio, sur les signaux ACARS émis à une fréquence suffisamment proche (131,725 MHz en Europe), avant de se lancer dans les signaux faibles venus de l'espace.

L'APRS nécessite deux interlocuteurs, un émetteur et un récepteur, l'ISS ne faisant qu'office de relais. Alors que ce type de mesure était courant il y a une dizaine d'années (voir par exemple la photographie prise en 2001 disponible à http://friedtj.free.fr/iss.gif), il semble que ce mode de communication soit devenu moins populaire à ce jour. Malgré notre incapacité à recevoir un message numérique lors d'un passage de l'ISS, nous avons eu la satisfaction de recevoir le 9 août 2014 un signal vocal en russe citant une observation du Stromboli qui s'avérait être justement en éruption à cette date, que nous attribuons à un des cosmonautes à bord de l'ISS. Un enregistrement de ce passage est mis à disposition du lecteur à http://jmfriedt.free.fr/140809_iss.ogv. Il y apparaît que le rapport signal à bruit de la réception (Fig. 8) est largement suffisant pour permettre un décodage des transmissions numériques. Cependant, au cours des divers essais pendant l'été 2014, aucun signal numérique n'a été détecté lors des passages de l'ISS, plaçant un doute sur le bon fonctionnement du répéteur ARISS dont le statut n'est pas clair.

5 Traitement des signaux par canaux

Le spectre radiofréquence est canalisé : diverses bandes de fréquences sont attribuées à diverses applications, et chaque sous bande contient elle-même des canaux définis par une fréquence centrale de travail et une bande passante (Fig. 9). Plus cette bande passante BW est élevée, plus le débit de communication C peut être important et uniquement limité par le rapport signal à bruit SNR de la liaison : il s'agit de la célèbre équation de Shannon publiée en 1948 **[5]** $C=BW*\log_2(1+SNR)$. À aucun moment nous ne voyons apparaître la fréquence de communication dans cette équation, si ce n'est pour des contraintes technologiques de taille d'antenne ou de facilité à atteindre BW élevé (il est technologiquement plus simple d'atteindre un BW élevé à une fréquence beaucoup élevée que BW).

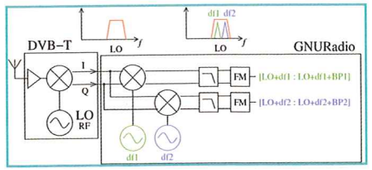


Figure 9: Canalisation des signaux radiofréquences, et stratégie de mesure simultanée des canaux par transposition numérique des diverses bandes de fréquences contenues dans la bande passante d'acquisition.

Un circuit analogique ne peut traiter qu'un canal de communication à la fois : l'oscillateur local est calé sur la fréquence centrale du canal, et les

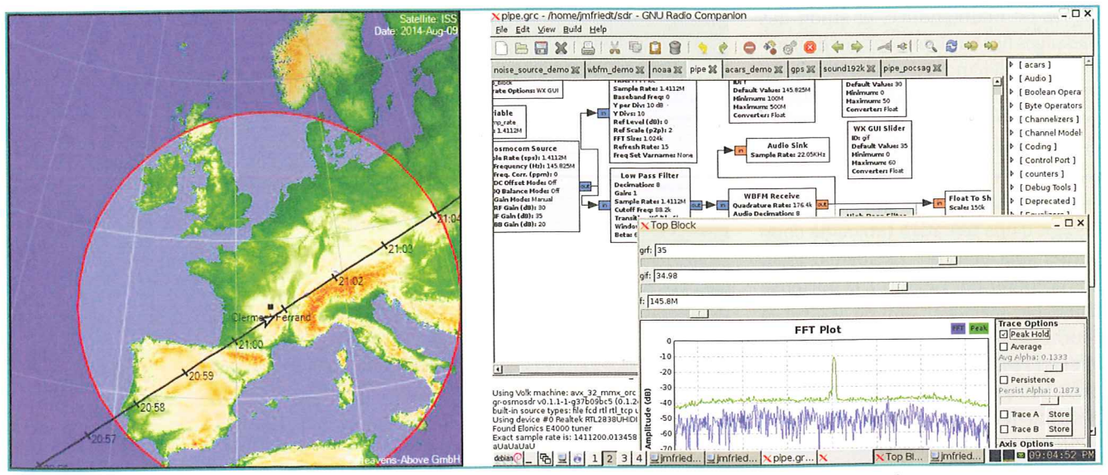


Figure 8: Gauche : zone de visibilité de l'ISS lors du passage du 9 août 2014 vers 21 h, qui comprend les îles Éoliennes, au nord de la Sicile, où se trouve le Stromboli dont il est question dans la conversation. Droite : capture d'écran de la FFT du signal reçu à 145,800 MHz pendant le passage de l'ISS, indiquant l'excellent rapport signal à bruit du signal reçu de la station spatiale.

filtres en sortie du mélangeur sont de largeur fixe pour respecter la définition de chaque canal. Changer la nature du canal implique de sortir son fer à souder et placer un nouveau filtre en sortie du mélangeur, une opération fastidieuse qui nécessite généralement par la suite une phase d'étalonnage et de qualification de la modification. Dans une approche logicielle, modifier la nature d'un filtre revient uniquement à recalculer les coefficients du filtre à réponse impulsionnelle finie (FIR) qui est généralement placé entre les flux de données I/Q issu du matériel et l'étage de démodulation logicielle (AM, WBFM ...).

La bande passante d'échantillonnage du dongle DVB-T est suffisante pour couvrir plusieurs canaux. Tous les canaux couverts par la bande d'acquisition des signaux pourront être traités en parallèle : la seule contrainte est de comprendre comment séparer les signaux associés à chaque canal pour ne pas mélanger toutes ces informations. Pour prendre l'exemple de la bande FM commerciale, les canaux sont séparés de multiples de 100 kHz. Ainsi, le récepteur avec ses plus de 2 MHz de bande passante peut couvrir plus d'une dizaine de chaînes. Dans un exercice purement académique et (dans un premier temps) sans intérêt pratique, nous allons démontrer sur cette bande de fréquence comment multiplexer les traitements pour les appliquer à plusieurs canaux et ainsi écouter simultanément plusieurs stations de la bande FM commerciale.

Le composant clé pour effectuer cette tâche est le bloc GNURadio nommé *Frequency Xlating FIR filter*. Ce composant est un nouvel étage de transposition de fréquence (par mélange avec un oscillateur local numérique cette fois) suivi d'un filtre passe-bas. Les coefficients du filtre passe-bas sont sélectionnés pour ajuster la bande passante du filtre à la largeur du canal de transmission, tandis que la fréquence de l'oscillateur local va sélectionner un canal dans les ± 1 MHz de bande échantillonée par le récepteur DVB-T et l'amener près de la fréquence nulle. Ainsi, la chaîne de démodulation qui suivra (démodulateur WBFM ou AM par exemple) se s'appliquera qu'à *un* canal bien défini et pas sur la somme de tous les signaux issus de tous les canaux dans la bande observée. Cette approche est très puissante car elle permet d'observer continûment une multitude de canaux avec un seul récepteur radiofréquence **[7]**, au lieu de multiplier les récepteurs (matériels) comme on le ferait dans une approche analogique.

5.1 Illustration du concept : recevoir deux chaînes de la FM commerciale

Pour illustrer l'exemple de la bande FM commerciale, nous sélectionnons deux canaux (les stations France Info et Europe 1 - séparées de 500 kHz donc nettement moins que les 2 MHz accessibles par la fréquence d'échantillonnage du récepteur DVB-T). Nous nous plaçons au milieu de cette bande, 300 kHz sous un des canaux et 200 kHz au-dessus de l'autre. Deux *Frequency Xlating FIR filter* sont connectés au flux de données I/Q afin d'alimenter deux démodulateurs WBFM. Les deux flux de données audiofréquences sont alors sommés pour alimenter la carte son. Le résultat est certes difficilement audible, mais démontre nettement que les deux stations FM ont été analysées simultanément - il aurait évidemment été possible d'envoyer les deux flux de données sur deux périphériques ou fichiers disctincts. La vidéo illustrant ce traitement, difficile à présenter sous forme de figure, est disponible à http://jmfriedt.free.fr/xlating_filter.mp4. Cette méthode de travail s'étend en ajoutant autant de Frequency Xlating FIR filter que de canaux à analyser, uniquement limité par la puissance de calcul de l'ordinateur qui traite le flux de données I/Q.

Un point clé de la transposition de fréquences tient dans le filtre passe-bas (Fig. 10) situé après le mélangeur - voir Fig. 11 pour un schéma similaire à celui utilisé dans la vidéo citée ci-dessus - contenu dans chaque bloc *Frequency Xlating FIR filter*. Trop étroit, la bande passante du signal utile se verra tronquée et la démodulation de pourra pas se faire efficacement du fait de la perte d'informations. Trop large, le canal adjacent risque de laisser de l'énergie baver dans le canal en cours d'étude qui sera donc perturbé (cas visible en haut à gauche de la figure du milieu de Fig. 12 ou dans le cadre du milieu-gauche dans la figure de droite de Fig. 12). GNURadio se propose de nous faire utiliser son outil de synthèse de filtre en générant dans une variable les coefficients par

```
firdes.low_pass(1,samp_rate,15000,5000,firdes.WIN_HAMMING,6.76)
```

Cependant, cet outil ne nous informe ni du nombre de coefficients (qui déterminera la puissance de calcul nécessaire pour appliquer le filtre en temps réel), ni de la nature de ces coefficients. Compte tenu des implications sur les ressources de calcul nécessaires, il est important de bien maîtriser la génération des coefficients du filtre. Pour ce faire, nous générons le `top_block.ply` à partir de gnuradio-companion et ajoutons après `self.firdes_tap = firdes_tap = firdes.low_pass(1,samp_rate,1500,500,firdes.WIN_RECTANGULAR)` les instructions Python `print len(firdes_tap)` pour connaître le nombre de coefficients requis et `print firdes_tap` pour obtenir ces coefficients eux-mêmes. L'exécution du `top_block.ply` nous informe que GNURadio a généré un filtre de 153 coefficients. Nous comparons ces coefficients avec ceux issus de la fonction `firls` de GNU/Octave qui prend en argument les fréquences pour lesquelles le gabarit du filtre est défini, et les amplitudes pour chacun de ces points, ainsi que le nombre de coefficients. Ainsi ce même filtre passe-bas se définit par `a=firls(153,[0 1500 2000 samp_rate/2]/(samp_rate/2),[1 1 0 0]);` La réponse spectrale du filtre se trace par `[h,w]=freqz(a,1);` qui cette fois donne un axe des abscisses en fréquence angulaire au lieu d'une fréquence normalisée, donc l'affichage de cette réponse spectrale est `plot(w/pi*samp_rate/2,abs(h))`. Le résultat de cette analyse est proposé sur la Fig. 10.

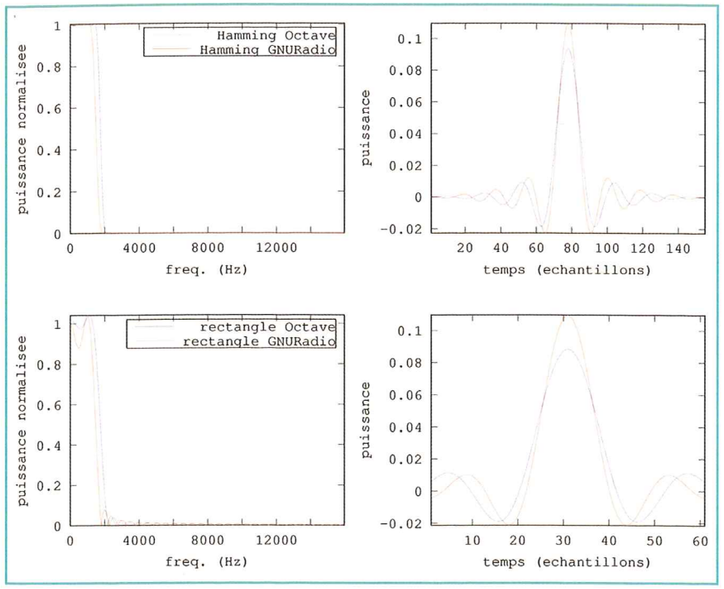


Figure 10: Comparaison des réponses spectrales (gauche) et temporelles (droite) des filtres définis par GNURadio (rouge) et GNU/Octave (bleu).

Nous constatons que pour une fenêtre de pondération du filtre de Hamming, les deux outils génèrent des coefficients induisant des filtres aux réponses spectrales similaires. Pour une pondération rectangulaire, GNURadio se comporte beaucoup moins bien que GNU/Octave, avec des oscillations importantes dans la bande passante du filtre pour une largeur de transition du filtre comparable.

5.2 Application du concept à quatre canaux POCSAG

Le concept du traitement d'une multitude de canaux est alors fort simple : le récepteur radiofréquence transpose de la plage RF d'intérêt vers une bande accessible pour le traitement numérique. Une fois le flux de données numérisé, nous transposons chaque canal vers la zone accessible par le démodulateur. Dans l'exemple de la Fig. 11, la démodulation du signal FM s'effectue dans une bande passante de 176 kHz donc nous devons ramener chaque canal dans cette bande de fréquence. Le filtre passe-bas reste toujours le même donc exploite toujours les mêmes coefficients du FIR, mais la fréquence de l'oscillateur local (NCO) est ajustée pour transposer chaque bande sous les 176 kHz de cet exemple. Ces opérations sont cependant gourmandes en ressources de calcul : nous avons constaté en pratique qu'il est peu judicieux que le nombre de canaux analysés dépasse le nombre de tuyaux de traitement (*pipelines*) disponibles sur le ou les processeurs de l'ordinateur exécutant GNURadio. Sur un Panasonic CF-19, nous avons constaté une chute significative des performances lorsque le nombre de canaux dépasse 4 (soit le nombre de *pipelines* du processeur i5).

Chaque flux de données démodulé est ensuite transmis à une des quatre instances du programme **multimon** qui est à l'écoute d'un *pipe* nommé différent. Les décodages se font donc bien en parallèle sur tous les canaux de flux encodés selon le protocole POCSAG (Fig. 12). En pratique, un seul canal est décodé à chaque instant car il est rare de voir des transmissions sur deux canaux simultanément.

Fig. 12 chacune des fenêtres du mode *scope* de **multimon** à une des raies spectrales correspondant à un des canaux POCSAG. Après une nuit de ce traitement

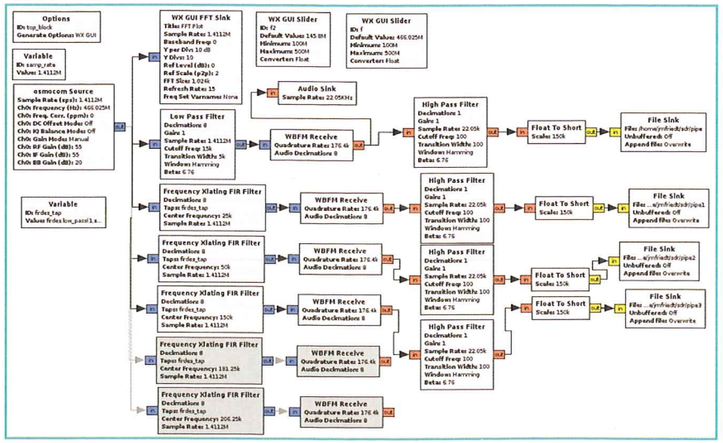


Figure 11: Graphique gnuradio-companion illustrant l'analyse simultanée de 4 canaux dans la bande d'émission des messages encodés en POCSAG. Le récepteur radiofréquence est calé sur la fréquence du premier canal, donc la démodulation se contente d'un filtre passe-bas et d'un démodulateur FM. Pour les autres canaux, le filtre passe-bas est remplacé par un Frequency Xlating FIR filter qui comporte un oscillateur de transposition de fréquence suivi du filtre passe-bas. Chacune des sorties est connectée à un gain, une conversion de flottant en entier signé sur 16 bits, et une sortie sur un fichier qui est en réalité un pipe nommé.

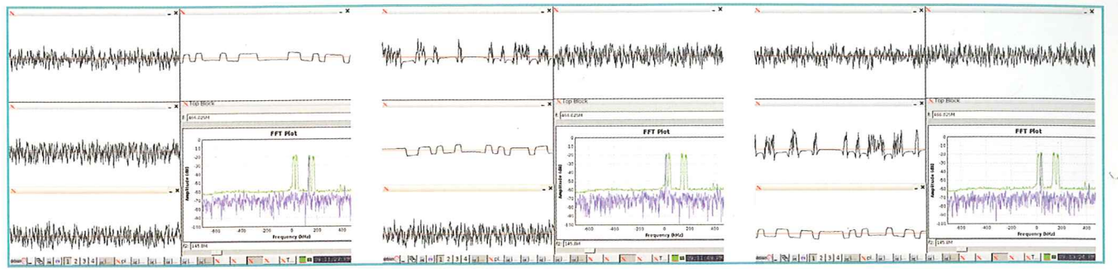


Figure 12: Quatre instances de multimon sont exécutées et connectées à quatre sorties de GNURadio, chacune issue d'une valeur différente de translation de fréquence par le Frequency Xlating FIR Filter. Des quatre affichages en mode SCOPE de multimon, un seul est actif pour chaque émission dans un canal donné (de gauche à droite : le troisième canal pour l'affichage du haut, le premier canal pour l'affichage du milieu à gauche, et le second canal pour l'affichage en bas à gauche).

(Fig. 13), les 4 canaux analysés ont clairement été définis, tandis que 358 chaînes alphanumériques ont été reçues et convenablement décodées.

5.3 Application aux satellites en orbite polaire basse

Nous mentionnons cette fonctionnalité d'analyse multicanaux pour revenir à nos satellites : plusieurs satellites émettent sur des fréquences différentes (FDMA - *Frequency Division Multiple Access*). Chaque signal occupe donc son propre canal spectral. Aux latitudes tempérées, il est rare que deux satellites passent en même temps au-dessus d'un récepteur, et la question de recevoir simultanément les deux canaux ne se pose pas. Près des pôles cependant, chaque satellite repasse toutes les 90 minutes (de par leur orbite polaire), et il est fréquent que deux satellites soient visibles

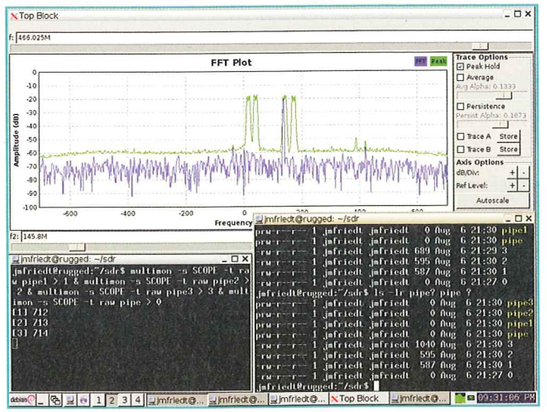


Figure 13: Démonstration du bon fonctionnement du décodage multicanaux en réduisant la puissance de calcul nécessaire en retirant l'option SCOPE de multimon (option -s). Après une nuit d'enregistrement, chaque fichier correspondant à chaque canal décodé contient plus de 10 koctets d'enregistrements dont 358 chaînes alphanumériques.

