ENSICAEN
Embedded Systems

BOARD SUPPORT PACKAGE

Embedded Linux

Hugo Descoubes – mars 2014

Une BSP Linux (Board Support Package) est un système d'exploitation complet combinant un jeu de composants logiciel en user space, des librairies, des drivers et services kernel prêts à l'emploi pour une plateforme matérielle donnée (interfaces matérielles de la board connues). *Ceci est à comparer aux distributions dans le monde du PC qui sont censées tourner sur n'importe qu'elle machine au monde (souvent x86/x64).*

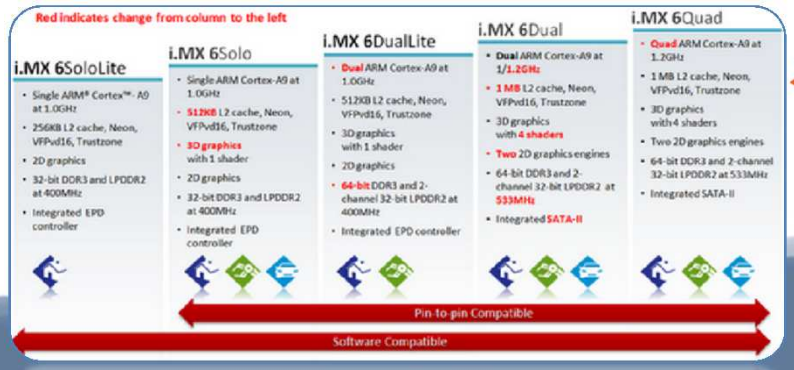
La Beagle Bone est une plateforme d'évaluation, il est néanmoins possible de travailler sur de boards durcies (souvent fournies avec BSP) pour une future intégration en milieu industriel (ex. EOLANE sur Caen) :

éolane
Caen

Le marché Allemand étant beaucoup plus riche que celui français sur le domaine des BSP's complètes Linux/Windows/Android (BSP essentiellement sur famille i.MX6 Freescale et AM335x de TI). Un grand nombre d'acteurs Allemands sont présents sur ce secteur, notamment PHYTEC :

PHYTEC

Pour information, Freescale propose la famille i.MX6 de processeurs qui est extrêmement bien pensé (single/dual/quad cores), très modulable, software et pins compatible :



freescale semiconductor

Afin de créer un système Linux complet plusieurs solutions sont possibles :

- ***Distribution précompilée :*** *rapide à mettre en œuvre mais manque de flexibilité. Seules certaines architectures sont supportées. Ce que nous avons fait jusqu'à présent.*

- ***Compilation manuelle des composants :*** *grande flexibilité mais exercice long, fastidieux et non reproductible qui nécessite une gestion manuelle des dépendances inter-paquets.*

- ***Outils d'automatisation de génération de système :*** *solution flexible et reproductible compilant directement les sources de chaque package (BuildRoot, OpenEmbedded, Yocto …)*

*Plaçons nous dans le cas d'une BSP sur laquelle nous souhaitons rajouter quelques services, pour une phase de prototypage par exemple. Première solution, passer par un gestionnaire de paquets, si le package est proposé (**apt-get** sous Debian, **opk** sous Angstrom, **yum** sous Red Hat …). Bien s'assurer de la configuration réseau de la cible (proxy système, serveur DNS …) :*

/etc/network/interfaces

```
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet static
        address 172.24.123.13
        netmask 255.255.0.0
        gateway 172.24.0.1
        dns-nameservers 193.49.200.14
```

/etc/network/ifstate

```
lo=lo
eth0=eth0
service restart networking
```

/etc/resolv.conf *(vérifier config. serveur DNS)*

```
domain localdomain
search localdomain
nameserver 193.49.200.14
```

```
export http_proxy=http://proxy.ensicaen.fr:3128
export ftp_proxy=http://proxy.ensicaen.fr:3128
```

*Beaucoup de packages nécessites la récupération et la recompilation des sources, c'est le cas pour **can-utils** (gestion du nombre de plateformes cibles et de versions quasi impossible à maintenir). Néanmoins, si vous avez besoin d'installer un grand nombre de paquets, cette solution peut devenir très longue et fastidieuse, notamment pour des problématiques de gestion de version et de dépendance entre packages (toujours regarder les dates de mise à jour des solutions en ligne). Observons les dépendances nécessaires afin d'installer can-utils (http://baydogar.blogspot.fr/2013/05/cross-compiling-can-utils.html) :*

