à destination de **arm-uclinuxeabi**. Ce n'est généralement pas une démarche que j'apprécie et j'ai toujours tendance à voir d'un mauvais oeil l'installation d'un binaire « étranger » sur un de mes systèmes. Cependant, ici il s'agit de gagner du temps et des ressources en reposant sur un OS connu, borné et documenté. Il serait dommage d'investir directement ce gain dans la construction/adaptation d'une chaîne de compilation plutôt que de l'utiliser pour développer une meilleure solution embarquée.

On se bornera donc à télécharger une archive ici :

- http://www.codesourcery.com/sgpp/lite/arm/portal/package6503/public/arm-uclinuxeabi/arm-2010q1-189-arm-uclinuxeabi-i686-pc-linux-gnu.tar.bz2

ou là :

- https://sourcery.mentor.com/public/gnu_toolchain/arm-uclinuxeabi/arm-2010q1-189-arm-uclinuxeabi-i686-pc-linux-gnu.tar.bz2

Celle-ci sera alors décompressée dans un emplacement comme **/opt** ou, plus judicieusement **$HOME/chemin** pour ensuite ajouter au **PATH** le chemin vers le répertoire **bin/** ainsi installé. Notez qu'il n'est pas nécessaire de faire cette modification de manière permanente, un simple **export PATH="$HOME/chemin:$PATH"** sera suffisant dans le terminal où vous procéderez au reste des manipulations.

Un dernier prérequis sera l'installation de l'outil **genromfs**, un **mkfs** pour le système de fichiers simple et en lecture seule ROMFS. Cet outil est utilisé par le système de construction afin de créer le rootfs qui sera copier dans la flash du STM32, à une position dont le noyau sera informé lors de son exécution.

2.2 Construction

La construction de l'ensemble du système se fait très simplement et très rapidement grâce au travail de Jim Huang (jserv) puisque cela consiste à récupérer ses éléments et se plier d'un simple **make** :

```
% git clone https://github.com/jserv/stm32f429-linux-builder.git
% cd stm32f429-linux-builder
% make
```

Le fichier **config/source** contient les commandes permettant de télécharger les sources des trois composants :

- le bootloader,
- le noyau,
- BusyBox.

Ces trois éléments seront ensuite compilés dans l'ordre sur la base des configurations listés dans **mk/uboot.mak**, **mk/kernel.mak** et **mk/rootfs.mak**. La compilation d'U-Boot repose sur une configuration spécifique créé pas Robutest : **stm32429-disco**. Le noyau, quant à lui, est configuré par un fichier **configs/kernel_config** correspondant, là encore, à la configuration initialement placée dans le fichier **config.robutest** sur le dépôt GitHub de Robutest. Il en va de même pour BusyBox via **configs/busybox_config**. La commande **make** va systématiquement ré-appliquer ces configurations même si vous procéder à des changements dans les configurations des répertoires **u-boot** et **uclinux**. En réalité, la compilation elle-même se fait dans le sous-répertoire **out/** et laisse les sources synchronisées par Git intacts.

C'est d'ailleurs également là que vous trouverez les binaires qui pourront être installés dans la flash du microcontrôleur :

- U-Boot : **out/uboot/u-boot.bin**
- le noyau : **out/kernel/arch/arm/boot/xipuImage.bin**
- le système de fichiers racine : **out/romfs.bin**

Notez la petite particularité dans la création de l'image du noyau. Lors de l'utilisation du XiP (*eXecute in Place*) permettant l'exécution du code binaire directement depuis la mémoire flash (exactement comme pour un microcontrôleur 8 ou 16 bits en développement *bare metal*), le point d'entrée doit être l'adresse de chargement (avec U-Boot) plus 64. Ceci étant, il est nécessaire de préfixer l'image du noyau de 64 octets à **0xFF**, d'où la présence du fichier **uclinux/arch/arm/boot/tempfile** et l'utilisation de **cat** pour concaténer son contenu avec **out/kernel/arch/arm/boot/xipImage** avant l'invocation de **mkimage** :

```
out/uboot/tools/mkimage -x -A arm -O linux -T kernel \
-C none -a 0x08020040 -e 0x08020041 \
-n "Linux-2.6.33-arm1" \
-d out/kernel/arch/arm/boot/xipImage.bin \
out/kernel/arch/arm/boot/xipuImage.bin
```

2.3 Texane/ST-Link, OpenOCD et problème

En jetant un oeil à **mk/flash.mak**, on remarque l'utilisation d'OpenOCD et non du très généralement utilisé *stm32 discovery line linux programmer* de Fabien Lementec (alias Texane) et ses camarades, avec les plateformes STM32. Les devkits « discovery » sont équipés d'une interface de programmation et de débogage comme la plupart des devkits du domaine, tout fabricants confondu. STM utilise une interface ST-Link/V1 ou ST-Link/V2 qui prend la forme d'une implémentation dans un microcontrôleur STM32F103 sur les kits. Dans le cas du 32F429IDISCOVERY c'est un protocole ST-Link/V2, reposant sur USB *raw* (par opposition au