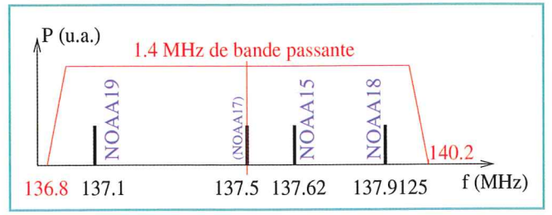


Figure 14: Les 4 canaux autour de 137,5 MHz alloués aux communications des satellites météorologiques en orbite basse. Tous ces canaux tiennent dans les 2 MHz de la bande passante d'un récepteur DVB-T.

simultanément. Dans ces conditions, le traitement simultané de plusieurs canaux prend tout son sens (Fig. 14).

Afin d'illustrer pourquoi travailler en région arctique est le paradis pour la réception de signaux de satellites en orbite *polaire* basse (et de façon générale, mais nous sortons du sujet de l'article), les tableaux ci-dessous fournissent les horaires prédits par **wxtoimg** pour tous les satellites survolant une région tempérée (par exemple pour Clermont-Ferrand) :

2014-08-13 UTC

Satellite	Dir	MEL	Long	Local Time	UTC Time	Duration	Freq
NOAA 19	N	71E	6E	08-13 14:45:22	12:45:22	11:47	137.1000
NOAA 18	N	74E	6E	08-13 17:09:38	15:09:38	11:44	137.9125
NOAA 15	N	68E	7E	08-13 18:00:02	16:00:02	11:15	137.6200

tandis qu'à la même date pour Ny-Ålesund, par 79° N, nous obtenons :

2014-08-13 UTC

Satellite	Dir	MEL	Long	Local Time	UTC Time	Duration	Freq
NOAA 19	S	61E	27E	08-13 04:47:54	02:47:54	11:54	137.1000
NOAA 19	S	89E	12E	08-13 06:28:59	04:28:59	12:01	137.1000
NOAA 18	S	59E	27E	08-13 07:12:22	05:12:22	11:41	137.9125
NOAA 15	S	62E	25E	08-13 08:08:14	06:08:14	11:24	137.6200
NOAA 19	S	75W	10E	08-13 08:09:52	06:09:52	11:57	137.1000
NOAA 18	S	87E	13E	08-13 08:53:25	06:53:25	11:58	137.9125
NOAA 15	S	89W	12E	08-13 09:48:22	07:48:22	11:38	137.6200
NOAA 19	N	75E	15E	08-13 09:50:34	07:50:34	11:57	137.1000
NOAA 18	S	76W	10E	08-13 10:34:16	08:34:16	11:46	137.9125

```
NOAA 15     S  72W   10E    08-13 11:28:16    09:28:16    11:25  137.6200
NOAA 19     N  89W   11E    08-13 11:31:22    09:31:22    12:00  137.1000
NOAA 18     N  76E   14E    08-13 12:14:57    10:14:57    11:47  137.9125
NOAA 15     N  73E   15E    08-13 13:08:01    11:08:01    11:25  137.6200
NOAA 19     N  60W    3W    08-13 13:12:36    11:12:36    11:52  137.1000
NOAA 18     N  87W   11E    08-13 13:55:44    11:55:44    11:51  137.9125
NOAA 15     N  90W   12E    08-13 14:47:52    12:47:52    11:29  137.6200
NOAA 18     N  58W    4W    08-13 15:36:58    13:36:58    11:41  137.9125
NOAA 15     N  61W    2W    08-13 16:28:08    14:28:08    11:21  137.6200
```

par exemple deux satellites qui passent simultanément en vue du récepteur vers 8h10 le matin, 9h50 ou 11h30. Cette fréquence élevée de passages explique l'intérêt stratégique des pays ayant accès aux ressources spatiales d'installer des stations d'écoute des signaux issus de satellites près des pôles, avec la question de l'utilisation de ces signaux à des fins militaires pour des régions supposées démilitarisées **[8]**.

Recevoir simultanément ces deux signaux sans devoir faire le choix cornélien de sélectionner un des deux satellites ne peut se faire que par la méthode présentée ci-dessus. À titre d'exemple, les figures 15 et 16 démontrent la réception *simultanée* de NOAA 15 et NOAA 18 émettant respectivement sur 137,620 MHz et 137,9125 MHz. L'écart de 300 kHz est largement compris dans les 1,5 MHz de bande passante d'échantillonnage que nous avons sélectionné. Cette bande passante est en fait suffisante pour décoder les 3 bandes de fréquence accessibles aux satellites de la NOAA, de 137,100 à 137,9125, selon le schéma de `gnuradio-companion` proposé dans 17. Les ré-échantilloneurs sont imposés entre la sortie audio à la fréquence la plus basse supportée de 22050 Hz, et la fréquence d'échantillonnage imposée par `wxtoimg` lors du traitement des fichiers audio de 11025 Hz. La sortie audio, somme des 3 flux issus des démodulateurs, s'avère une aide indispensable pour aider l'opérateur à orienter l'antenne sans avoir à observer sur l'écran de l'ordinateur la puissance de la raie démodulée à 2400 Hz.

On notera par ailleurs que `wxtoimg` impose, pour générer une carte de la localisation du récepteur et des traits de côtes, un nom de fichier de la forme MMDDHHMM.wav comprenant la date et heure de création (par exemple au moyen de `touch -d "08/12 17:30" 08121722.wav` pour dater le fichier acquis le 12 août lors d'un passage de satellite qui a commencé à 17h22, données audio enregistrées dans le fichier nommé `08121722.wav`). De plus, si les acquisitions sont automatisées par lancement d'un script python généré par `gnuradio-companion` en exécutant une commande retardée par le démon `at`, on prendra soin de désactiver tout accès aux interfaces graphiques et remplacer la nature du flux (*workflow*) de *WX GUI* à *No GUI*. En l'absence de cette modification, le script python ne pourra s'exécuter car l'accès aux ressources graphiques lui est interdit.

Le RX2 du RIG reste la référence en qualité d'images acquises, mais ne répond pas à notre ambition de recevoir les signaux de plusieurs émetteurs simultanément. Nous ne résistons cependant pas à la présentation d'images acquises il y quelques années dans les mêmes conditions (Fig. 18) : un tel résultat doit être accessible avec un récepteur DVB-T sous

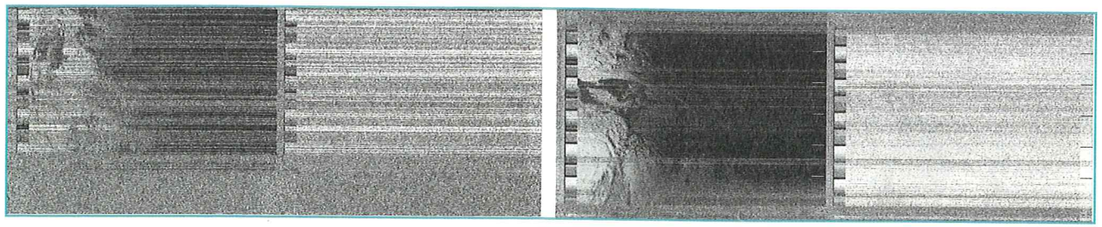


Figure 15: Signaux acquis simultanément autour de 137,620 et 137,9125 MHz lors du passage de NOAA 15 et 18 au-dessus du Spitsberg vers 9h12 heure locale.

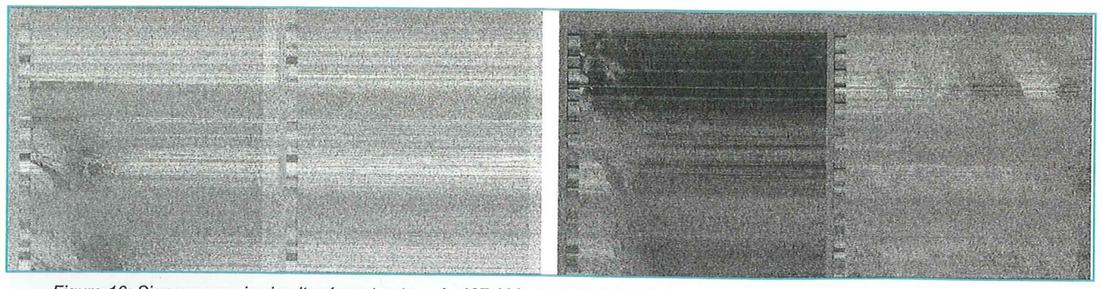


Figure 16: Signaux acquis simultanément autour de 137,620 et 137,9125 MHz lors du passage de NOAA 15 et 18 au-dessus du Spitsberg ver 8h00 heure locale. La mauvaise qualité de l'antenne, ainsi que l'illumination faible des régions arctiques aussi tard dans l'année (1er Octobre 2014), ne permettent de distinguer que peu de structures au sol au-delà des nuages aux latitudes les plus basses. Les barres de télémétrie, verticales sur ces figures, sont quant à elles bien décodées.

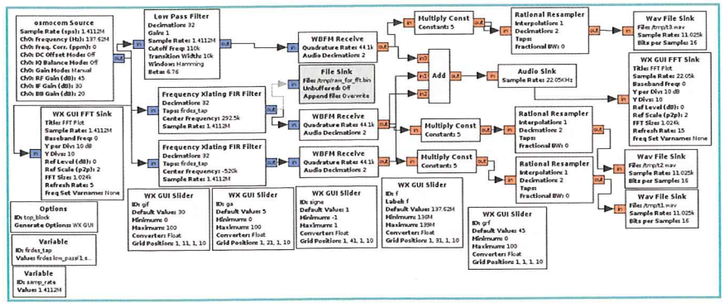


Figure 17: Schéma gnuradio-companion pour le décodage simultané de tous les satellites NOAA. Le récepteur est calé sur la fréquence de NOAA 15 de 137,62 MHz, et la fréquence d'échantillonnage de 1,4 MHz permet de décoder les signaux dans la plage de 137,62±0,7=136,92-137,32 MHz, comprenant les fréquences des autres satellites. Deux Xlating FIR filter ramènent les deux signaux en 137,1 MHz et 137,9125 MHz proche de la fréquence centrale pour une démodulation FM en vue de l'extraction des images par traitement des fichiers audio résultant.

réserve d'affiner la qualité de l'antenne pour compenser la sensibilité médiocre du récepteur, et d'effectuer les mesures en avril-mai lorsque les régions polaires sont convenablement illuminées, plutôt qu'en octobre lorsque le pôle nord est déjà dans l'ombre de la nuit.

La méthode de travail que nous avons exposé au cours de cette prose est applicable pour le traitement de tout signal dans la bande passante d'analyse. On insistera donc lourdement sur le fait que la caractéristique principale en traitement logiciel de signaux radiofréquences n'est pas la fréquence centrale de travail (qui en pratique est encore implémentée sous forme d'un mélangeur analogique avec un oscillateur local programmable) mais la bande passante d'acquisition du flux de données I/Q. Cette bande passante détermine aussi le débit des signaux décodés dans une approche de sauts de fréquence (*frequency hopping* en vue de réduire les potentielles interférences sur un canal donné) telle qu'utilisée en Zigbee ou Wifi. Dans ce cas, suivre les sauts de fréquence ne se fait *pas* en reprogrammant l'oscillateur local (tâche qui serait bien trop longue par rapport aux débits de communication et temps de stabilisation de l'oscillateur local) mais simplement en reprogrammant l'oscillateur numérique du *Xlating FIR filter* et poursuivre le traitement sur un signal qui a ainsi été convenablement ramené près de la fréquence nulle.

6 GPS

L'apothéose de cette étude tient en la réception de signaux GPS. La constellation orbite à près de 20000 km, et pourtant le signal reçu par un récepteur de télévision peut être traité pour sortir un signal pertinent du bruit. Ce résultat est certes atteint au prix d'une antenne active alimentée par un T de polarisation (vu auparavant en Fig. 1), mais est impressionant compte tenu de l'application initiale du récepteur radiofréquence. Pour le moment nos compétences dans le domaine se résument à celles de simple utilisateur d'une bibliothèque de traitement de signaux de positionnement par satellite, `gnss-sdr` disponible à gnss-sdr.org. Le rôle de ce logiciel est en particulier d'associer chaque code à chaque satellite

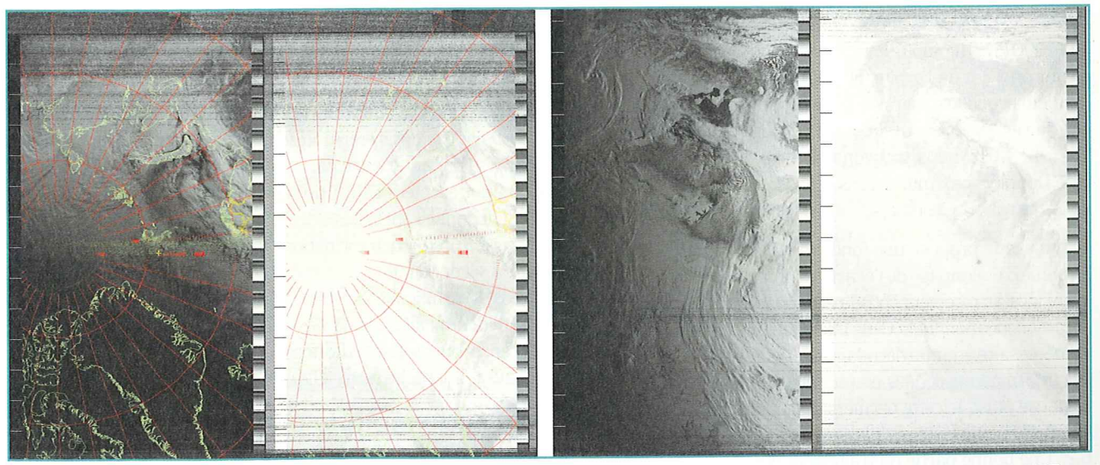


Figure 18: Images acquises au moyen d'un RIG RX2, antenne dipôles croisés, les 30 mars 2007 à 8h36 (gauche) et 31 mars 2007 à 14h09 (droite). Non seulement la sensibilité du récepteur est adéquate pour compenser l'antenne médiocre, mais l'illumination des régions polaires est ici uniforme, facilitant le décodage. La figure de gauche comporte une carte qui indique la position du pôle nord, visible sur les deux images, et du Spitsberg, avec la position de la station réceptrice comme croix jaune.

émetteur (CDMA - *Code Division Multiple Access*), mais nous nous intéresserons ci-dessous au problème amont d'identification des propriétés de la porteuse avant ce traitement.

La compilation des outils est un peu longue car faisant appel à de nombreuses bibliothèques, mais l'archive de la version 0.4 disponible sur http://sourceforge.net/projects/gnss-sdr/ est parfaitement fonctionnelle. Tous nos tests s'effectuent avec un récepteur équipé d'un *frontend* E4000 et d'un T de polarisation pour alimenter par le bus USB (5V) une antenne active[9] de gain nominal de 27 dB. Nos expérimentations n'ont pas vocation à se positionner au moyen des signaux GPS (tout récepteur OEM compact fera cela mieux qu'un récepteur de télévision numérique terrestre détourné de sa fonction d'origine) mais d'acquérir une connaissance détaillée du traitement de signaux transmis par le système de navigation.

6.1 Propriétés de la porteuse

Comme nous l'avons vu dans le cas des satellites météorologiques en orbite passe, la fréquence de la porteuse du signal émise par le vecteur de vol mobile est décalée par effet Doppler. Dans le cas du GPS et de ses satellites situés à 20000 km de la surface de la Terre, la vitesse linéaire est de 14000 km/h compte tenu de la période d'environ 12 h de l'orbite. En reprenant le calcul proposé auparavant, et sachant que la porteuse du GPS est autour de 1575,42~MHz, nous trouvons un décalage Doppler maximum, dans le modèle de Terre sphérique, de ±4,2 kHz.

gnss-sdr propose une fonctionnalité fort utile d'étalonnage de l'écart de l'oscillateur local au récepteur DVB-T par rapport à sa fréquence nominale. Cette information est nécessaire pour un décodage ultérieur des informations codées en phase sur la porteuse puisque cette dernière doit être éliminée lors de la démodulation (section 6.2). Une bonne partie du traitement des signaux GPS vise à identifier la fréquence de la porteuse synthétisée par les horloges

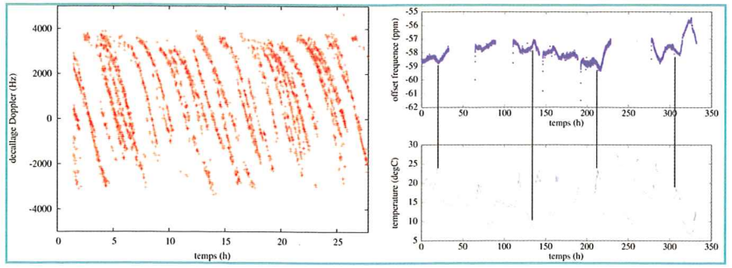


Figure 19: Gauche : évolution du décalage Doppler observé par un récepteur DVB-T dans la gamme de fréquence des signaux GPS, après correction du biais de fréquence de l'oscillateur local. Les faisceaux de courbes correspondant chacune à un satellite sont clairement visibles. Droite : évolution du biais de fréquence du récepteur DVB-T en fonction du temps (haut), et température extérieure proche de Besançon (http://www.wunderground.com/personal-weather-station/dashboard?ID=IFRANCHE9). L'influence de la température sur la fréquence de l'oscillateur est clairement visible, avec quelques point remarquables indiqués par des traits verticaux entre les deux graphiques.

atomiques embarquées dans les satellites et asservir l'oscillateur local au sol sur cette information. Deux causes de décalage des fréquences relatives du signal GPS reçu et de l'oscillateur local sont le décalage Doppler d'une part, et la dérive en température du quartz d'autre part. Nous allons étudier ces deux sources de perturbations.