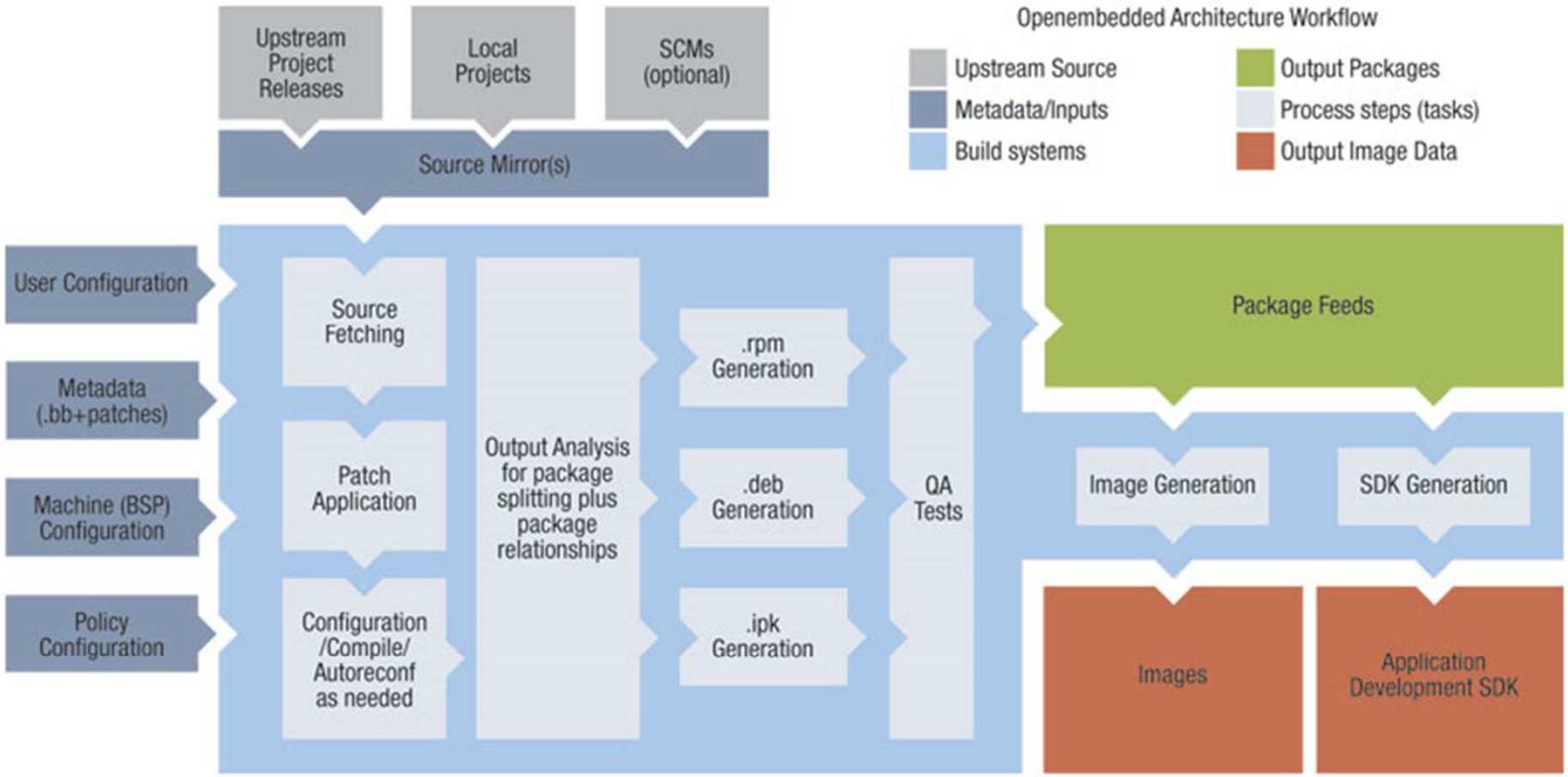
- ***iproute** (ici, vs 2.6.39)*
- ***libsocketcan** (ici, vs 0.0.8)*
- ***canutils** (ici, vs 3.0.2)*

Il existe sinon plusieurs solution permettant l'automatisation du processus de génération de systèmes Linux complets :