En pratique, le programme **front-end-cal** fourni par **gnss-sdr** dans le répertoire de compilation **build/src/utils/front-end-cal** et prenant comme fichier de configuration **conf/front-end-cal.conf** nous fournira les données nécessaires à notre analyse. **front-end-cal** utilise l'almanach de position des satellites et la position du récepteur pour prédire quels satellites sont visibles et réduire le temps de recherche des codes lors de la démodulation des signaux reçus par le récepteur radiofréquence (services de positionnement par GPS assisté fourni pour la téléphonie mobile en particulier) : nous devons donc renseigner notre latitude et longitude dans le fichier de configuration. Une fois cette correction effectuée, nous lançons périodiquement l'étalonage du récepteur DVB-T par

```
while true; do nom=`date +%s`;echo $nom;../build/src/utils/front-end-cal/front-end-cal | tail
-26 > $nom;sleep 300;done
```

Chaque fichier de sortie fournit deux informations précieuses : une observation du décalage Doppler de chaque porteuse émise par les satellites (tous les satellites GPS émettent sur la même porteuse - la différentiation se fait sur les codes transmis dans la modulation de phase) d'une part, et d'autre part une estimation du décalage de fréquence de l'oscillateur local au récepteur DVB-T puisque l'almanach informe **gnss-sdr** du décalage Doppler prédit (connaissant la position du satellite dans l'espace et la position du récepteur sur Terre). La première information nous permet de suivre le satellite sur son orbite, tandis que la seconde information nous permet de remonter à la principale source de dérive de l'oscillateur local, à savoir la température (Fig. 19). Il s'agit donc d'une approche logicielle d'identification de dérive d'un oscillateur à quartz au sol par comparaison avec la référence fournie par les horloges atomiques embarquées dans les satellites qui se compare avec l'approche matérielle que nous avions proposé dans **[9]**.

6.2 Décodage du code modulé en phase

Le paragraphe suivant sort quelque peu du cadre concernant l'exploitation d'un logiciel libre pour décoder un signal numérique transmis par radiofréquence puisque nécessite du matériel spécialisé pour monter l'expérience. Il s'agit d'une démonstration

[9] https://www1.elfa.se/data1/wwwroot/assets/datasheets/07840051.pdf

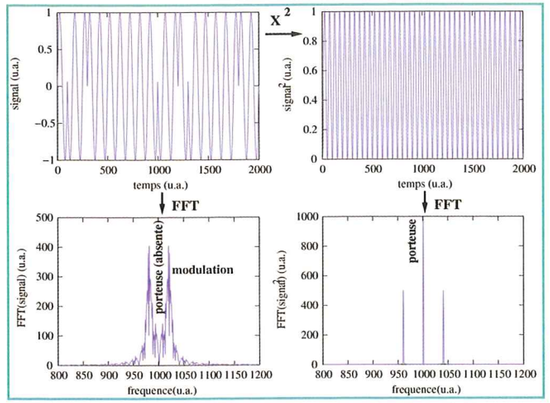


Figure 20: En haut à gauche : simulation d'un signal modulé en BPSK, un état de phase (0) représentant un état de bit et l'autre état de phase (π) l'autre état de bit. La transformée de Fourier (en bas à gauche) présente deux raies de modulation de part et d'autre de la fréquence centrale à laquelle la porteuse n'est pas visible. Prendre le carré du signal temporel (en haut à droite) induit une sinusoïde continue puisque la rotation de phase de π est devenue 2π : la transformée de Fourier de ce signal (en bas à droite) présente une unique raie dominante à la fréquence de la porteuse.

et donc soumis à des variations de fréquence différentes (Doppler pour l'émetteur, dérive thermique pour le récepteur). En l'absence de synchronisation des oscillateurs qui sont alors séparés de Δf, une rotation de phase φ s'ajoute au cours du temps t aux variations de phase de l'encodage : $\varphi = 2\pi \Delta f^* t$. Il nous faut donc une stratégie pour identifier Δf et l'annuler. La littérature nomme ce problème *carrier recovery* (reconstition de la porteuse).

L'encodage BPSK[10] présente la propriété qu'en prenant le carré du signal radiofréquence, la phase de 0° reste à 0° mais la phase de 180° devient 360° (en se rappelant ses relations trigonométriques qui indiquent que le carré d'une fonction trigonométrique induit de doubler l'argument de cette fonction, c'est-à-dire que la phase φ de l'argument devient 2φ). Ainsi, le carré du signal radiofréquence perd sa modulation et il ne reste que la porteuse au double de la fréquence (toujours selon $\sin^2(f) \propto \sin(2f)$, Fig. 20). En divisant par deux la fréquence de ce résultat, il nous reste une copie locale propre de la porteuse du signal reçu de l'émetteur. Le code ci-dessous de simulation pour GNU/Octave illustre ces concepts qui ont résulté dans la génération de la Fig. 20.

de concept sur la reconstruction du signal encodé sur une porteuse radiofréquence selon un mode de modulation représentatif de l'encodage utilisé en GPS.

Le signal GPS est encodé en phase sur une porteuse à 1575,42 MHz. Cette porteuse est modulée en phase, signifiant que le signal présente une phase de 0° pour coder un état de bit, et 180° pour représenter l'autre état. En présence de deux états, il s'agit d'une modulation BPSK, pour *Binary Phase Shift Keying* (le concept s'étend à un nombre arbitraire d'états pour transmettre plus d'un bit par intervalle de temps élémentaire). Le problème de la démodulation consiste à retrouver la séquence encodée dans les rotations de phase, sachant que les deux oscillateurs locaux (sur l'émetteur et le récepteur) ne sont pas identiques

```
1  t =[0:0.01:20];                          % le temps est continu
2  s =(sin(2*pi* t ));                      % une sinusoide continue ...
3  s (100:300)=sin(2*pi*t (100:300)+pi);    % ... dont quelques bits passent
4  s (1000:1200)=sin(2*pi*t (1000:1200)+pi); %   a 1 en ajoutant une phase de
5  s (1300:1800)=sin(2*pi*t (1300:1800)+pi); %   180 degres
6  subplot(221)
7  plot(s );axis tight ;xlabel('temps (u.a.)');ylabel('signal (u.a.)')
8  subplot(222)
9  plot(s .^2);axis tight ;xlabel('temps (u.a.)');ylabel('signal^2 (u.a.)')
10 subplot(223)
11 plot(abs(fftshift(fft(s )))); xlim ([1000 -200 1000+200]);
12 xlabel('frequence(u.a.)');ylabel('FFT(signal) (u.a.)')
13 subplot(224)
14 plot(abs(fftshift(fft(s .^2)))); xlim ([1000 -200 1000+200])
15 xlabel('frequence(u.a.)');ylabel('FFT(signal^2) (u.a.)')
```

En mélangeant le signal reçu par l'antenne avec cette copie locale de la porteuse (compensée de l'écart Δf qui est donc nul lorsque la boucle d'asservissement fonctionne), nous retrouvons une copie au sol du signal modulant. Cet asservissement de l'oscillateur local pour fournir une copie de la porteuse en vue de démoduler un signal modulé en phase s'appelle la boucle de Costas[11] (Fig. 21, page suivante).

L'expérience décrite sur la Fig. 21 vise à démoduler un signal artificiel comportant une porteuse (choisie à 1,21 GHz afin que la mise au carré nous amène dans la bande de 2,45 GHz pour laquelle des filtres passe-bande son facilement accessibles dans une implémentation matérielle de la boucle de Costas[12]) modulée en BPSK à 100 kHz (donc bien en dessous des 2 MHz de fréquence d'échantillonnage du DVB-T).

[10] Un mélangeur attaqué sur son entrée radiofréquence RF par la porteuse, et sur l'entrée locale LO par un signal carré rendant une diode passante, donc d'amplitude supérieure à ~1 V, génère sur la broche IF une porteuse modulée en BPSK tel que décrit sur la Fig. 2 du document rédigé par E. Rubiola et disponible à http://arxiv.org/pdf/physics/0608211.pdf. Le principe est que selon la polarité de LO, le commutateur passant est tantôt sur l'alternance positive, tantôt sur l'alternance négative du transformateur dont le point milieu est à la masse. Pour des questions de bande passante et de symétrie des fonctions, nous plaçons en pratique la porteuse sur LO et la modulation sur IF pour générer le signal modulé sur RF.

[11] http://gnuradio.org/redmine/projects/gnuradio/wiki/Guided_Tutorial_PSK_Demodulation

[12] dans l'implémentation matérielle de la boucle de Costas, un mélangeur voit ses deux entrées LO et IF attaquées par le signal arrivant sur l'antenne, produisant donc le carré du signal incident. La sortie RF du mélangeur ne contient alors plus que la porteuse et des raies de modulation très atténuées - de plus de 40 dB dans nos expériences. Théoriquement il suffirait de diviser par deux la fréquence de ce signal pour avoir une image de la porteuse sur laquelle asservir l'oscillateur local contrôlable en tension. En pratique un diviseur est un compteur qui agit sur une version saturée ("numérisée") du signal radiofréquence. La saturation est très sensible à toute composante spectrale autre que la raie à diviser. Le mélangeur génère une composante au double de la fréquence porteuse mais aussi une image du signal incident qu'il faut filtrer pour permettre au diviseur de fonctionner convenablemement. Les filtres passe-bande à 2,45 GHz étant facilement accessibles car utilisés pour les gadgets grand-public communiquant dans cette bande radiofréquence, nous avons choisi d'effectuer cette expérience sur une porteuse de 1,21 GHz, par ailleurs proche des 1575,42 MHz de la porteuse L1 du GPS.

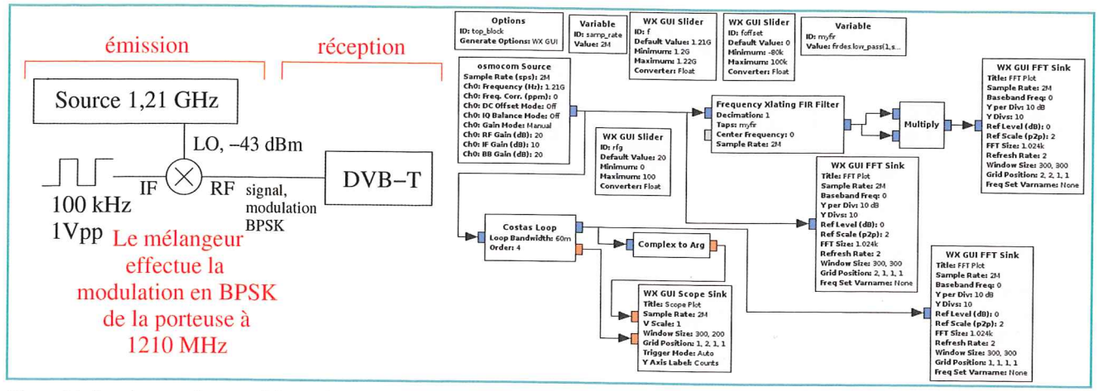


Figure 21: Gauche : montage expérimental pour générer un signal modulé en BPSK sur porteuse à 1,21 GHz (moitié gauche), et connexion de la sortie au récepteur DVB-T pour démodulation (« réception »). La puissance radiofréquence émise est ajustée pour ne pas endommager ni saturer l'étage de réception du DVB-T. Droite : schéma de démodulation gnuradio-companion pour l'extraction de la porteuse et du signal démodulé par « boucle de Costas ». La sortie complexe de la boucle fournit le signal mélangé avec l'image de la porteuse dont une représentation de la fréquence est fournie sur la sortie réelle. La constante d'asservissement de la boucle est celle fournie dans la page d'aide, à savoir 2π/100 (?!).

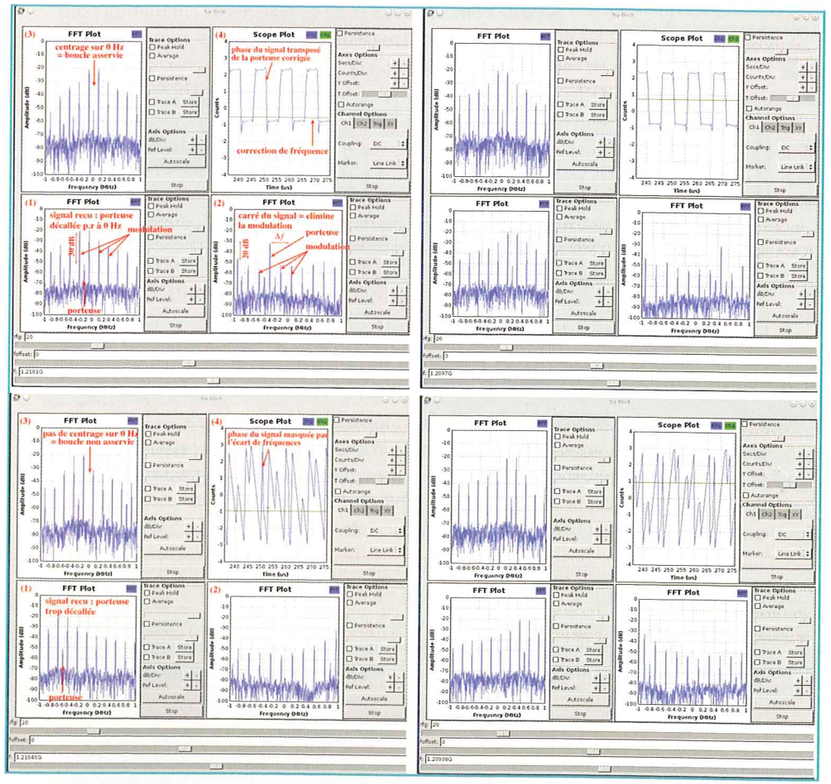


Figure 22: Résultats expérimentaux, selon le schéma de la Fig. 21, de l'application du bloc « boucle de Costas » au signal modulé en BPSK enregistré par DVB-T. Pour chaque condition de mesure : en haut à droite l'évolution temporelle de la phase du signal démodulé (bleu) et de la fréquence de la porteuse estimée (vert) ; en haut à gauche le spectre du signal en sortie de la boucle de Costas (recentrée sur 0 si la boucle est vérouillée) ; en bas à gauche le spectre en entrée du récepteur DVB-T (décalé en fréquence de Δ f≠ 0) ; en bas à droite le carré du signal reçu, qui atténue les raies de modulation et fait ressortir la raie associée à la porteuse.

Le signal acquis à 2 Méchantillons/seconde est fourni au bloc de traitement *Costas loop* d'une part, et multiplié par lui-même (pour une mise au carré) afin de démontrer l'élimination de la porteuse par ce traitement. Le *Frequency Xlating FIR filter* a pour vocation de permettre de transposer la fréquence centrale du signal mis au carré : une mise en œuvre manuelle de la boucle de Costas aurait pour vocation d'asservir l'oscillateur numérique de ce bloc pour annuler le signal d'erreur à la fréquence nulle. Cependant, GNURadio ne permet pas de fermer une boucle d'asservissement en dehors d'un bloc de traitement donc cette recherche de porteuse reste ouverte. Par ailleurs, la sortie de la boucle de Costas est d'une part une image de la fréquence estimée (sortie réelle, en orange), et d'autre part le signal transposé de l'estimation de la porteuse (sortie complexe, en bleu). Étant donné que le signal est modulé en phase, l'extraction de la phase du signal transposé de la fréquence de la porteuse (bloc *Complex to Arg*) est nécessaire pour obtenir une grandeur pertinante que nous affichons sur sortie oscilloscope.

Nous constatons que tant que l'asservissement est stable (Fig. 22, haut), la phase (en haut à droite de chaque série de graphiques, en bleu) marque des rotations toutes les 5 µs, en accord avec le signal carré modulant la porteuse. Par ailleurs, le spectre transposé de la copie de la porteuse (en haut à gauche de chaque série de graphiques) est toujours centré sur 0, quelquesoit l'écart de la porteuse, positif ou négatif (haut, gauche et droite respectivement). Le carré du signal ne présente plus des maxima écartés de la fréquence du signal modulant (±100 kHz) mais une raie unique à la fréquence de la porteuse (en bas à droite de chaque série de graphiques). Au contraire lorsque l'écart entre la fréquence locale et la porteuse devient plus importante que la bande passante de la boucle d'asservissement, la boucle n'est plus vérouillée et la phase du signal en sortie de boucle de Costas présente les rotations de 180 degrés de la modulation superposées aux rotations dans le temps de $2\pi\Delta f^*t$. La seconde composante tend à masquer la première et à rendre la restitution du signal modulant inopérante. Que nous fassions varier la fréquence de l'émetteur (1,21 GHz nominal, variation de quelques dizaines de kHz représentative du décalage Doppler) ou du récepteur (1,21 GHz nominal, variation de quelques dizaines de kHz représentative de l'effet de la température sur l'oscillateur à quartz), la boucle de Costas compense le décalage entre les deux fréquences : ce décalage est indiqué par la courbe verte dans chaque graphique en haut à droite (valeur positive si l'oscillateur local est plus élevé que la porteuse reçue, négative dans le cas contraire).

Cette discussion, limitée au cas particulier du GPS, est évidemment appliquable à tout transfert de signaux numériques, plus complexes que notre simple carré à 100 kHz, sur porteuse radiofréquence par modulation de phase.

Conclusion

Nous avons exploité un récepteur de télévision numérique terrestre pour la réception de signaux venus de l'espace. Bien que la portée puisse paraître impressionnante, il s'agit en fait des conditions idéales de propagation de signaux radiofréquences qui ne sont pas perturbés par les obstacles dans la zone dite de Fresnel entre les antennes émettrice et réceptrice. Une bonne antenne reste néanmoins une condition nécessaire à la réception d'un signal de qualité acceptable compte tenu de la sensibilité de ces récepteurs. Afin de se familiariser avec les méthodes de traitement du signal et en particulier de signaux distribués dans des canaux de fréquences adjacentes, nous nous sommes rabattus sur les signaux bassement terrestres des *pagers* encodés en POCSAG et le packet radio.