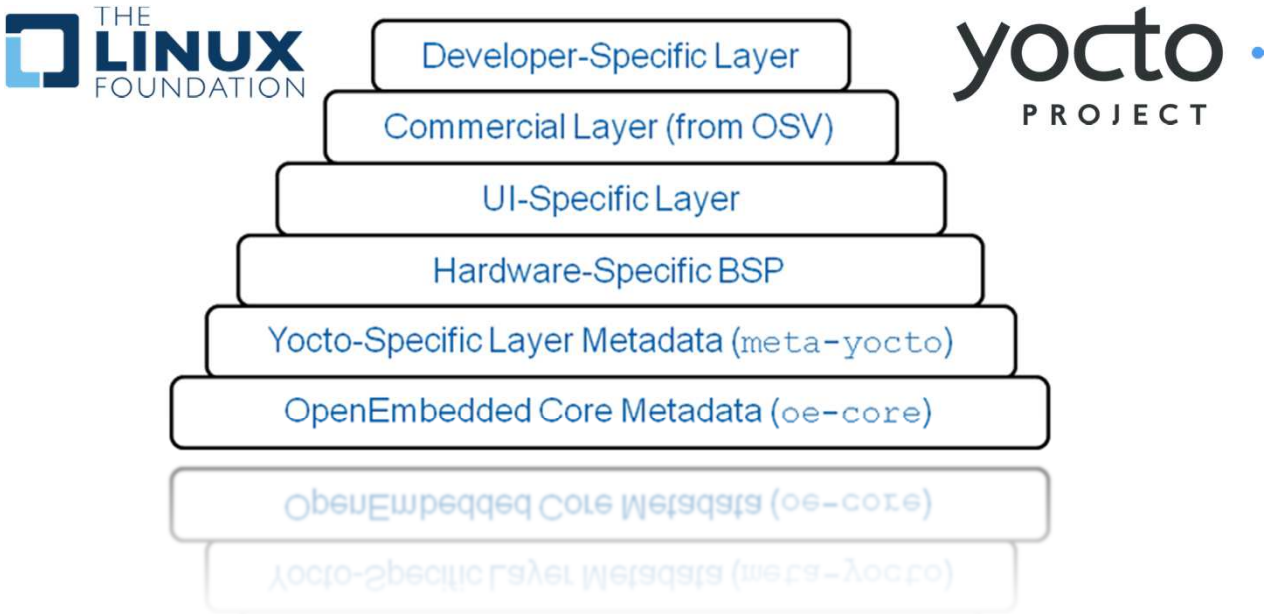
- ***BuildRoot*** *(http://buildroot.uclibc.org) : outil simple et efficace de génération de systèmes Linux basé sur la commande* ***make****. Configuration via interface* ***ncurses*** *(cf. make menuconfig) bien adaptée au développement de petits systèmes. BR manque néanmoins d'un gestionnaire dynamique de packages (souvent inutile pour une application embarquée) et supporte environ 1000 paquets en 2014, peu être problématique pour la génération de grosses distributions, notamment pour le monde du PC.*

- ***OpenEmbedded*** *(http://www.openembedded.org/) : outil puissant mais plus lourd à prendre en main que BuildRoot adapté à la génération de tout type de distribution basé sur le moteur d'exécution et de gestion de métadonnées* ***BitBake*** *(moteur écrit en python à comparer à make). OpenEmbedded travaille avec un ensemble de* ***métadonnées*** *composées de fichiers de configuration, de classes et de recettes/recipes basé sur le principe de l'héritage. Chaque recette (.bb) décrit les tâches à effectuer afin de construire une image, un package et gérant les dépendances associées (plusieurs milliers de paquets supportés).*

Observons le processus de génération de système ou workflow utilisé par OpenEmbedded :