Nos prochaines investigations concerneront le décodage de signaux émis par les ballons sondes - impliquant de nouveaux concepts de traitement du signal tels que extraction de l'amplitude du signal par transformée de Hilbert et encodage de Reed-Solomon pour la correction d'erreurs. ■

Remerciements

Y. Haddab et É. Carry (Institut FEMTO-ST, Besançon) ont amélioré le manuscript par leur relecture et leurs commentaires, tandis que E. Rubiola a expliqué l'utilisation du mélangeur pour la modulation BPSK. Les études sur la réception de signaux issus du GPS sont en partie motivées par la préparation des cours du Séminaire Européen sur le Temps-Fréquence (EFTS - http://efts.eu) soutenu par le labex FIRST-TF (http://first-tf.com).

Références

[1] J.-M Friedt, G. Goavec-Mérou, *La réception radiofréquence définie par logiciel (Software Defined Radio - SDR)*, GNU/Linux Magazine France 153 (Octobre 2012), pp.4-33

[2] J.-M Friedt, S. Guinot, *La réception d'images météorologiques issues de satellites : principes de base*, GNU/Linux Magazine France, Hors Série 24 (Février 2006)

[3] B. Hoover, D. Santek, M. Lazzara, R. Dworak, C. Velden, J. Key & N. Bear, *High Latitude Satellite Derived Winds From Combined Geostationary and Polar Orbiting Satellite Data*, 11th International Winds Workshop, IWWG (New Zealand, 2012) disponible à https://www.eumetsat.int/

[4] W. Dege, *War North of 80: The Last German Arctic Weather Station of World War II*, consultable sur books.google.com

[5] C.E. Shannon, *A Mathematical Theory of Communication*, The Bell System Technical Journal 27, pp. 379-423, 623-656, (July, October, 1948), disponible à http://cm.bell-labs.com/cm/ms/what/shannonday/shannon1948.pdf

[6] M. Schillat, *A visit to Haudegen station on Nordaustlandet - the weather war in Svalbard*, dans *Polar Essays of the Arctic*, Ed. M. Schillat & I. Stone, Editorial Fuegia (2007), disponible à monika-schillat.eu/texte/e_essays.pdf

[7] B.H. Tietche *et al.*, *Software radio FM broadcast receiver for audio indexing applications*, IEEE Int. Conf. on Industrial Technology (ICIT), pp.585-590 (2012), ou B.H. Tietche *et al.*, *FPGA-Based Simultaneous Multichannel FM Broadcast Receiver for Audio Indexing Applications in Consumer Electronics Schenarios*, IEEE Trans. Consumer Electronics 58 (4), pp. 1153-1161 (2012)

[8] B. Wormdal, *The Satellite War*, CreateSpace Independent Publishing Platform (2011)

[9] J.-M. Friedt, A. Masse & F. Bassignot, *Les microcontroleurs MSP430 pour les applications faibles consommations - asservissement d'un oscillateur sur le GPS*, GNU/Linux Magazine France 98, Octobre 2007

ÉLECTRONIQUE ET DOMOTIQUE LIBRE : PARTIE 4

UN MODULE « EN VRAI » : FABRICATION MAISON, FABLABS ET PRODUCTION

par Nathaël PAJANI

Les trois articles précédents (parus dans les numéros 7 et 8) vous ont présenté la partie conception sur ordinateur d'un module pour le projet DomoTab. Il est maintenant temps de passer de la conception à la réalisation, et de fabriquer le module que nous avons conçu.

Avant de rentrer dans le vif du sujet, une note pour ceux qui n'ont pas réalisé le schéma proposé dans les articles précédents : vérifiez que vous avez prévu un moyen simple pour accéder à l'interface de programmation du microcontrôleur choisi, si possible un bouton reset, et un moyen permettant de mettre le microcontrôleur en mode programmation si besoin.

1 Dernière étape sur ordinateur

Les articles précédents nous ont permis de créer le schéma du module GPIO-Demo et de réaliser le routage pour préparer la réalisation des PCB (*Printed Circuit Board*, ou Circuits Imprimés (CI) en français). Il reste une dernière étape avant la fabrication : la génération des plans de fabrication. Cette étape dépendra grandement de la solution choisie pour la fabrication du PCB. Certains sous-traitants acceptent directement le format natif KiCad, mais ils sont assez rares et la majorité des fabricants acceptent le format Gerber. Cependant si vous souhaitez réaliser vos PCB vous-même aucun de ces deux formats ne vous sera utile et nous verrons plus tard comment procéder.

1.1 Fichiers Gerber et « layers »

Les plans de fabrication les plus utilisés sont au format Gerber. Il s'agit d'un ensemble de fichiers décrivant chacun une couche (« layer » en anglais) du circuit imprimé final : couches de cuivre, contours, perçages (trous métallisés et trous non métallisés sur deux couches différentes), sérigraphies, vernis épargne, et masques de soudure. Toutes ces couches correspondent aux couches que l'on a sous KiCad (pcbnew), rien de nouveau donc. Cependant, toutes ne seront pas forcément présentes en fonction de votre circuit et de la finition que vous désirez.

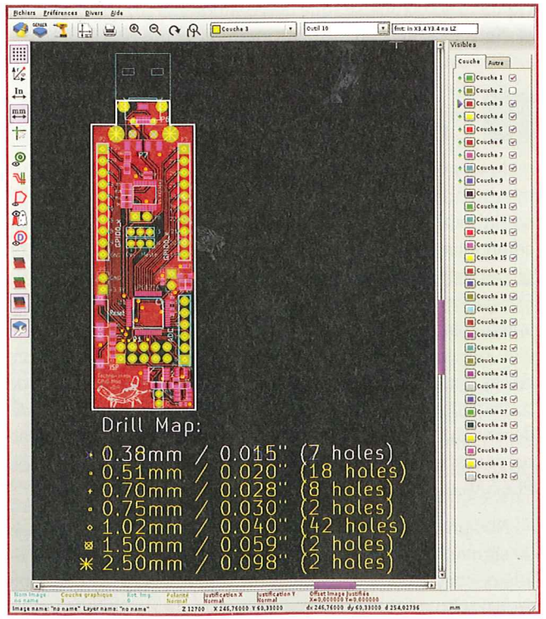


Figure 1 : Gerbview : Visualisation des fichiers gerbers

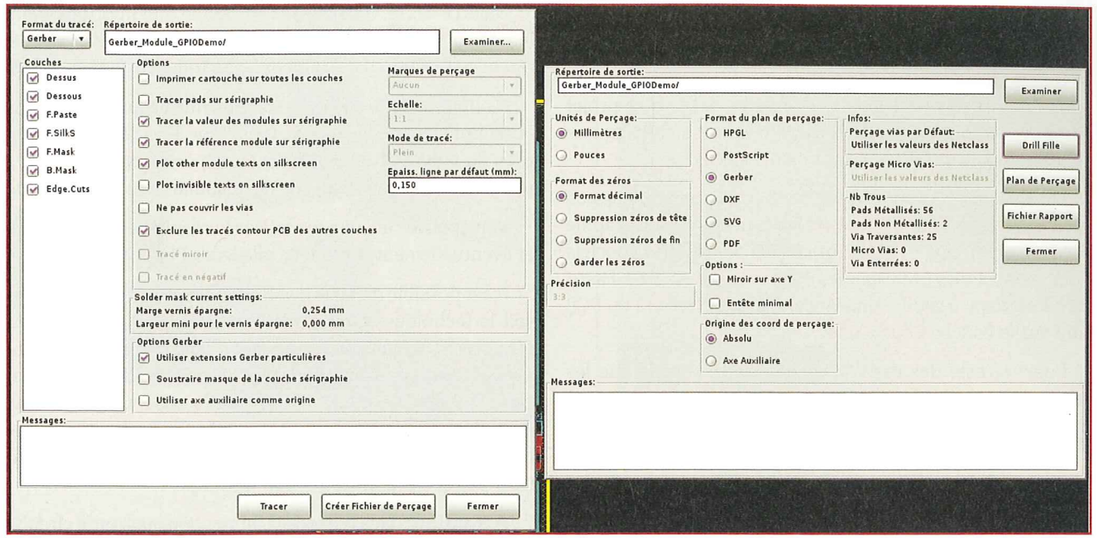


Figure 2 : Paramètres pour l'export au format gerber

Le nombre de couches de cuivre dépendra de ce que vous aurez choisi pour le routage de votre circuit, donc deux couches dans le cas du module que nous avons créé, mais cela peut aller de une seule couche de cuivre, à dix couches pour les circuits les plus complexes et les plus denses. Lorsqu'il y a plus de deux couches la fabrication « maison » sur un seul PCB n'est plus possible, et les prix grimpent.

La couche de contour correspond aux découpes du circuit imprimé. Tous les fabricants n'acceptent pas les découpes complexes, renseignez vous avant de payer si votre circuit comporte des découpes intérieures ou des découpes courbes.

Les couches de perçage correspondent aux trous pour les composants « traversants » et aux vias (pour les trous métallisés) et aux trous de fixation ou de placement de certains connecteurs qui ont des picots de placement en plastique (trous non métallisés). Lorsque vous visualisez les fichiers gerbers sous KiCad (gerbview) les trous sont représentés par des formes géométriques diverses, une par diamètre de perçage, mais les trous auront bien la forme que vous avez définie :)

Les couches suivantes sont optionnelles, mais bien utiles.

Les couches de vernis épargne (dessus et dessous) définissent un masque de vernis qui sera appliqué sur la carte pour recouvrir les zones ne nécessitant pas de soudures. C'est cette couche qui définit la couleur de la carte. Dans certains cas le vernis vert est inclus dans le prix de fabrication, mais les couleurs moins habituelles (rouge, bleu, noir, ...) impliquent un surcoût parfois non négligeable. Cette couche est générée automatiquement à partir des données des empreintes des composants et d'un paramètre global sous KiCad qui se trouve dans le menu *Dimensions -> Marge Masque des Pads* ([RefCard-5.26]). Dans la majorité des cas il n'est pas nécessaire de laisser de marge supplémentaire, et une valeur de 0 fait l'affaire. Les cas particuliers peuvent être définis dans l'empreinte du composant. L'utilisation des couches de vernis donne un rendu beaucoup plus propre et facilite grandement la soudure des composants CMS.

Les couches de sérigraphie (dessus et dessous) définissent les textes et autres tracés imprimés en blanc (dans la très grande majorité des cas) sur votre PCB. Ces textes vous aideront à placer les composants lors de la fabrication et à repérer les connecteurs ou les broches qui vous intéressent lors de l'utilisation. Bien souvent une seule couche de sérigraphie suffit.

Les masques de soudure (dessus et dessous) servent à la fabrication des pochoirs pour les composants CMS. Ils ne sont nécessaires que si vous désirez faire fabriquer un pochoir métallique en découpe laser. Ce pochoir est très utile si vous avez de nombreux composants CMS, et PCB-Pool le propose gratuitement pour les petites quantités (moins de 100 PCB), mais il est possible de le réaliser dans un FabLab en découpe laser avec une feuille de plastique (voir plus loin).

1.2 Génération des fichiers Gerbers

Sous KiCad la génération des fichiers Gerbers se fait via le menu *Fichiers -> Tracer* ([RefCard-5.27]). KiCad supporte plusieurs types de fichiers, il faut bien sélectionner « *Gerber* ».

Je vous conseille de créer un dossier pour les fichiers gerbers, cela simplifiera

la création d'une archive pour envoyer les fichiers au sous-traitant qui fabriquera vos cartes.

Il faut ensuite sélectionner les bonnes couches à inclure dans l'export. Toutes les couches actives dans votre projet se trouvent dans la colonne de gauche, à vous de vérifier que celles qui vous intéressent sont bien cochées.

Les couches de cuivres sont notées `*.Cu`, les couches de vernis épargne sont les `*.Mask`, les couches de sérigraphie correspondent aux `*.SilkS` (pour *Silk Screen*), les couches pour les pochoirs CMS sont les `*.Paste` (pour *Solder Paste*, ou pâte/crème à souder en français), et les contours du PCB sont sur la couche `Edge.Cuts`.

Pour le reste des cases à cocher, il me semble que les cases cochées par défaut conviennent, sinon reportez vous à la capture d'écran (Figure 2).

Il faut ensuite « *Tracer* » (ce qui génère les fichiers sélectionnés), puis « *Créer les Fichiers de Perçage* ». Pour les fichiers de perçage, il faut bien sélectionner le format Gerber, et cliquer sur « Drill file » et « *Plan de perçage* ». Je ne sais pas si les deux sont utiles, mais j'envoie systématiquement les deux et je n'ai pas eu de problème ni de retours ou remarques, donc je continue comme ça :)

Pour la suite, il ne vous restera plus qu'à faire une archive (zip le plus souvent) pour envoyer votre PCB en fabrication.

Vous pouvez vérifier les fichiers en utilisant l'utilitaire `gerbview` qui est intégré à la suite KiCad, et qui vous permet d'activer ou désactiver les couches une à une (après les avoir « chargées » une première fois), et ainsi vérifier que vous n'en avez pas oublié une, ou qu'il n'y a pas d'erreur par rapport à ce que vous attendiez (trop de textes sur la couche sérigraphie, plans de masse pas mis à jours, ...)

1.3 Version « fait maison » : fichiers SVG

Pour la version fait maison, les fichiers Gerbers ne sont pas utiles, mais KiCad permet l'export au format SVG des différentes couches de fabrication.

L'interface d'export se trouve dans le menu *Fichiers –> Export SVG* ([RefCard-5.28]). Dans la majorité des cas seules les couches de cuivre sont intéressantes, mais vous pouvez très bien ajouter des guides de placement sur une couche « utilisateur » (`Eco*.User`) et vouloir les imprimer aussi, ainsi que le contour du circuit, pour faciliter la découpe finale.

De même, compte tenu de la difficulté d'impression de la sérigraphie, vous pouvez imprimer une partie des textes sur la couche de cuivre, ce qui vous permet de disposer de quelques indications supplémentaires sur votre PCB sans devoir jouer du marqueur.

Dans ce cas là, vous pouvez choisir de tout imprimer dans le même fichier, ce qui vous évite de devoir replacer les différents éléments correctement à partir de plusieurs fichiers.

N'oubliez pas de cocher le tracé en miroir, que ce soit pour la technique de l'insolation (pour que l'encre soit au plus près du cuivre et le protège mieux, permettant des tracés plus fins), ou la technique de décalcomanie (voir plus loin).

Vous pouvez ensuite visualiser le résultat avec Inkscape, et éventuellement regrouper plusieurs PCB.

Il est possible de faire des PCB doubles couches, quelque soit la technique utilisée, mais attention, l'alignement doit être précis, il faudra donc penser à mettre des repères qui permettront de positionner les masques lors de la fabrication. Ces repères peuvent être placés sous KiCad ou sous Inkscape, mais il est préférable de le faire sous KiCad car ils resteront présents si vous faites des modifications, ce qui n'est pas le cas pour la version Inkscape.

Attention : dans le cas de la fabrication maison de circuits simple couches avec des composants traversants, ceux-ci doivent être placés de l'autre côté du PCB lors du placement des composants sous KiCad pour pouvoir être soudés sur les pistes.

Note : il est possible, lorsqu'il n'y a que quelques pistes sur la couche de cuivre du dessous, de ne faire qu'un PCB simple couche, et de remplacer les pistes manquantes par des morceaux de pattes de composants ou des fils.

2 Fabrication du PCB

2.1 Home Made et FabLab/ElectroLab

La fabrication maison de cartes électroniques n'est pas très compliquée et peut être à la portée de tous ... ou presque.

Pour ceux qui ont accès à l'Electrolab de Nanterre (http://www.electrolab.fr/), ils vous donneront des indications bien plus précises et détaillées que celles que je pourrais vous donner ici, et surtout, en échange d'une cotisation modique (comme pour tout FabLab) vous aurez accès à du matériel vous permettant de réaliser vos circuits imprimés.

Mais pour le reste de la France ... il faut se débrouiller (renseignez vous tout de même, le FabLab le plus prêt de chez vous a peut-être une partie du matériel, ou sera intéressé pour s'équiper).

Il y a globalement trois techniques pour fabriquer des PCB sans faire appel aux services d'un sous-traitant professionnel : la méthode classique « isolation / révélation », la méthode « décalcomanie », et la méthode usinage.

Les trois méthodes ne diffèrent que par les premières étapes, je détaillerais les étapes suivantes dans la partie « finitions » qui sera commune aux trois méthodes.

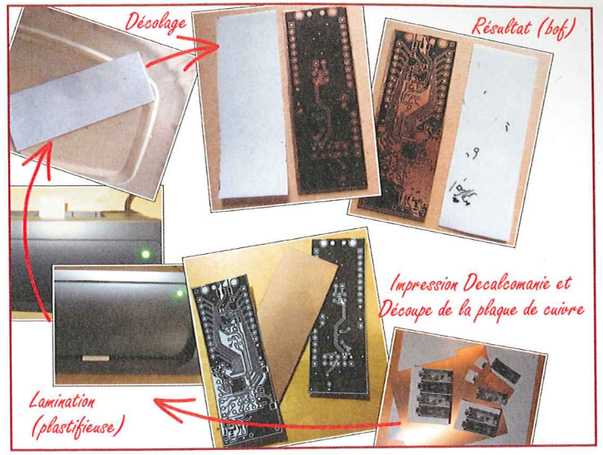


Figure 3 : Méthode "PCB Fab-in-a-Box" : exemple raté

Figure 4 : Gravure au Perchlorure de Fer

2.1.1 Insolation / révélation

Cette méthode nécessite l'utilisation de plaques pré-sensibilisées, que vous allez « graver » en trois étapes. Je n'ai personnellement pas re-testé récemment cette solution, que j'avais mise en œuvre à l'école, car je n'aime pas trop manipuler les produits chimiques dangereux, et que je n'ai pas d'insoleuse UV, mais vous trouverez de nombreux tutoriels sur internet, et je vais vous en présenter rapidement le principe, agrémenté de remarques venant de diverses personnes ayant plus l'habitude de cette technique.

La première étape est l'insolation, ou exposition. La plaque de cuivre pré-sensibilisée est recouverte d'un film sensible aux UV. La plaque doit être exposée aux UV à travers un « typon » qui n'est autre qu'une impression du circuit imprimé sur film transparent. Il vous faudra donc disposer d'une insoleuse à UV. Les UV rendront « solubles » les parties du film exposées, et le reste de la résine continuera à protéger le cuivre.

L'étape suivante, la révélation, utilise un premier produit dangereux, appelé révélateur, pour dissoudre les parties du film photosensible qui ont été exposées aux UV. Attention, j'ai bien dit produit dangereux, utilisation de gants et de lunettes de protection obligatoire.