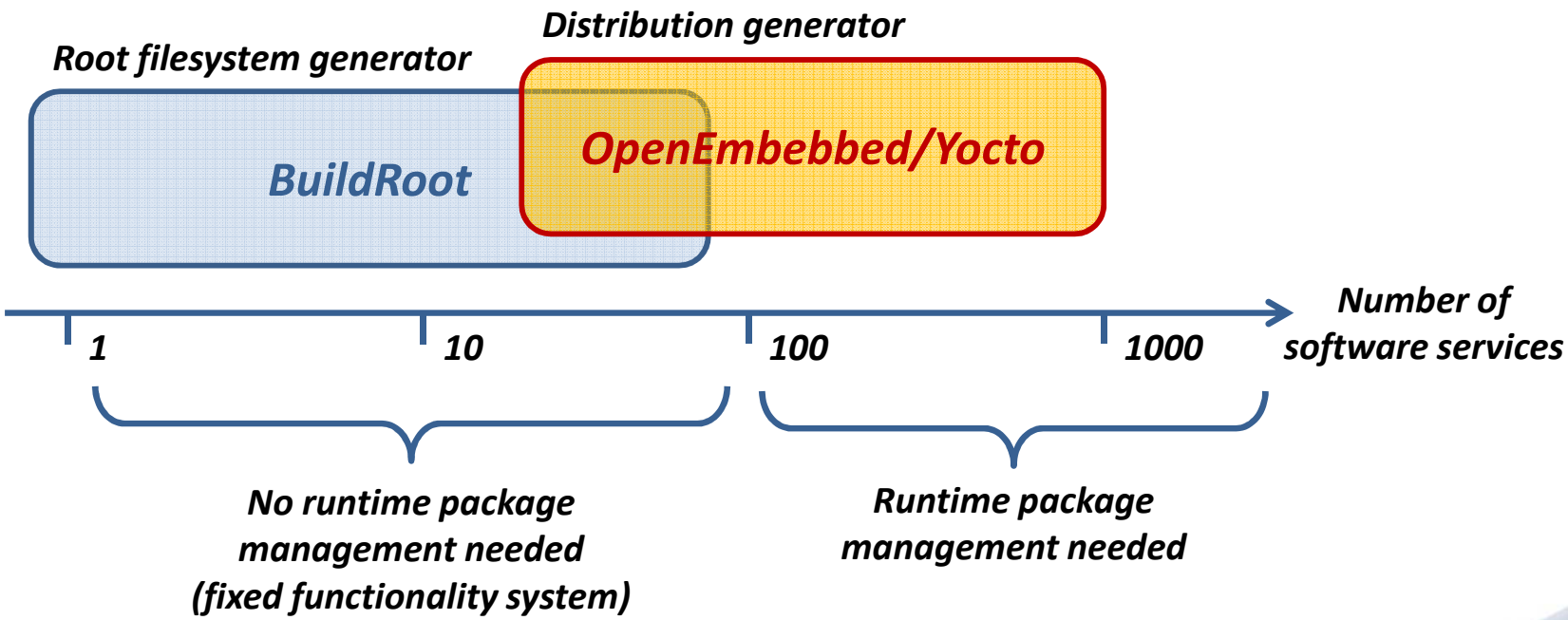


- ***Yocto*** *(https://www.yoctoproject.org/) : projet OpenSource collaboratif porté par la Linux Foundation dit projet "chapeau" (umbrella project) travaillant notamment autour du projet OpenEmbedded (Poky, Eglibc, Matchbox …).*

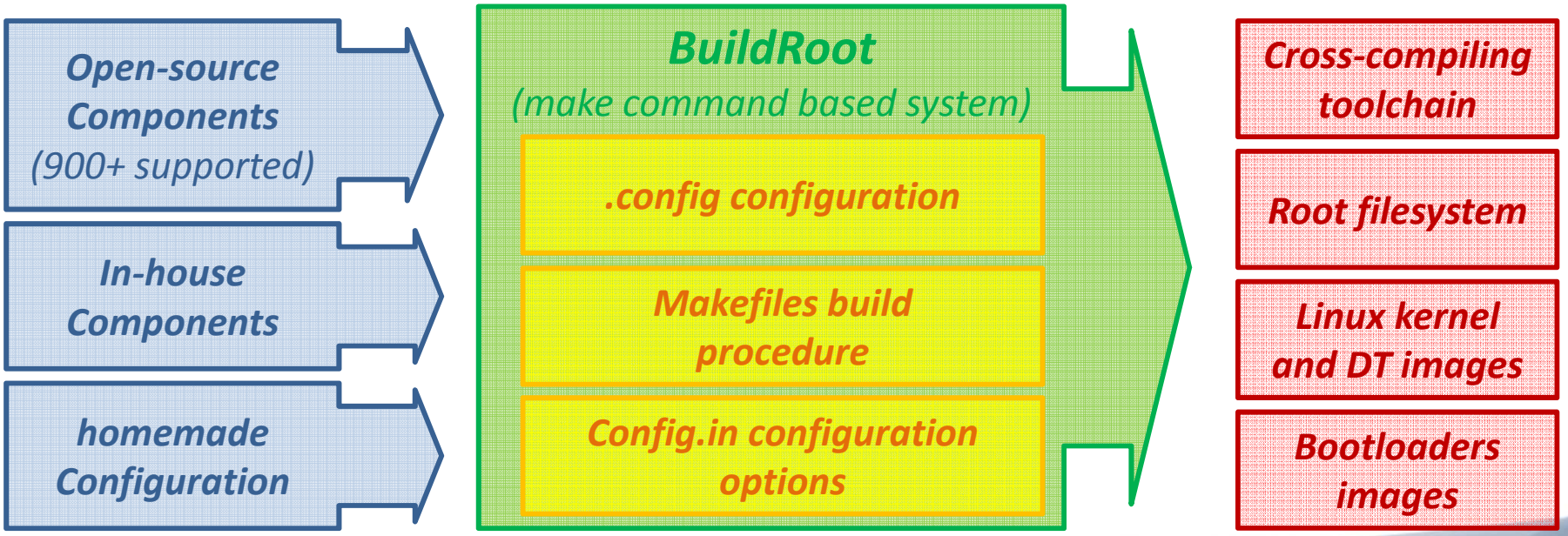


- ***Scratchbox***, ***OpenWRT*** *(dérivé de BuildRoot avec gestion de paquets IPK) …*

En fonction de l'application, du marché ciblé et du nombre de composants logiciels à déployer en user space, nous pouvons adapter l'outil de génération de système à utiliser en fonction de nos besoins :



*Intéressons-nous maintenant à **BuildRoot**, projet initialement créé en 2001 par la communauté de développeur du projet uCLibc pour la génération de petits systèmes Linux (début réel du projet en tant que tel en 2005). Observons le workflow typique de BuildRoot :*



Se placer dans le répertoire ***~/elinux/work/buildroot/*** *et récupérer la dernière version de Buildroot sur le site officiel dans la section download (documentation, http://buildroot.uclibc.org/downloads/manual/manual.pdf) :*

```
cd  ~/elinux/work/buildroot/
wget http://buildroot.org/downloads/buildroot-<current-version>.tar.bz2
tar xjf buildroot-<current-version>.tar.bz2
cd buildroot-<current-version>
```

Un certain nombre de packages sont également nécessaires côté host, s'assurer de leur bonne installation :

```
sudo apt-get update
sudo apt-get install \
build-essential gawk bison flex gettext texinfo patch \
gzip bzip2 perl tar wget cpio python unzip rsync
```

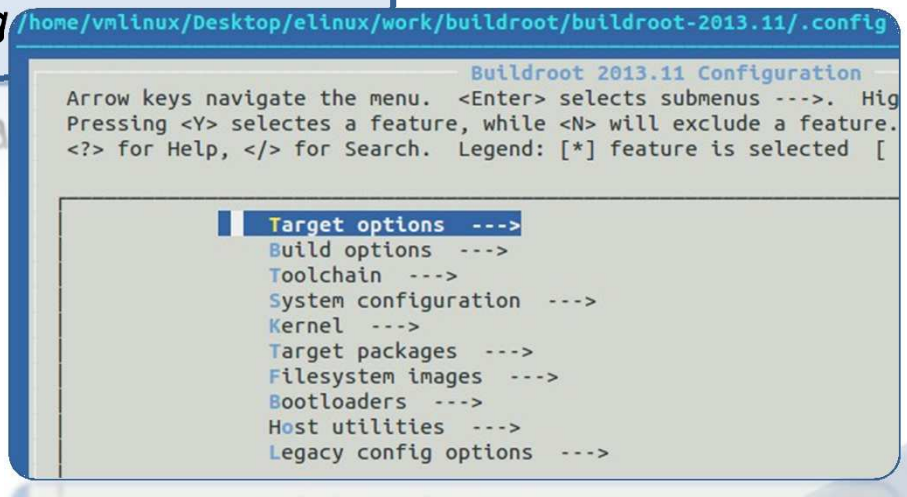
Observons succinctement le système de fichiers de BuildRoot. Nous pouvons constater qu'avant la génération d'un premier système, les différents répertoires sont vides et ne contiennent que des fichiers de configuration et d'automatisation de procédures :