La troisième étape, la gravure proprement dite, utilise un autre produit dangereux et surtout qui tache définitivement les tissus et qui nécessite un neutralisant pour être jeté sans polluer : le perchlorure de fer. (Résultat, moi je ne le jette pas, je remet le tout dans la bouteille quand j'ai fini). Celui-ci va attaquer le cuivre qui n'est plus protégé par le film photosensible, ne laissant du cuivre que pour les pistes, faisant apparaître le circuit imprimé. Pour accélérer le processus et éviter de gaspiller une grande quantité de produit, il est possible d'utiliser une éponge et de frotter délicatement la surface du PCB avec un coin de l'éponge imbibé de Perchlorure de fer et de chauffer légèrement le Perchlorure de fer (ne dépassez pas 60-70°). Le film de protection est ensuite retiré avec de l'alcool à brûler (il doit être possible de tester avec de l'acétone ou de l'alcool à 70°).

Note : il est possible d'utiliser du Persulfate de sodium ou d'ammonium, qui ne tachent pas, et ne polluent pas, mais ils sont plus chers et les temps d'application sont plus longs.

2.1.2 Décalcomanies

Il existe une première alternative à la solution « classique », qui utilise le principe des décalcomanies.

Cette solution est en deux étapes, et utilise des plaques de cuivre non présensibilisées, une imprimante laser (obligatoire) et une plastifieuse pour la première partie et toujours du perchlorure de fer pour la gravure.

Le dessin de votre carte doit être imprimé sur un papier à décalcomanie (je n'ai testé que celui proposé par *PCB-Fab-In-A-Box* dans leur kit, mais je pense que d'autres font l'affaire), puis transféré à chaud avec une pression régulière sur le circuit imprimé. C'est cette étape qui est critique et qu'il faut parfois reprendre plusieurs fois, surtout si vous ne pouvez pas vous procurer une des plastifieuses spécifiées par *PCB-Fab-In-A-Box*, que Amazon refuse de livrer en dehors des États-Unis. (Denis me souffle qu'il a modifié sa plastifieuse premier prix pour qu'elle chauffe plus et pour avoir plus de pression entre les rouleaux).

Pour améliorer les résultats, il faut frotter la plaque de cuivre avec une éponge abrasive type « tampongex » (éponges vertes) pour la nettoyer (sans insister pour ne pas rayer la surface), la dégraisser (avec de l'alcool ou de l'acétone), puis la chauffer à l'aide d'un fer à repasser (attention, elle devient brûlante) avant d'y déposer le décalcomanie et de passer le tout dans la

plastifieuse. Il est possible de se passer de la plastifieuse, mais le dosage de la pression avec le fer à repasser est alors critique.

Le papier est ensuite décollé en faisant tremper le tout une minute dans l'eau, et si tout c'est bien passé, l'encre est alors déposée sur le cuivre (en fait, l'encre de l'imprimante laser est un plastique, qui a fondu dans la plastifieuse et colle au cuivre). Si il y a des pistes rompues, ou des parties qui n'ont pas collées sur le cuivre, il est possible de recommencer en enlevant l'encre qui s'est déposée avec de l'acétone.

Le site *PCB-Fab-In-A-Box* conseille ensuite d'appliquer une couche de protection à l'aide d'un film plastique (fourni dans leur Kit), mais avec ma plastifieuse je n'ai pas réussi à le déposer correctement, et la gravure se passe très bien sans.

Ensuite il faut procéder à la gravure avec le perchlorure de fer, comme pour la méthode précédente, l'astuce avec l'éponge étant toujours valable.

Cette méthode permet d'obtenir des pistes assez fines, mais nécessite plusieurs essais pour prendre en main la technique d'application des décalcomanies.

2.1.3 Usinage

La troisième méthode utilise une fraiseuse numérique pour retirer le cuivre inutile par gravure mécanique et non plus gravure chimique.

Il vous faut alors transformer votre fichier SVG en tracé de découpe et la machine fait le reste. Il doit être possible de faire percer les vias et autres trous au même moment ou avec un autre outil, mais je n'ai pas pu tester cette solution car les pistes utilisées pour mes PCB ont été jugées trop fines par les personnes disposant d'une fraiseuse qui auraient pu réaliser des essais.

2.1.4 Finitions

Une fois le cuivre gravé, il reste encore deux étapes obligatoires, et une facultative.

La première étape est optionnelle, il s'agit de l'étamage chimique des pistes. Il suffit pour cela de faire tremper le circuit dans une solution d'étain chimique jusqu'à ce que les pistes aient changé de couleur. (Denis le fait « à l'ancienne », en déposant de l'étain avec le fer à souder et en retirant le surplus avec de la tresse à dessouder).

La deuxième étape est le perçage des vias et des différents trous pour les composants. L'utilisation d'une petite perceuse sur colonne permet un travail bien plus efficace et précis, mais reste que les trous des vias ne seront pas métallisés.

La métallisation des trous et des vias peut se faire de façon chimique, mais cela demande un peu (beaucoup) de matériel et une installation en conséquence, ou plus simplement en utilisant des pattes de composants que vous avez coupées lors de l'assemblage d'un autre circuit pour relier les deux côtés du circuit imprimé en les soudant de chaque côté sur les pastilles, et en coupant avec une pince coupante tout ce qui dépasse.

La dernière étape est l'application d'un vernis sur la totalité du circuit. Il faut choisir un vernis thermosoudable qui permettra de souder les composants malgré la couche de vernis. Je n'ai personnellement jamais testé, je ne connais pas la tenue des ces vernis et leurs comportements lors de la soudure autre qu'au fer à souder.

2.2 FabLabs et mutualisation des ressources

Les trois techniques présentées précédemment sont très pratiques pour réaliser un prototypage rapide, mais nécessitent toutes un peu de matériel, qui bien que peu coûteux pour les deux premières solutions représente au final entre 50 et 300 euros, et prend de la place.

Il peut être intéressant de trouver un FabLab qui dispose de tout ou partie de ce matériel, ou qui serait intéressé pour l'acquérir, libérant de la place chez vous et permettant de mutualiser l'ensemble des coûts, et éventuellement de tester les différentes solutions.

Post Scriptum : des tests sont en cours au FabLab de Lyon pour réaliser des PCB en utilisant la découpe laser et des plaques pré-sensibilisées. Nous mettrons les infos sur le Wiki lorsque la méthode sera au point.

2.3 Services de fabrication en ligne

Si vous n'avez pas la fibre DIY jusqu'au bout des doigts, ou que manipuler des produits chimiques potentiellement dangereux ne vous amuse pas plus que moi, il existe des services de fabrication un peu partout dans le monde pour faire fabriquer des circuits imprimés avec une qualité professionnelle (ce sont souvent ceux utilisés par les professionnels) à un coût raisonnable (voir défiant toute concurrence).

Attention, je ne parle pas ici de personnes qui se proposent de faire des circuits imprimés en utilisant une des méthodes citées ci-dessus à votre place, pour un tarif souvent plus élevé que les fabriquant industriels, avec les mêmes délais, et pour une qualité qui n'a rien de comparable avec ce que proposent les fabricants industriels.

Bien entendu, si vous êtes extrêmement pressés et que vous voulez pouvoir tester votre circuit dans l'heure, ceci n'est pas une solution.

J'ai testé plusieurs services de fabrication en ligne, dont *PCB-Pool*, qui produit en Irlande, *Dipoles Electroniques*, qui produit en France, *PCB Train*, qui produit en Angleterre, et *Seeed Studio* qui produit en chine.

Chaque fabricant a ses avantages et inconvénients, que je vais essayer de vous présenter, mais qui n'engagent que moi.

- *PCB-Pool* (https://www.pcb-pool.com) : les plus chers (ou pas), mais avec un réel service ajouté de vérification du circuit imprimé, la possibilité d'envoyer votre circuit au format natif KiCad (pas de possibilité d'erreur sur la génération des Gerbers), et le pochoir CMS en métal découpé au laser ... disons inclus dans le prix de base sur les quantités inférieures à 100 pièces.
  - Délai « standard » de 8 jours ouvrés, plusieurs modes de livraison, possibilité de livraison plus rapide en payant plus cher.
  - Tarif : 140€ HT pour un circuit de 100mm x 160mm double face, vernis et sérigraphie double face, avec pochoir CMS en délai 5 jours, mais cela passe à 85€ HT sur le délai standard 8 jours avec une seule face de sérigraphie.
  - SAV / Support en Français.
  - Possibilité de réalisations plus techniques (pistes plus fines, perçages plus fins) ou de délais plus cours (jusqu'à 2 jours).

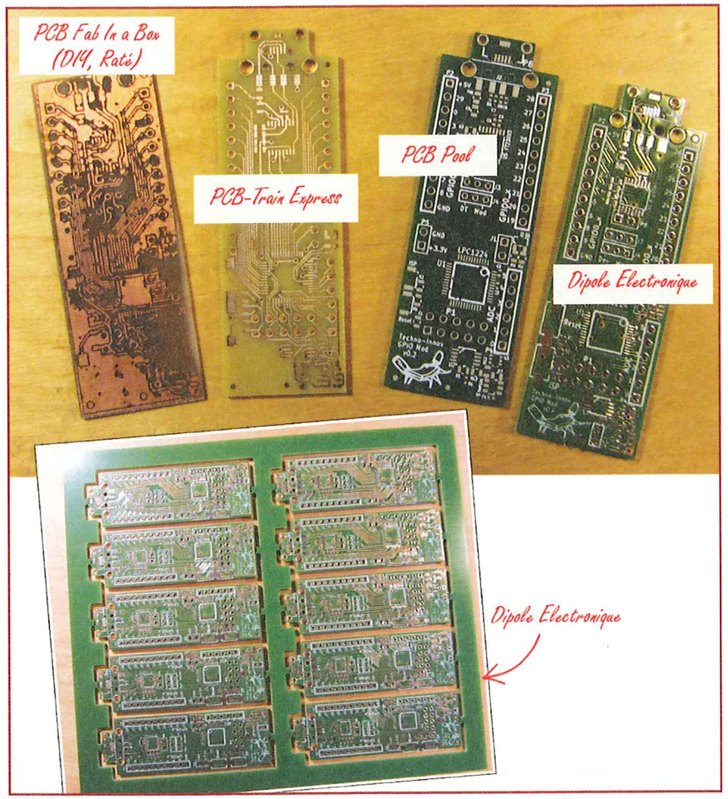


Figure 5 : Exemples de Résultats: fait maison et fabricants industriels

- *Dipole Electronique* (http://www.dipole-electronique.fr/) : même service que *PCB-Pool*, mais l'envoi des fichiers se fait par mail au format Gerber, le format est fixe (ils font les découpes du contour, rassurez vous), et vous avez moins d'options de fabrication sur l'offre « prototypes taille fixe ».
  - Délai « standard » de 5 jours ou 10 jours.
  - Tarif : 60€ HT pour un circuit de 100mm x 160mm double face, vernis et sérigraphie double face, `'sans'` pochoir CMS, en délai 5 jours, et 45€ HT sur le délai de 10 jours.
  - Pochoir CMS métal en découpe laser : forfait de 70€ HT.
  - SAV / Support en Français.
- *PCB Train* : j'ai été très déçu par leur prestation, lorsque vous sélectionnez les délais rapides il faut lire les petits caractères pour comprendre qu'il n'y aura ni vernis ni sérigraphie, la sélection des produits est globalement incompréhensible, pour un prix équivalent à PCB-Pool sans le pochoir CMS gratuit.
- *Seeed Studio* (http://www.seeedstudio.com/service/) : *Low cost*, mais haute qualité. Envoi des fichiers en ligne au format Gerber, et peu d'options (obligatoirement vernis et sérigraphie double face pour les circuits double face), mais vu le tarif, difficile de se plaindre.
  - Délai : variable, en général l'envoi se fait sous 5 jours, et possibilité de livraison gratuite si vous n'êtes pas pressés (en dessous d'un certain poids).
  - Tarif : à partir de 10$ pour 5 circuits de 5cm x 5cm, prix croissant par pas de 5cm, décroissants en volume (classique) et options payantes (finition, couleur du vernis), pour comparaison, 46$ pour 5 circuits imprimés de 10cm x 15cm.
  - Pochoir CMS en métal à partir de 70$.
  - SAV / Support en Anglais.
  - Autre avantage : la possibilité de grouper votre commande avec plein de gadgets intéressants ou d'outils ou de matériel de prototypage, comme une pince brucelle à pointe fine pour le placement des composants CMS si vous n'en avez pas.

Il en existe bien entendu beaucoup d'autres, mais je ne les ai pas testés.

Personnellement j'utilise *PCB-Pool* quand j'ai des contraintes spécifiques (délais, taille des pistes et perçages, ou que j'ai besoin d'un pochoir CMS métallique pour une production en série), et sinon soit *Dipoles Electroniques* pour les productions en série (made in France :) et *Seeed Studio* quand le budget est serré.

Ha oui, un point hyper important que j'allais oublier pour *PCB-Pool* : le suivi des étapes de production, avec photo de votre PCB entre chaque étape ... quoi ? je suis le seul que ça intéresse ? [NDLR : Non, on est tous des grands malades et on surveille aussi les commandes Mouser en retrouvant le numéro de vol Fedex pour le suivre avec Flightradar24 au dessus de l'Atlantique].

3 Achat des composants

Maintenant que votre circuit est en cours de fabrication, plutôt que de trépigner d'impatience en attendant les informations de suivi de fabrication par email, je vous propose de vous pencher sur la commande des composants (cela vous fera un deuxième sujet d'impatience pour trépigner de plus belle).

Cette étape devrait être rapide si vous avez bien noté les références des composants sélectionnés lors de vos recherches pour la conception du circuit, mais rien ne vaut une ultime vérification. Vérifiez bien les boîtiers sélectionnés (de nombreuses références sont disponibles en plusieurs boîtiers, qu'il s'agisse des condensateurs et résistances (0402, 0603, 0805, ...), des diodes, transistors, régulateurs (SOT-23, SOT363, SOT-223, TO-92, SOD, ...) ou même des microcontôleurs (QFN, TQFP, TSOP, SOIC, ...).

Deux solutions s'offrent à vous. Chercher à économiser chaque centime en écumant les sites de tous les revendeurs de France et de Navare, et payer plein plein plein (plein) de frais de port (après avoir perdu encore plus de temps), ou sélectionner un unique fournisseur au catalogue bien fourni, pour éventuellement avoir les frais de port gratuits car vous commandez plus qu'une certaine somme.

Je n'ai pas d'accords commerciaux avec les différents fournisseurs de composants (pas plus que pour les services de fabrication de circuits imprimés, malheureusement), mais je vous conseillerais tout de même ceux que j'utilise couramment, à savoir *Farnell*, *Mouser* et *DigiKey*. Je ne peux que déconseiller *Conrad* (chers, site mal fichu, et encore plus chers avec un compte pro !) et *Electronique Diffusion* (qui renvoie vers le site all-datasheets pour les documentations techniques sans donner le fabriquant des composants vendus !!!).

Ma préférence va à *Farnell* dont j'aime bien l'interface et le système de recherche (et la livraison gratuite le lendemain pour les comptes pro sur le stock Europe), mais c'est une question de préférence personnelle, et il m'arrive de commander chez les deux autres fournisseurs cités quand je ne trouve pas un composant ou quand j'ai besoin d'un gros volume, les réductions en volume étant parfois plus intéressantes (mais pas toujours).

(PS : Si vous réalisez le module GPIO-Demo ou une déclinaison, le Kit CMS à commander sur la boutique en ligne permet d'avoir tout ce qu'il faut et d'économiser quelques euros, c'est là tout l'intérêt des kits open hardware.)

Figure 6 : Module GPIO-Demo en kit : PCB, composants, pochoir, étain en pâte et instructions

4 ... et du matériel

Si vous n'êtes pas équipés, il est aussi temps d'acheter le matériel manquant, il vous sera de toute façon utile pour plein de projets, il n'y a rien de spécifique.

J'ai dressé une petite liste qui devrait vous permettre de vous en sortir avec votre circuit :

- Un fer à souder. (Je déconseille fortement les fers sans station permettant de régler la température, une station qui tient la route coûtant dans les 60 euros, ce n'est pas un investissement déraisonnable. Lisez l'article sur ce sujet dans le premier numéro de « Hackable Magazine » si vous êtes sceptiques.)
- De l'étain. (Personnellement, je n'ai pas trouvé d'étain sans plomb utilisable, mais vous pouvez essayer.)
- De la tresse à dessouder.
- Une pince coupante. (De préférence avec la ligne de coupe au bord des mâchoires pour pouvoir couper à ras.) [NDLR : astuce à la MacGyver, le coupe-ongle peut s'avérer être un outil de secours fort pratique].

- Une pince brucelle.
- Un multimètre (Avec fonction « bip » pour tester la continuité des pistes ... ou les courts-circuits :( )
- Une plancha. (Si si, vous avez bien lu. Tout type de poêle électrique devrait faire l'affaire, lire l'article de Sparkfun (en anglais, lien en fin d'article) sur le sujet)
- Une petite loupe. (Toujours pratique pour vérifier les soudures.)

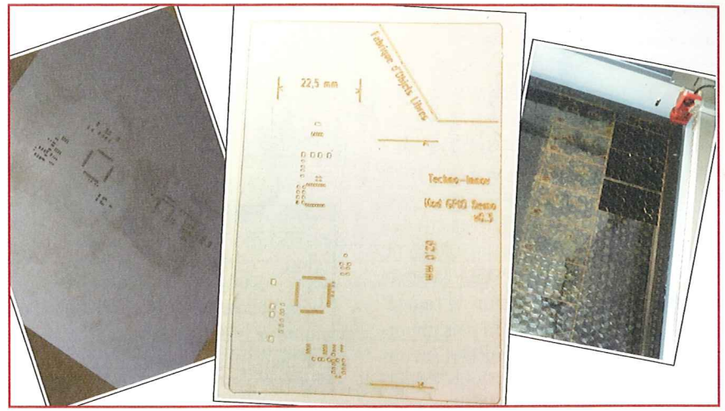


Figure 7 : Masque de soudure métallique (PCB-Pool) et plastique (DIY - FabLab)

5 Masque de soudure

Il y a heureusement une deuxième activité que vous pouvez pratiquer (trépigner ne sert à rien (si si, je vous assure), et ça fatigue (au moins votre entourage)) en attendant vos cartes électroniques (et vos composants maintenant). Si vous n'avez pas commandé le pochoir CMS en métal et que vous choisissez cette solution pour la soudure, c'est la fabrication du pochoir CMS en plastique (à condition d'avoir accès à une machine de découpe laser, mais si ce n'est pas le cas il y a une alternative ... qui vous fera trépigner d'impatience une fois de plus).