- ***arch/*** *: pré-configurations fournies architecture CPU dépendant.*
- ***board/*** *: pré-configurations kernel fournies pour des boards du marché. Permet également de sauver les différents fichiers et composants maison complémentaires propres à notre plateforme de développement (non vu dans cet enseignement).*
- ***output/*** *: répertoire de destination, notamment des images (kernel, device tree, Bootloader …) et rootfs générés.*

- ***configs/*** *: pré-configurations BuildRoot fournies pour des plateformes du marché (beaglebone, raspberryPI, pandaboard …).*
- ***docs/*** *: documentation BuildRoot*
- ***fs/*** *: recettes ou recipes utilisées par BR pour la génération du système de fichiers choisi.*
- ***linux/*** *: recettes ou recipes utilisées par BR pour la génération de l'image kernel.*
- ***boot/*** *: recettes ou recipes utilisées par BR pour la génération du bootloader choisi.*
- *…*

Appliquer la configuration par défaut donnée pour la beaglebone et ouvrir l'interface ncurse de configuration :

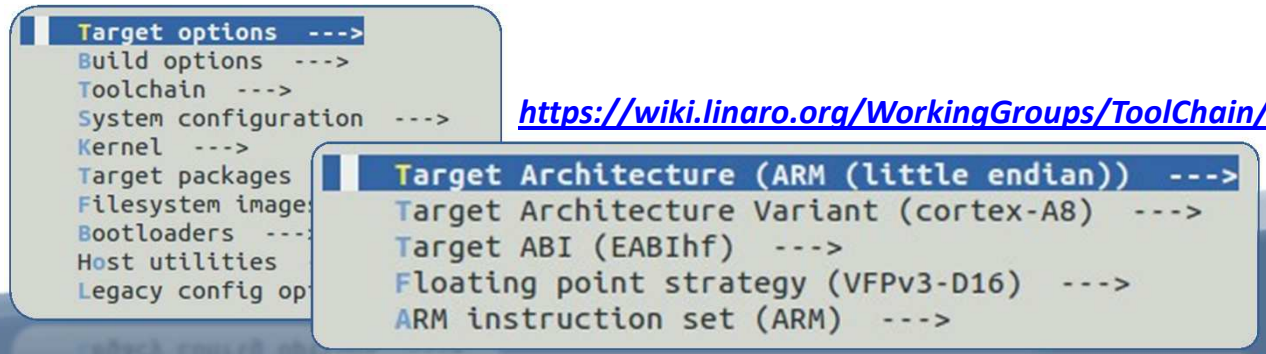
```
cd  ~/elinux/work/buildroot/buildroot<current-version>/
make distclean
cp  configs/beaglebone_defconfig .config
make  menuconfig
cp .config  board/beaglebone/br-bbb-config
```



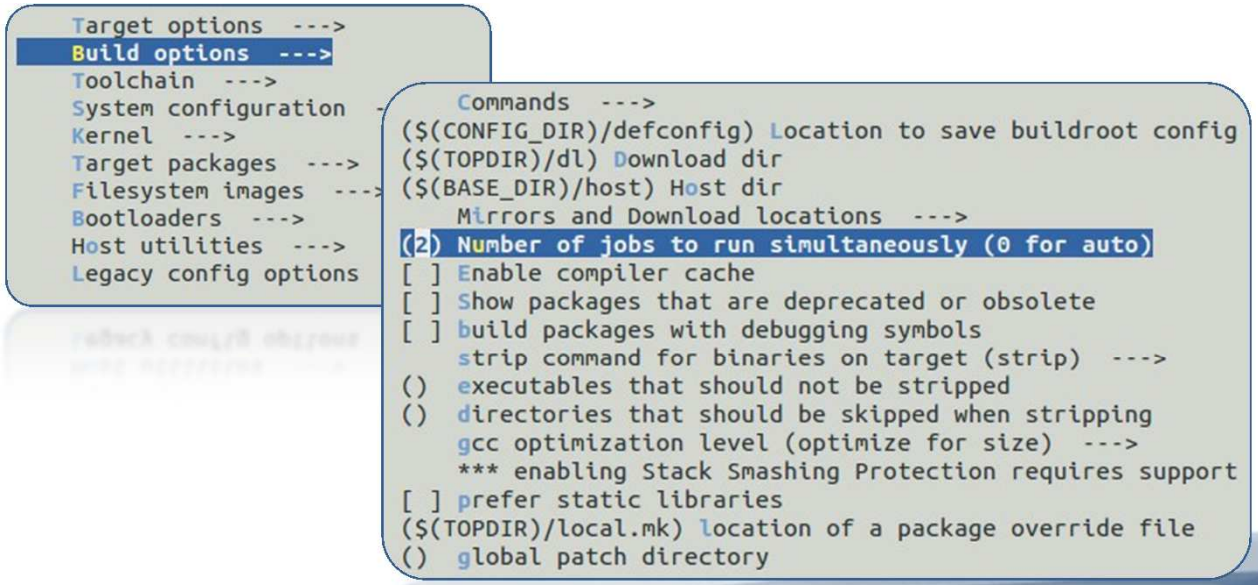
Etudier les configurations proposées et adapter cette configuration à notre projet. Nous n'utiliserons BuildRoot que pour la génération d'un rootfs (minimisation du temps de génération), nous garderons donc dans un projet séparé la récupération, l'application des patches et la compilation du kernel, du device tree, des modules, des firmwares et des deux étages de bootloader.

- ***Target options :*** *description précise de l'architecture CPU, utile afin de lever des options d'optimisation à la compilation. Par exemple :*

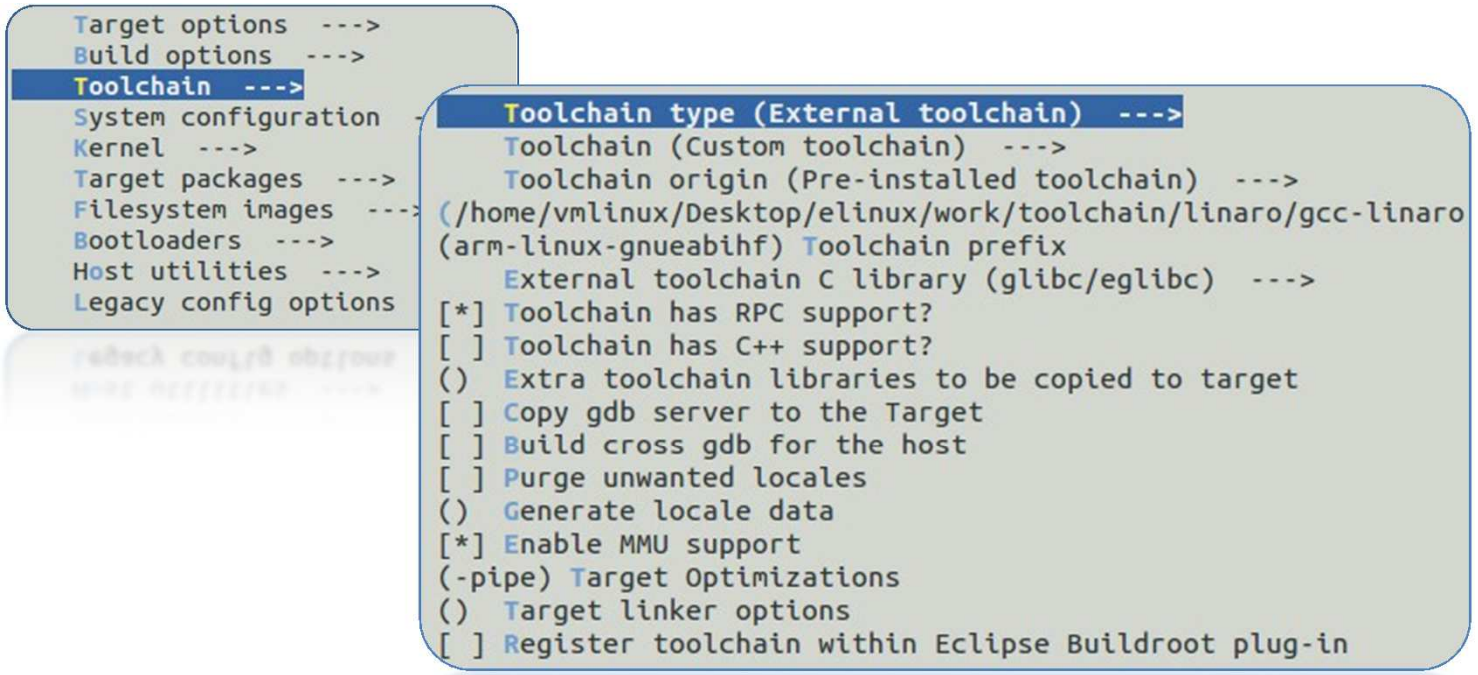
  arm-linux-gnueabihf-gcc –mcpu=cortex-a8 –mfpu=vfpv3-d16 –mfloat-abi=hard …