Le Pochoir CMS (ou masque de soudure) est habituellement une fine feuille de métal (120 à 175 microns d'épaisseur) qui est utilisée par les fabricants de cartes électroniques pour mettre la soudure sur le circuit imprimé avant le placement des composants et le passage en four pour la soudure. Cette plaque est percée de trous, un pour chaque « pad » de chaque composant CMS, délimitant un volume équivalant à la quantité de soudure nécessaire pour souder la patte du composant sans faire de pâté pour qu'il n'y ait pas de court circuit.

Le soucis du pochoir CMS en métal c'est son prix, impossible d'en inclure un dans chaque Kit du module GPIO-Démo. Il me fallait une solution alternative, et j'ai donc pensé à faire la même chose avec la machine de découpe laser du FabLab de Lyon (La Fabrique d'Objets Libres). Le problème, c'est qu'elle ne peut pas couper le métal, j'ai donc du me rabattre sur le plastique.

L'étape la plus compliquée est de trouver des feuilles de plastique dont l'épaisseur est spécifiée par le commerçant, et de la même épaisseur que les pochoirs métalliques, donc de l'ordre de 150 microns. L'épaisseur est primordiale, car c'est elle qui donnera la quantité de soudure déposée.

Les premiers plastiques que j'ai trouvé avec l'épaisseur spécifiée sont des rhodoids alimentaires (175 microns et 130 microns), mais ils sont difficiles à acheter autrement qu'en ligne, et sont souvent de grande taille (60cm x 40cm), donc pas pratiques à manipuler, mais en cherchant bien on en trouve de dimension raisonnable (A4). Et je suis tombé par hasard sur des feuilles plastiques pour premières pages de dossiers (ceux que l'on relie, que l'on doit rendre en quatre exemplaires pour nos projets et qui finiront au fond d'une poubelle dans quelques années^W semaines) dont l'épaisseur est spécifiée sur la pochette : 130 microns, pile ce qu'il faut, et bien moins cher que le rodhoid alimentaire, si ce n'est qu'il faut en acheter 100 feuilles.

Après quelques tests réalisés au FabLab, j'ai fini par obtenir un pochoir plastique très convenable à partir de la couche « *Solder Paste* » de KiCad, mais il faut cependant faire quelques retouches du fait de la rétractation du plastique quand il est fondu par le laser. Pour obtenir un résultat correct en prenant en compte ce problème de rétractation du plastique qui donne des trous plus grands que prévu, il faut diminuer la taille de certaines pastilles.

Pour le faire, j'utilise Inkscape après avoir exporté la couche « *Solder Paste* » en SVG. Il faut commencer par dégrouper toutes les pastilles, sans quoi leurs positions changent quand on applique la réduction au groupe. Il faut ensuite ne laisser que les bords, qui seront les traits de coupe, car KiCad exporte des rectangles pleins.

Vous pouvez enfin sélectionner les pastilles « de taille moyenne » pour les réduire de 20 à 60% (selon l'épaisseur du plastique que vous avez trouvé, le type de plastique, et la puissance du laser lors de la découpe, et le type de composant).

Je sais que « de taille moyenne » ne veut rien dire du tout, et je vais donc l'expliquer. Il n'est pas nécessaire de faire de modification pour les grandes pastilles comme les pattes de fixation de connecteurs ou les plans de masse des régulateurs de tension. Passé 3mm x 3mm, il faut laisser la pastille telle quelle. Il en va de même pour les composants en

boîtier 0402, car si le trou est trop petit la soudure colle au plastique et reste dans le trou du pochoir quand on le retire, ce qui n'est pas l'effet recherché. Pour tous les autres trous, il faut appliquer une réduction, plus ou moins grande en fonction de la feuille de plastique, mais aussi du type de composant et de l'espacement des broches. Pour les circuits intégrés comme le microcontrôleurs du module, mais aussi l'EEPROM, la porte multifonction, ou le capteur de température, je fais une réduction plus importante (50% à 60%) que pour le reste des composants (20% à 40%), pour limiter ou supprimer les courts-circuits lors de la soudure qui se produisent quand il y a trop de soudure.

Une fois les pastilles réduites (en veillant bien à conserver leur position), vous pouvez ajouter un contour pour que le pochoir se détache de la feuille. Veillez dans ce cas à le faire plus grand que le circuit imprimé, ce qui permettra de le fixer, et faites attention à ne pas faire un contour continu, sans quoi le plastique risque de se replier pendant la découpe du fait de l'aspiration. Pour éviter cela, il suffit de laisser de petits espaces qui rattachent le pochoir au reste de la feuille (moins de 1mm de large), ce qui maintiendra le pochoir pendant la découpe et permettra de détacher le pochoir sans outils, juste en tirant dessus.

Vous pouvez maintenant aller au FabLab le plus proche si vous en avez un qui ait une machine de découpe Laser pour faire votre découpe. Si ce n'est pas le cas, vous pouvez demander à un FabLab qui en a une, si ils peuvent vous faire la découpe en leur envoyant le plastique et une enveloppe timbrée pour le retour, avec un petit quelque chose pour le travail, négocié avec eux au préalable (et tous les FabLab de France qui ont une découpe Laser vont me maudire). Ou bien utiliser les services de Pololu, qui proposent cette solution depuis quelques temps, soit parce qu'ils ont fait le même constat et eu la même idée que moi, soit parce que mon idée a voyagé (laissez-moi croire en la deuxième solution svp), même si ils proposent ce service un peu cher à mon goût. http://www.pololu.com/product/446.

Et enfin, avant d'utiliser le pochoir, je le « racle » avec une règle métallique pour enlever les surépaisseurs de plastique sur les bords des trous, du fait de la rétractation du plastique pendant la découpe.

Vous voila avec un (joli) pochoir CMS à moindre frais (en tout cas normalement bien moins de 70 euros), même s'il ne tiendra pas des centaines d'utilisation, et ne peut pas être nettoyé à l'acétone. J'en ai utilisé certains une dizaine de fois, et je me sers d'une petite épingle pour nettoyer les restes d'étain une fois sec. (Denis me souffle que le Mylar pourra être nettoyé à l'acétone, et il devrait donner le même résultat pour les pochoirs).

6 Assemblage

Après quelques (trop nombreux) jours d'attente, vous avez enfin reçu votre circuit imprimé et vos composants, et vous allez pouvoir attaquer la suite des réjouissances :)

Bien entendu, il existe des sociétés spécialisées dans la réalisation des étapes suivantes pour des prototypes, mais les coûts sont souvent non négligeables, et le résultat pas toujours au rendez-vous (composants inversés, mal soudés, ...). A noter que certaines sociétés de fabrication de PCB proposent ce service ... à condition que vous utilisiez un logiciel propriétaire pour envoyer les données d'assemblage :(

6.1 Dépose de l'étain

Mais attention, ne vous précipitez pas, il faut commencer par un petit check-up, il est encore temps. Avant de mettre de l'étain plein partout, vérifiez que vous avez bien reçu tous les composants, que vous n'en avez pas oublié, et que les empreintes correspondent bien. Il est encore temps de commander les références manquantes, en revanche, une fois l'étain appliqué avec le pochoir, il commence à sécher, et même si il est toujours possible de positionner les composants plus tard, c'est bien plus sympa quand ils « collent » dans le « gel » qui compose l'étain « en pâte » (ou crème à braser, ou étain « liquide », qui n'est autre que des micro billes d'étain dans un gel).

Vous avez tout ? Bien. Pas d'erreur de miroir sur les pattes des composants ? Parfait ! (Et oui, c'est du vécu).

Mais avant de vous lancer, un dernier préparatif : vérifiez que vous avez accès à un schéma de votre carte, pour pouvoir placer les composants au bon endroit une fois l'étain déposé sur le circuit.

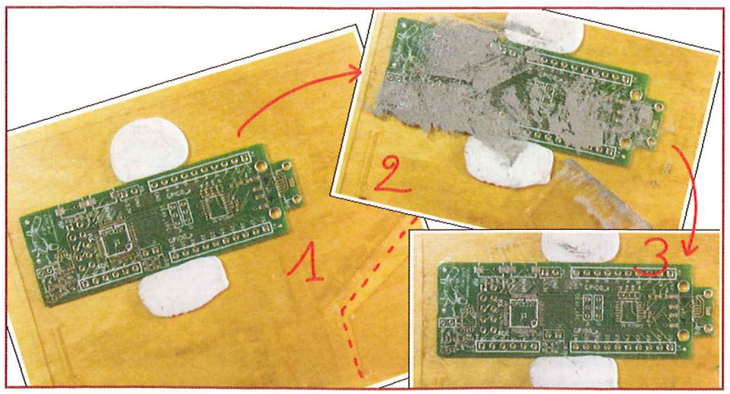


Figure 8 : Application de l'étain avec le pochoir plastique

Note : pour ceux qu'un texte et quelques photos ne sauraient contenter, j'ai fait une vidéo de l'assemblage du module GPIO Demo, voir le lien en fin d'article.

Vous pouvez enfin attaquer les choses sérieuses, et dans une petite heure (ou moins, parce que vous êtes très fort) vous aurez votre nouveau jouet rien qu'à vous.

La première étape, dégraisser le circuit imprimé, pour que le gel adhère bien. Frottez le légèrement avec un chiffon non abrasif et un peu d'alcool, sans insister (normalement les circuits arrivent relativement propres, mais nos doigts ne le sont pas toujours).

Ensuite, le plus simple pour fixer le circuit pour l'application de l'étain avec le pochoir (ou tout autre méthode en fait) c'est la « Patafix » (ou tout équivalent, je n'ai pas d'actions non plus). Cette matière semi-collante permet de maintenir en même temps le circuit imprimé et le pochoir CMS que vous allez devoir aligner correctement. Pour cette opération, le pochoir en plastique transparent offre un très net avantage sur son homologue en métal opaque, mais on s'en sort dans tous les cas. Veillez à ce que la Patafix n'empêche pas le pochoir d'être parfaitement plaqué au circuit imprimé.

Déposez un (tout petit) peu d'étain sur le pochoir (si vous avez de l'étain en seringue c'est plus pratique), et étalez le avec une spatule (j'utilise un morceau de pochoir métallique découpé dans un vieux pochoir inutile, et j'ai mis un morceau « prédécoupé » sur les pochoirs en plastique du kit du module GPIO Demo, mais n'importe quel morceau de plastique avec un bord droit fait l'affaire – une CB par exemple, comme pour la p... heu non, j'ai rien dit). Appuyez bien sur le pochoir pour qu'il reste plaqué au circuit pendant toute l'opération, mais sans le faire bouger. Remplissez bien tous les trous, puis raclez l'étain qui est en trop, pour qu'il ne reste que la bonne quantité d'étain dans les trous.

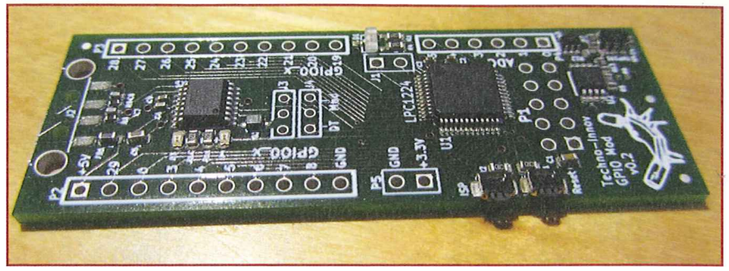


Figure 9 : Composants CMS placés sur le module GPIO-Demo avant soudure

Vous pouvez maintenant passer au plus délicat, retirer le pochoir sans le faire bouger, et en laissant l'étain sur le circuit imprimé. Ne paniquez pas, vous pourrez recommencer plusieurs fois si besoin. J'ai constaté que pour bien laisser tout l'étain sur le circuit il faut retirer le pochoir en le pliant, comme s'il s'enroulait autour d'un rouleau.

Vérifiez bien qu'il ne reste pas (ou presque) d'étain dans les trous du pochoir, et si vous êtes satisfaits du résultat vous pouvez passer à la suite. Vérifiez aussi que l'étain ne se soit pas glissé sous le pochoir pendant l'application pour former de gros pâtés. Sinon, nettoyez le pochoir (sommairement) et recommencez à l'étape ... un peu plus haut quoi.

6.2 Placer les composants

Il ne vous reste que quelques étapes, mais celle-ci est peut-être bien la plus minutieuse, et la plus longue.

Vous allez maintenant pouvoir placer les composants CMS sur votre circuit. Attention, ne placez pas les composants traversants, ils gêneraient pour l'étape suivante.

Personnellement je préfère commencer par les petits composants, car ils collent au gel quand celui-ci est bien frais, et ne bougent plus quand on tourne ou déplace le circuit imprimé. Pour ce faire, j'utilise une pince brucelle avec une pointe très fine, mais une simple pince à épiler peut faire l'affaire.

Une fois le composant positionné correctement, c'est à dire avec toutes les pattes ou les extrémités au dessus d'un plot de soudure, j'applique une petite pression sur le composant pour qu'il colle bien au gel et ne bouge plus par la suite. Cela évite notamment les composants qui se déplacent pendant que l'étain fond et se retrouvent en l'air, soudés par une seule patte ou d'un seul côté.

Attention à ne sortir qu'une référence de composants à la fois, ils sont tellement petits qu'il est difficile de lire les références sur les composants, ou même n'ont pas de références imprimées du tout (les condensateurs et les inductances par exemple).

Attention aussi au sens pour les composants polarisés (diodes, leds, certains condensateurs) ou avec plus de deux pattes. Il y a le plus souvent un indicateur, parfois sous le composant (pour les leds), parfois difficile à détecter, et il faut alors se référer à la documentation technique (ou utiliser une loupe, ou les deux).

Continuez avec les références suivantes, pour finir par les plus gros composants, que je place en dernier pour qu'ils ne gênent pas pour la pose des autres composants, mais aussi parce qu'une fois que le gel a un peu séché il est possible de bouger le composant sans déplacer l'étain, et de ne le « faire tenir » par une petite pression que lorsque le positionnement est bon.

Allez, plus que quelques composants à placer, courage ! (Oui, on a créé des machines pour ça, mais elles ne sont pas dans nos budgets).

6.3 Soudure CMS

Passons aux choses sérieuses : sortez la plancha !

Pour souder des composants CMS il existe différentes méthodes, mais si l'on en croit les tests que l'équipe de Sparkfun a réalisé le plus efficace c'est la plancha (ou toute surface chaude équivalente : poêle électrique, machine à crêpes, ou même une vieille poêle sur le gaz (attention aux problèmes conjugaux ...)).

Cette étape est la plus simple : on pose le circuit imprimé sur la plancha (composants sur le dessus bien entendu), on la branche, et on attend. C'est aussi la plus dangereuse : la plancha va atteindre plus de 250° C ! Gare aux brûlures, à ne faire qu'accompagné d'un adulte (responsable ? ça existe ?). En fonction de la puissance de la plancha, cela dure entre 2 et 5 minutes, et l'étain se met à fondre, soudant les composants. On voit très bien quand l'étain fond car il devient brillant. Et Denis a raison (toujours) : ne respirez pas les vapeurs qui se dégagent, tant du PCB que du flux contenu dans l'étain, et ventilez votre espace de travail.

Une fois que le processus a commencé, cela va très vite, et il ne faut pas insister si quelques points ne veulent pas fondre tout de suite, car les autres composants sont entrain de cuire. Il faut retirer la carte de la plancha tout de suite, en faisant bien attention à la garder à l'horizontal car ils ne sont plus tenus en place par le gel. Si vous inclinez la carte à ce moment là, les composants vont glisser (Il faut entre 4 et 10 secondes à l'étain pour solidifier). Pour cela les plancha sont bien pratiques car elles ont généralement une zone sans rebord, ce qui permet de faire glisser la carte vers le bord et de la saisir avec une pince avant de la déposer sur une surface protégée (surtout pas sur du plastique, elle est autour de 200° C !)

Ça y est, c'est fait, c'est soudé ! Enfin ...

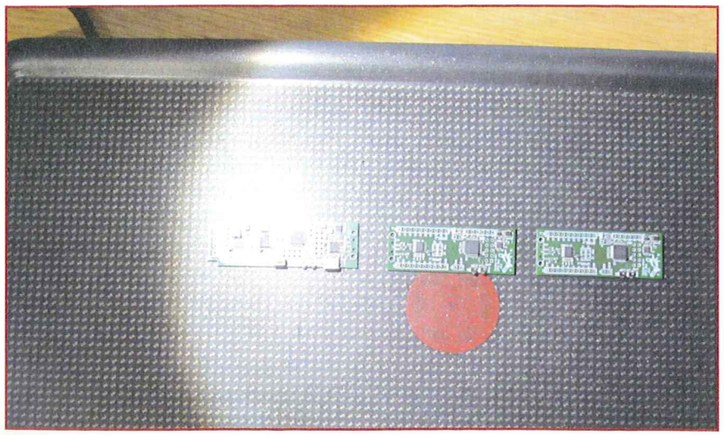


Figure 10 : Soudure à la plancha

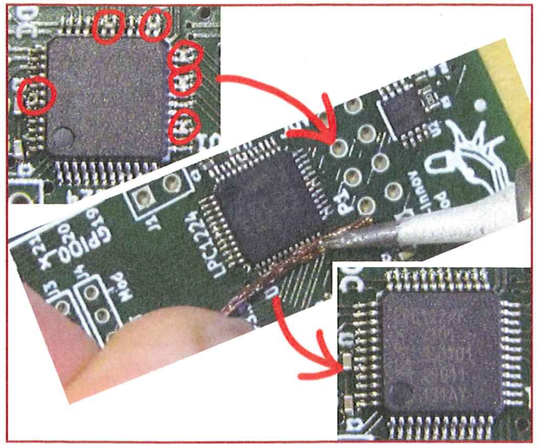


Figure 11 : Reprise à la tresse à dessouder

6.4 Reprise des faux contacts

Bon, en théorie, tout c'est bien passé et il n'y a pas de faux contacts. Malheureusement, même avec un pochoir CMS métallique certaines pattes de circuits intégrés peuvent être reliées par de l'étain s'il y en avait un petit peu trop.

Dans ce cas là, pas d'affolement, la première chose à faire : ne surtout pas prendre le fer à souder ou remettre la carte sur la plancha. La première étape est de reprendre le schéma de la carte.

Et oui, j'ai parfois passé près d'une demie heure à tenter d'enlever la soudure qui faisait contact entre deux pattes ... reliées par un fil sur le schéma.

Si effectivement le schéma dit qu'il ne devrait pas y avoir de contact, utilisez la tresse à dessouder et le fer pour enlever le surplus d'étain, en mettant un morceau propre de la tresse à dessouder entre l'étain à enlever et le fer à souder. Testez avec le multimètre pour vérifier que le court-circuit a bien disparu.

Il faut aussi finir de souder les composants des bords de la carte qui n'auraient pas été soudés sur la plancha. Pour ceux-ci, pas besoin de rajouter de soudure, il suffit de poser le fer à souder (chaud) très proche de la pastille (ou sur la pastille) pour que l'étain fonde et soude le composant.

Profitez-en pour vérifier que tous les composants sont bien soudés des deux côtés.

Attention, vous pourrez avoir l'impression que certains composants sont mal soudés ou qu'il n'y a pas assez de soudure car la quantité de soudure utilisée avec cette méthode est très faible et que la soudure ne forme pas de pâté. C'est normal. Pour vous rassurer, vous pouvez utiliser la même méthode que pour finir de souder les composants dont l'étain n'avait pas fondu à la plancha. Vous verrez alors l'étain briller un peu plus, et re-solidifier lorsque vous retirez le fer.

6.5 Soudure des composants traversants

Avant-dernière étape avant de jouer avec votre circuit : souder les composants traversants, et éventuellement ajouter de la soudure sur les pads de fixation des gros composants et des connecteurs.

Pour cela rien de magique (quoique, voir l'astuce qui suit). Mettez en place le composant (attention au brochage s'il y en a un), positionnez votre fer à souder en contact avec le circuit imprimé et la patte du composant que vous voulez souder, puis apportez la soudure, et retirez la soudure et le fer (dans cet ordre).

Petite astuce : utilisez de la Patafix pour maintenir le composant en place et pour maintenir le circuit sur le plan de travail. Depuis que j'utilise cette technique ma troisième main sert de support aux toiles d'araignées...

7 Tests

Dernière étape avant de passer à la programmation, qui fera l'objet du prochain article : les tests électriques.

Ne mettez pas votre circuit sous tension tout de suite, certains composants ne tiennent pas longtemps en cas de court circuit, même alimentés par l'USB.

Utilisez le multimètre pour vérifier qu'il n'y a pas de court circuit entre le +5V et la masse, entre le +3V et la masse, et pour ne pas risquer d'endommager les ports USB de votre PC/HUB (certains sont très fragiles), vérifiez aussi entre les deux pistes USB (D+ et D-) et la masse et le +5V.

Si vous n'êtes pas trop impatient, vous pouvez aussi tester entre chaque pattes consécutives des circuits intégrés, mais vérifiez bien le schéma en même temps, toujours pour éviter de passer des heures à vouloir enlever un court-circuit qui n'en est pas un.

Tout est bon ? et bien c'est fini pour cette fois.

Si vous avez réalisé le circuit du module GPIO Demo, vous pouvez le connecter sur un port USB de votre PC.

Le microcontrôleur doit se mettre automatiquement en mode programmation (ISP) et la led bicolore doit faire apparaître deux points lumineux très très faibles (vert et rouge).

Si votre noyau a bien le support FTDI (suffisamment récent pour avoir le composant FT230XS, plus récent que 2.6.28 de mémoire), vous aurez alors un nouveau `ttyUSB` (présent dans `/dev`, et annoncé dans les logs du noyau), et en envoyant le caractère `'?'` le microcontrôleur devrait répondre « `Synchronised` ».

Rendez-vous dans l'article suivant pour l'écriture du code en C, la compilation, et la programmation avec l'utilitaire lpcprog :) ■

Liens

FabLabs :

- ElectroLab de Nantère : http://www.electrolab.fr/
- Fabrique d'Objets Libres (FabLab de Lyon) : http://www.fablab-lyon.fr/ et http://wiki.fablab-lyon.fr/

Informations sur la fabrication de circuits imprimés :

- http://alain.canduro.free.fr/circuits.htm
- http://electroremy.free.fr/elec-info-ci.html

Sparkfun :

- https://www.sparkfun.com/tutorials/59

Fabricants de circuits imprimés :

- https://www.pcb-pool.com/
- http://www.dipole-electronique.fr/
- http://www.seeedstudio.com/service/

Module GPIO Demo :

- Page du module sur notre site : http://www.techno-innov.fr/technique-module-gpio-demo/
- Kit du module dans la boutique : http://boutique.techno-innov.fr/module-gpio-demo-26.html
- Vidéo de l'assemblage du module sur Youtube : http://urlalacon.com/A78Q53

ESP8266 ET MODULE WI07C : AJOUTEZ DU WIFI ÉCONOMIQUE À VOS PROJETS (OU PAS)

par Denis BODOR

Ajouter une connectivité Wifi et/ou Internet à un projet ou un matériel existant n'est pas chose facile. En particulier si les ressources disponibles ou déjà en place réduisent grandement les options. La prise en charge du Wifi rime invariablement avec implémentation d'une pile TCP/IP et de mécanismes avancés d'authentification. Il faut donc généralement trouver une solution intégrant tout ce que le projet de départ n'est pas en mesure de réaliser lui-même, ce qui implique souvent des solutions plus coûteuses que la plateforme à connecter elle-même. Mais à l'heure de l'IoT, une solution économique semble se profiler à l'horizon...

Il y a des jours comme ça où on tombe par hasard sur des composants qui changent la vie. C'est ainsi qu'en visionnant une vidéo Youtube de CNLohr, j'ai appris l'existence d'un petit module très intéressant, distribué sur le site d'Electrodragon sous le nom Wi07c et vendu un peu plus de 3,50 euros. Non, il ne s'agit pas d'une faute de frappe, j'ai bien écrit trois euros et cinquante centimes. Bien entendu, la documentation et le firmware embarqué sont en rapport direct avec ce prix étonnant mais l'ensemble forme quelque chose de prometteur. Mais je vous laisse le soin de constater et juger par vous-même.

Le module en question est construit autour d'un SoC ESP8266 d'*Espressif Systems*. Celui-ci comprend un processeur 32 bits Xtensa, un peu de SRAM, quelques interfaces (SDIO, SPI, i2c, GPIO) et surtout intègre une interface Wifi 802.11 b/g/n. Le circuit du module regroupe ainsi ce SoC, une EEPROM SPI, un quartz et une poignée de composants passifs. Côté logiciel, le SoC dispose de sa propre pile TCP/IP et du support CCMP, TKIP, WEP, WPA, WPA2 et WPS avec accélérateur matériel ainsi qu'une gestion relativement avancée de la gestion d'alimentation.

Le module est vendu pré-chargé avec un firmware offrant une interface série (3,3V) permettant de paramétrer le mode de fonctionnement, la configuration Wifi et la partie TCP/IP. Ce firmware, stocké dans l'EEPROM SPI, peut être mis à jour via l'interface série grâce à un mode particulier défini au démarrage ou directement via une connexion Internet (firmwares récents).

La connectivité Wifi et TCP/IP offerte par ce module est à la fois très riche et très pauvre. En effet, nous avons d'une part un SoC dédié capable de prendre en charge un comportement client (STA) vis-à-vis du Wifi mais aussi d'agir comme un point d'accès (AP), l'ensemble avec différents niveaux de chiffrement/authentification : *open*, WEP, WPA_PSK, WPA2_PSK, WPA_WPA2_PSK. Ajoutons à cela que la pile TCP/IP peut être configurée de manière à agir comme un client ou comme un serveur.

Mais ces fonctionnalités sont limitées par ailleurs car l'interface reste textuelle et « unique » dans le sens où la communication série avec le module devra gérer à la fois les directives de configuration (sous forme de commandes AT) et les données. Ajoutez à cela que la taille des paquets de données est également limitée puisque qu'en réception, du moins, on dispose d'un maximum de 1410 caractères.

En cas de dépassement, les données excédentaires sont traitées comme un autre envoi. Il ne sera donc pas raisonnable d'envisager d'utiliser ce module (avec le firmware par défaut du moins) pour échanger des données binaires comme des images, par exemple, du moins pas sans encodage et fragmentation.

Enfin, le problème le plus important en terme de limitation est le manque de documentation fiable à la fois sur le SoC ESP8266, le module et le firmware. Ce à quoi s'ajoute les traductions approximatives non seulement sur les sites des constructeurs mais aussi dans le firmware lui-même. En guise d'exemple, voici un petit avant-goût :

```
AT+CIPSEND?
no this fun

AT+CIPSEND=0,5
link is not
```

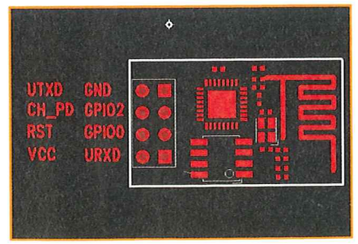
Limpide, n'est-ce pas ? Ainsi les explications incluses dans cet article sont issues de la documentation disponible, des commentaires des utilisateurs s'étant déjà frottés à ce matériel et de nos propres expérimentations, le tout vérifié avec les exemplaires du module que nous avons acquis directement auprès d'Electrodragon.

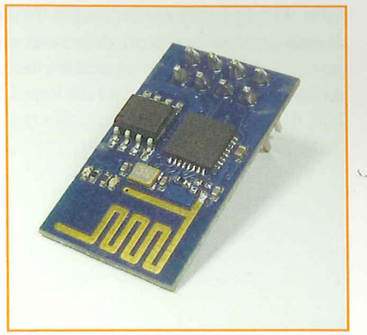
Notez qu'il existe deux versions du module : une ancienne dite « V080 » ne proposant que le brochage pour l'alimentation et TX/RX, et une actuelle « V090 » ajoutant des GPIO et une broche CH_PD destinée à l'activation. Nous ne traiterons pas ici de l'ancienne version, normalement introuvable à présent mais tantôt décrite dans certains tutoriels obsolètes (comme le wiki SeeedStudio).

1 Connexion physique

Le module se présente sous la forme d'un PCB de 24 mm par 14 mm équipé d'une série de 2×4 connecteurs au pas de 2,54mm. La nomenclature des broches est la suivante :

- VCC : alimentation en 3,3 volts avec un courant pouvant aller jusqu'à 240 mA en crête. Ceci implique une alimentation capable de fournir le courant nécessaire selon l'usage du module, ce qui n'est généralement pas le cas, par exemple d'un adaptateur USB/série type FT232RL ou PL2303.
- GND : la masse.
- UTXD : émission des données du module à relier à RXD sur votre montage ou adaptateur USB/série. Attention, ces lignes comme toutes celles du module sont en 3,3V, non tolérantes au 5V.
- URXD : réception des données.
- RST : *reset* du module par mise à la masse de cette ligne. Il peut être intéressant de contrôler cette ligne par l'intermédiaire d'une GPIO sur votre montage, en cas de problème ou de rupture de communication.
- CH_PD : c'est le *chip enable* permettant d'activer le module. Il est nécessaire de relier cette ligne à VCC. Là encore, il peut être intéressant de contrôler cette broche logiciellement pour activer/désactiver le module.
- GPIO0 : configurée en entrée, cette ligne reliée à la masse permettra de flasher le firmware du SoC pour mise à jour (cf, plus loin dans l'article).
- GPIO2 : semble être une GPIO disponible en cas de développement d'un nouveau firmware pour l'ESP8266 (des projets existent mais rien n'a pour l'instant dépassé le stade de la spéculation du fait d'un manque drastique de documentation et d'un SDK open source).





La connexion simple pour expérimentation et prise en main se résume à interconnexion UTXD et URXD avec les lignes RXD et TXD d'un adaptateur USB FT232RL par exemple. On prendra soin d'alimenter le module avec une source externe 3,3V capable de fournir un courant stable et suffisant. Enfin, on n'oubliera pas de relier CH_PD au VCC pour activer le module. Notez que certaines documentations ne font pas mention de cette connexion car elles référencent l'ancienne version du module qui se limitait aux broches VCC, GND, UXTD et URXD.

Dès la connexion de l'alimentation, une led rouge doit s'activer. La connexion de CH_PD fait brièvement clignoter la led bleu et, si votre adaptateur USB/série dispose d'indicateurs à led, vous constaterez qu'un *burst* de données est envoyé par le module. Il s'agit des informations de débogage (boot) dont l'envoi se termine par **ready**.

En phase de test comme une fois le module utilisé en situation, il est important de bien comprendre que l'attente de la chaîne de caractères **ready** est impérative avant d'envoyer des commandes AT. De plus, durant nos essais, il s'est avéré que le module, même en l'absence apparent de problème de stabilité d'alimentation, redémarrait de lui-même sans prévenir (mais nous ne pouvons être catégorique sur la cause du problème). Dans un mode où le module est utilisé côté client, pour se connecter en Wifi et initier une communication avec un serveur distant, ceci ne pose pas tant de problèmes puisqu'il

suffit d'intégrer la réinitialisation du module à la procédure d'envoi des données. En mode serveur en revanche, les choses sont très différentes puisque si le client, sur PC par exemple, veut se connecter au module sur lequel fonctionne le serveur TCP/IP, il faut que celui-ci soit effectivement démarré. Or, en cas de redémarrage intempestif, ce ne sera pas le cas. Pour contourner le problème, il faudra que le programme pilotant le module surveille l'apparition de la chaîne **ready** pour relancer le serveur TCP/IP le cas échéant. Ce n'est pas très propre mais rappelons que le Wi07c n'est pas un modèle de conception industriel et ne doit donc pas être utilisé en tant que tel.

2 Mise à jour

Une fois la connexion faite, l'adaptateur USB branché et le module alimenté, il vous suffira d'utiliser un quelconque émulateur de terminal série comme **screen** ou **minicom** sous GNU/Linux, ou encore CoolTerm pour Windows ou Mac OS X. La communication se fera normalement en 115200 8N1 (ou en 57600 selon la documentation mais nous n'avons pas pu vérifier ce point).

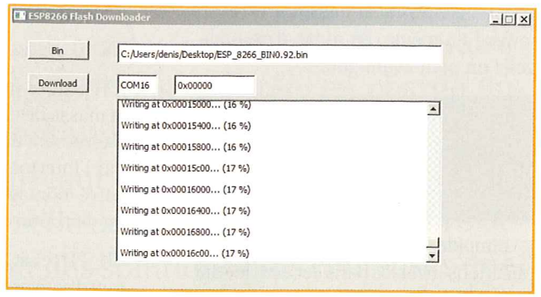
La vérification de la communication se fera très simplement en envoyant la commande **AT** suivie d'un CRLF (*Carriage Return + Line Feed*, soit **\015\012**) ou, selon la version du firmware un simple LF (**\012**). Vous êtes censés obtenir un écho de ce que vous envoyez au module. Si la réponse est **OK** on enchaînera sur une interrogation de la version du firmware en place :

```
AT

OK

AM+GMR
00150900

OK
```

Ce chiffre se décompose en **0015** qui est la version du SDK et **0900** (V090) qui est celle du firmware lui-même. Il est plus que probable que ces versions soient celles que vous allez voir apparaître après avoir validé la commande. La dernière version en date est **0018000902** (V092) et à partir de la version V091 la mise à jour se passe via le *cloud* (ce n'est pas moi qui le dis, c'est la documentation). En réalité, le module peut tout simplement se mettre à jour s'il dispose d'un accès Internet après s'être connecté à un routeur Wifi (Internet = *cloud*, en somme). Le problème qui se pose alors est celui de la mise à jour V090 à V091. Notez au passage ceci n'est pas optionnel car un grand nombre de bugs ont été corrigés depuis la V090 et en particulier l'un d'entre eux pouvant bloquer le module et le rendre incapable d'accepter la moindre commande (le « *busy bug* »).

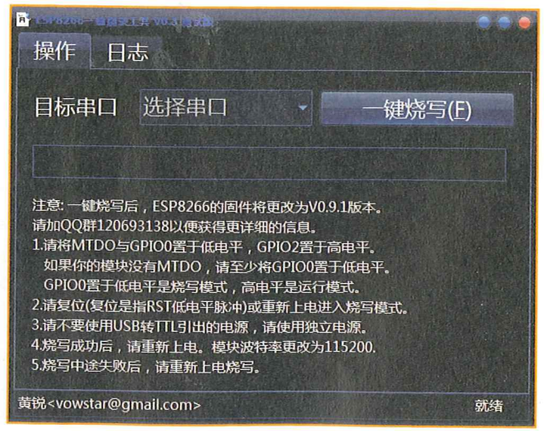
Pour mettre à jour, vous serez dans l'obligation d'utiliser un outil binaire Windows mis à disposition par le constructeur. Pointez votre navigateur sur https://www.amazon.com/clouddrive/share/oghB7_GosqMIPiyi3Lu2oZj8OQNB2JFy8sy9Azd6cA0 ou via le *post* sur le blog Electrodragon, http://blog.electrodragon.com/cloud-updating-your-wi07c-esp8266-now/. Le fichier Zip que vous pourrez récupérer s'appel **Could update ESP8266.zip** (je déteste les noms de fichier avec des espaces presque autant que les typos du genre « cloud »/« could »). Celui-ci contient un exécutable **esp8266_flasher.exe** et un fichier binaire qui est le firmware en version V091 : **ESP_8266_BIN0.92.bin**.

Après avoir connecter votre Wi07c à Windows via l'adaptateur USB/série dont il faudra très certainement installer les pilotes, vous devez disposer d'un nouveau port série (COM*). Assurez-vous que CH_PD est à VCC et GPIO0 à la masse et lancez l'outil. Spécifiez alors le port série adéquate ainsi que le fichier binaire du firmware et cliquez simplement sur le bouton « *Download* » pour entamer la mise à jour.



Les quelques 512 ko de firmware seront flashés dans l'ESP8266 et l'opération devrait se terminer par un message précisant que le module n'a put être redémarré en mode standard. Retirez alors GPIO0 de la masse et redémarrez le module en coupant brièvement l'alimentation ou en mettant un instant RST à la masse.

Notez qu'un autre outil de mise à jour existe et est référencé sur le wiki d'Electrodragon (http://www.electrodragon.com/w/Wi07c). La capture présentée ici justifie à elle seule le fait que j'ai préféré ne pas m'en servir :



Après redémarrage la commande **AT+GMR** doit vous retourner **00170901**. A ce stade, le firmware accepte toujours les lignes validées par un simple **\n** (LF), mais ça ne durera pas. En effet, nous disposons maintenant d'une version V091 incluant la fonction de *cloud update* (ou *could update* on sait pas trop) et nous pouvons donc relier notre module à Internet. Pour cela, ce dernier doit être configuré de manière à agir comme une « station » Wifi. Ceci se règle via la commande **AT+CWMODE=** suivie d'une valeur pouvant être 1 pour STA, 2 pour AP et 3 pour STA+AP. Avec cette version du firmware il est nécessaire, après avoir changé de mode, de redémarrer le SoC. Nous faisons donc :

```
AT+CWMODE=1

OK
AT+RST
ets Jan  8 2013,rst cause:4, boot mode:(3,6)

wdt reset
load 0x40100000, len 212, room 16
tail 4
chksum 0x5e
load 0x3ffe8000, len 788, room 4
tail 0
chksum 0x1c
load 0x3ffe8314, len 72, room 8
tail 0
chksum 0x55
csum 0x55
jump to user1

ready
```

Le **AT+RST** est, bien entendu, la commande pour le *reset* et notez que nous sommes gratifiés d'informations intéressantes sur le fonctionnement du SoC, les adresses utilisées et le découpage entre ce qui semble être un RTOS et l'espace utilisateur. Si vous n'êtes pas coutumier des commandes AT utilisées dans les modems et d'autres modules comme le HC05 en Bluetooth (voir Open Silicium 5), voici les règles du jeu :

- **AT+XXXX** : commande sans argument comme **AT+RST**,
- **AT+XXXX=valeur** : définition d'une valeur, comme **AT+CWMODE=1**,
- **AT+XXXX=?** : consultation des valeurs possibles, **AT+CWMODE=?** retourne **+CWMODE:(1-3)**,
- **AT+XXXX?** : lecture de la valeur actuelle. Après changement de mode et *reset*, **AT+CWMODE?** retourne **+CWMODE:1**.

Bien entendu, toutes les commandes ne répondent pas forcément à toutes les syntaxes et, dans le cas présent, si vous cherchez à consulter la valeur pour une commande qui n'en stocke pas vous aurez droit à la réponse très explicite **no this fun** (?!). Ce qui est passé en arguments peut prendre la forme d'une option unique ou d'un ensemble de paramètres séparés par des virgules. Ainsi, comme nous souhaitons nous connecter à un point d'accès, nous utiliserons :

```
AT+CWJAP="monAP","phrase2passe"

OK
```

Si tout se passe bien (authentification, et DHCP), le module doit obtenir une adresse IP ainsi que tous les éléments permettant la connexion au net (DNS, route, etc). L'adresse obtenue peut être connue affichée par le module mais pas les autres informations :

```
AT+CIFSR
192.168.10.124
```

En cas de problème du type échec de l'authentification et/ou non attribution d'adresse, la commande retourne **ERROR** (notez qu'en tapant une commande qui n'existe pas, ce sera **Error**, la diversité c'est important). Dans le doute, vous pouvez utiliser la commande permettant de lister les points d'accès à porté :

```
AT+CWLAP
+CWLAP:(0,"",0,18:17:1f:1f:10:10,0)
+CWLAP:(2,"monAP",-61,10:11:c1:11:11:11,5)
+CWLAP:(0,"BouWiFi",-92,11:17:1e:11:91:10,13)

OK
```

Remarquez la présence non systématique des réponses **OK**, ils sont joueurs ces développeurs chinois... Quoi qu'il en soit, on voit s'afficher les modes de chiffrement, les SSID, le RSSI (*Received Signal Strength Indication*), l'adresse MAC de l'AP et le canal utilisé.

Si votre serveur DHCP fait bien son travail, vous avez maintenant accès au net et vous pouvez procéder à la mise à jour via la commande **AT+CIUPDATE**

```
AT+CIUPDATE
+CIUPDATE:1
+CIUPDATE:2
+CIUPDATE:3
+CIUPDATE:4

OK

ets Jan  8 2013,rst cause:4, boot mode:(3,6)

wdt reset
load 0x40100000, len 212, room 16
tail 4
chksum 0x5e
load 0x3ffe8000, len 788, room 4
tail 0
chksum 0x1c
load 0x3ffe8314, len 72, room 8
tail 0
chksum 0x55
csum 0x55
jump to user2

ready
```

Oui, la commande est **AT+CIUPDATE** et les messages **CIPUPDATE**... Comme nous sommes aussi joueurs que les développeurs chinois, nous avons surveillé l'opération et avons remarqué que le firmware était téléchargé en HTTP depuis http://114.215.177.97 (ou http://iot.espressif.cn/). En fouillant encore un peu, il s'avère qu'il semble exister tout un monde derrière ce serveur et le nom IotBucket puisque l'URL http://iot.espressif.cn/#/api/ détaille toute une API permettant d'utiliser ce qui semble être un ensemble de Web-Services relativement dense (mais mon niveau de mandarin est bien insuffisant pour tirer des conclusions valables). Selon toute vraisemblance, la disponibilité du module Wi07c n'est qu'un effet de bord d'un déploiement d'une infrastructure de collecte de données IoT du type « Sen.Se ». Je pense que ceci mériterait de plus amples recherches, sans doute par mes confrères du magazine Misc, ne serait-ce que pour savoir si ce module ne « parlerait » pas plus qu'il ne devrait le faire légitimement...

Concernant la procédure de mise à jour elle-même, les diverses mentions de

+CIPUPDATE: suivi d'un numéro résument la progression des opérations (dixit le *post* du blog cité précédemment) :

- 1 : serveur trouvé,
- 2 : connecté au serveur,
- 3 : version trouvée,
- 4 : « *start start* » (?), écriture en cours je suppose...

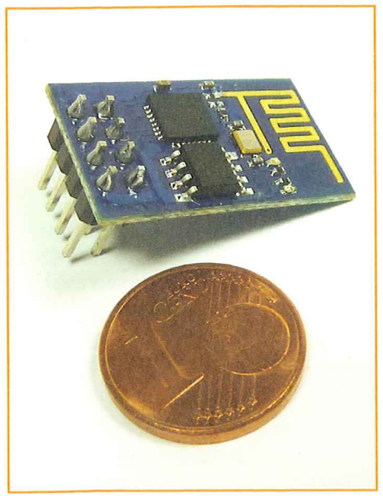
S'en suit un *reset* automatique et nous pouvons alors vérifier que :

```
AT+GMR
0018000902

OK
```

Nous avons la dernière version du firmware (V092) épuré du bug problématique de déconnexion menant à des messages **"busy"** en boucle. Notez que **\n** ne permet plus de valider les commandes. Désormais, c'est CRLF obligatoire. Petite astuce au passage, avec GNU Screen, ajoutez cette ligne dans votre **.screenrc** pour utiliser la touche F11 afin d'envoyer CRLF et valider les commandes AT :

```
bindkey -k F1 stuff "\015\012"
```

Nous avons, par ces manipulations, déjà bien entamé les explications de prise en main du module, mais il reste des choses à dire...



3 Le Wifi

Nous l'avons vu, se connecter à un point d'accès ne présente pas de difficulté particulière et nous permet de communiquer sur le net. Chose également très intéressante, la configuration (SSID, etc) perdure après une mise hors tension et même une mise à jour. Mais il est aussi possible, avec ce module, d'agir comme un point d'accès et donc d'offrir à un laptop, une tablette ou un smartphone une solution de contrôle indépendante d'un accès à un réseau existant. De quoi transformer n'importe quel objet de la lampe à la machine à café en passant par le lave-linge ou le simple capteur d'humidité, en entité IoT (*buzzword* permettant de faire sautiller un décideur sur son fauteuil).

Nous pouvons au choix agir en tant qu'AP (mode 2) ou en duo station+AP (mode 3). Comme nous l'avons vu, le choix du mode se fait via :

```
AT+CWMODE=3

OK

AT+CWMODE?
+CWMODE:3

OK
```

Remarquez qu'avec la dernière version du firmware, le redémarrage du module n'est plus nécessaire. En conservant les paramètres définis précédemment concernant la connexion Internet, on remarque que le module dispose maintenant de deux adresses IP :

```
AT+CIFSR
192.168.4.1
192.168.10.124
```

192.168.4.1 est l'adresse du point d'accès du module, qui n'est bien entendu spécifiée dans aucune documentation et ne peut être changée via les commandes AT. Ceci peut être problématique en cas de connexion à un réseau 192.168.4.0/24, le conflit d'adresses sera alors une certitude.

Il ne nous reste plus, à ce stade, qu'à configurer le point d'accès avec **AT+CWSAP** :

```
AT+CWSAP="totoAP","tralala4242",4,3

OK
AT+CWSAP?
+CWSAP:"totoAP","tralala4242",5,3

OK
```

Les arguments de la commande sont SSID, phrase de passe, canal Wifi et type d'authentification (pour rappel, 0=open, 1=WEP, 2=WPA, 3=WPA2, 4=WPA+WPA2). Dès lors, le point d'accès devrait apparaître pour n'importe quel périphérique à proximité. Nous avons cependant remarqué que la configuration, en cas de changement du mode de 1 en 3 par exemple puis définition des paramètres de l'AP, provoquait tantôt la création de deux AP dont un en mode *open* avec un SSID découlant d'une partie de l'adresse MAC du module. Un **AT+RST** règle le problème mais ce comportement peut être problématique. Surveillez vos arrières et vérifiez les actions du module systématiquement.

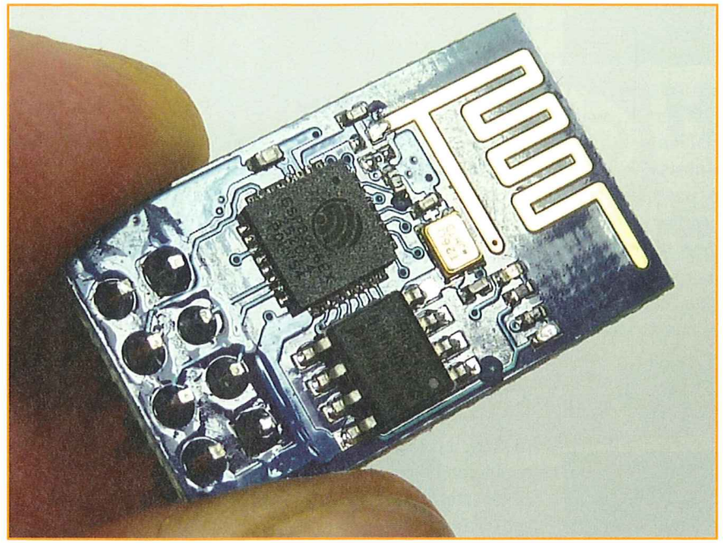
Les connexions des stations Wifi pourront être listées avec :

```
AT+CWLIF
192.168.4.100,a0:1b:ba:c9:07:c0
192.168.4.101,10:bf:48:c0:16:17

OK
```

Les adresses IP sont attribuées séquentiellement à partir de 192.168.4.100 par le serveur DHCP interne à l'ESP8266 mais côté module nous ne sommes malheureusement pas notifiés d'une nouvelle connexion. Il faut donc prévoir du *polling* pour éventuellement déclencher des actions en conséquence.

4 TCP/IP client et serveur

Une fois la connexion Wifi établie que ce soit en station ou en AP, il est possible de monter d'un cran dans les couches ISO pour entamer un dialogue



au niveau TCP/UDP aussi bien en tant que serveur que client. L'ensemble est totalement pris en charge par le firmware du module et nous ne pouvons agir qu'au niveau de la couche application.

Mais avant toutes choses, il faut choisir si nous souhaitons gérer ou non de multiples connexions. Notez que pour agir en temps que client nous pouvons choisir l'une et l'autre option mais qu'en tant que serveur nous n'avons pas le choix : il est impératif d'autoriser le multiplexage des connexions sinon la commande de création de serveur retourne tout simplement **ERROR** sans plus d'explications. Le choix du multiplexage se fait via **AT+CIPMUX=0** (mono-connexion) ou **AT+CIPMUX=1** (multi-connexions).

On peut entamer ensuite le dialogue avec un serveur. Pour l'occasion, nous utilisons Netcat (**nc**) en lançant simplement :

```
% nc -l -p 4444
```

Netcat est alors en mode *listen* sur le port 4444 de toutes les interfaces du PC. On se tourne alors vers l'interface série du module :

```
AT+CIPMUX=1

OK
AT+CIPSTART=0,"TCP","192.168.10.166",4444

OK
Linked
```

La syntaxe des arguments de **AT+CIPSTART** dépend de la valeur de **AT+CIPMUX**. En multi-connexion, on précise l'ID du « canal » entre 0 et 4, le protocole TCP ou UDP, l'adresse cible du serveur et le port. En mono-connexion, on se passera simplement de préciser le canal. En cas d'utilisation de l'ID en mono-connexion le message **Link typ ERROR** est retourné. Notez que protocole et adresse IP sont entre guillemets mais pas les données numériques comme l'ID et le port. Dès la commande validée (CRLF) le mot **Linked** apparaît nous signifiant que la connexion est établie. Chose qu'on pourra vérifier avec :

```
AT+CIPSTATUS
STATUS:3
+CIPSTATUS:0,"TCP","192.168.10.166",4444,0

OK
```

Une valeur de *timeout* est utilisée pour terminer la connexion en cas d'absence d'activité. Vous pouvez connaître la valeur en secondes avec **AT+CIPSTO?** et en définir une nouvelle via **AT+CIPSTO=** suivi du nombre de secondes souhaités. Vous pouvez couper manuellement la connexion avec **AT+CIPCLOSE** seul en mono-connexion. Mais en multi-connexion, il vous faudra un argument, l'ID de la connexion à fermer :

```
AT+CIPCLOSE
MUX=1
AT+CIPCLOSE=0

OK
Unlink
```

Notez que **AT+CIPCLOSE** ne retourne pas une erreur mais **MUX=1** précisant la valeur de **AT+CIPMUX**, ce qui est loin d'être cohérent (gardons toutefois un chiffre à l'esprit : 3,50€).

L'opération inverse consistant à faire fonctionner **nc** en guise de client pour le module en serveur, n'est guère plus complexe :

```
AT+CIPMUX=1

OK
AT+CIPSERVER=1,5555

OK
```

Le premier argument active le serveur et le second précise le port à écouter. Un serveur est alors accessible avec un simple **nc adresse port** et on voit alors apparaître **Link**/**Unlink** à la connexion/déconnexion. Notez que le serveur est à l'écoute sur le port à la fois en TCP et en UDP (**nc -u adresse port**) mais que, bien entendu, dans ce cas il n'y a pas de connexion/déconnexion. Il est possible de stopper le serveur en utilisant :

```
AT+CIPSERVER=0
we must restart
```

Comme le précise le message, il faut alors utiliser **AT+RST** pour redémarrer le module. Les connexions seront, avant redémarrage, toujours acceptées et signalées par **Link** mais la déconnexion ne retournera aucun message. Si une connexion existe, elle ne sera pas coupée, pas même lors du *reset*. Notez également que le fait de spécifier une autre commande **AT+CIPSERVER=1,** suivie d'un port identique ou différent affichera **"no change"** et le serveur sera toujours à l'écoute sur le port initialement spécifié (**no change** doit vouloir dire « nan, je change pas » et non « pas de changement à appliquer »).

Une fois une connexion établie, quelque soit le sens, les messages envoyés par **nc** apparaîtront ainsi :

```
+IPD,0,19:coucou le monde :)

OK
```

Le message précise **+IPD**, l'ID de la connexion, le nombre de caractères (LF, ou CRLF inclus) et la chaîne reçue. Si **AT+CIPMUX=0** alors le champ précisant l'ID de la connexion est absent.

L'envoi de messages depuis le module se fera en deux temps. Il faut, tout d'abord, utiliser la commande **AT+CIPSEND** accompagnée en argument de l'ID de la connexion (si **AT-CIPMUX=1**) et du nombre d'octets à envoyer. La validation de la commande affiche un *prompt* (**>**) permettant d'entrer le message qui ne nécessite pas de validation (le compte juste de caractères permet de reprendre la main). Ce qui nous donne en résumé :

```
AT+CIPMUX=0

OK
AT+CIPSTART="TCP","192.168.10.166",4444

OK
Linked
AT+CIPSEND=6
> coucou
SEND OK
AT+CIPCLOSE

OK
Unlink
AT+CIPMUX=1

OK
AT+CIPSTART=0,"TCP","192.168.10.166",4444

OK
Linked
AT+CIPSEND=0,5
> hello
SEND OK
AT+CIPCLOSE=0

OK
Unlink
```

Conclusion

Que penser finalement de ce module très économique ? Il apparaît clairement qu'il n'est pas question d'intégrer celui-ci dans l'état dans un quelconque produit final et la question de savoir s'il faut ou non l'ajouter à un projet existant dépendra complètement de l'aspect critique de la réalisation. Le firmware est bien entendu la cause de tous les problèmes avec un manque de documentation évident, des comportements et messages incohérents, des traductions approximatives, etc.

Pourtant le potentiel est là et le SoC lui-même ne semble pas être source d'inquiétude. La meilleure chose qui pourrait arriver dans la vie de ce module est clairement une mise à disposition des sources du firmware sous une licence permettant son développement communautaire. A mesure que les ventes se multiplient sur la toile, la communauté d'utilisateurs grossie et des initiatives apparaissent (comme le forum www.esp8266.com) essayant de collecter un maximum d'information pour, à terme, peut-être conduire à la disponibilité d'un firmware plus stable ou, disons-le franchement, tout simplement terminé.

On remarquera de plus que ce module est également distribué par *SeeedStudio* et que la description du produit s'accompagne d'un lien permettant de télécharger un SDK (**esp_iot_sdk_v0.6.zip**). La description précise aussi : « *We have a set of documents in Chinese[...] Please buy this module only when you understand the existing documents* », au moins c'est clair (et un tantinet discriminatif).

Le **Makefile** intégré dans le Zip fait cependant référence entre autre à **xt-xcc** qui un élément du compilateur pour la plateforme Xtensa de la société californienne Tensilica (à présent intégrée dans *Cadence Design Systems*, NASDAQ-100 svp). Un formulaire en ligne permet de demander une évaluation pour le SDK Windows intégrant les outils susceptibles de produire un firmware fonctionnel. Comme vous pouvez le voir, les étapes permettant de passer d'un firmware/SDK/compilateur propriétaire à une solution opensource qui pourrait développer pleinement le potentiel du SoC ESP8266 sont nombreuses. Il est presque futile d'espérer que ceci pourrait s'accomplir facilement sans l'intervention à la fois des américains de *Cadence Design Systems* et des chinois d'*Espressif Systems*. Dommage... ■