- ***Build options :*** *options de compilation (options d'optimisation, nombre de jobs, différents chemins utiles au projet, librairies statiques ou dynamiques, stripper/épurer les exécutables …). Dans notre cas, modifier le nombre de jobs (nombre de CPU's utilisés côté host pour la compilation).*



- ***Toolchain :*** *choix de la chaîne de cross-compilation ou ABI (interne à BuildRoot ou externe à télécharger ou locale préinstallée) et de la librairie standard associée.*



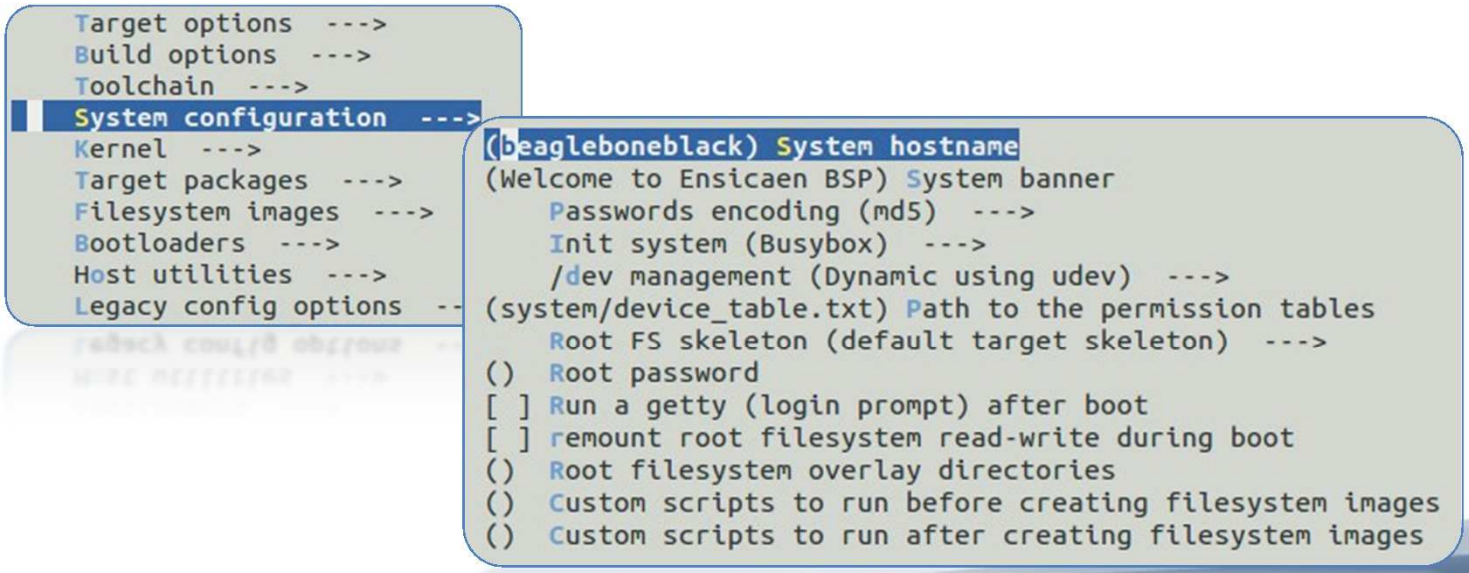
A une ABI (ARM dans notre cas) utilisée est associée une librairie standard, outil indispensable utilisé par la suite par tout composant logiciel du système. Présentons les 3 principales librairies standards rencontrés dans l'embarqué :

- ***glibc** (GNU C Library) : librairie standard du C, la même que celle portée sur des distribution GNU/Linux dans le monde du PC. Elle a pour avantage d'être très riche nativement en services, très robuste et très bien maintenue. Néanmoins, il s'agit de la librairie offrant la plus forte empreinte mémoire. Depuis la version 2 de la glibc (nommée libc6), elle est connue sur système Linux sous le nom libc.so.6 (présente sur le rootfs dans **/lib/<target-architecture>/**)*

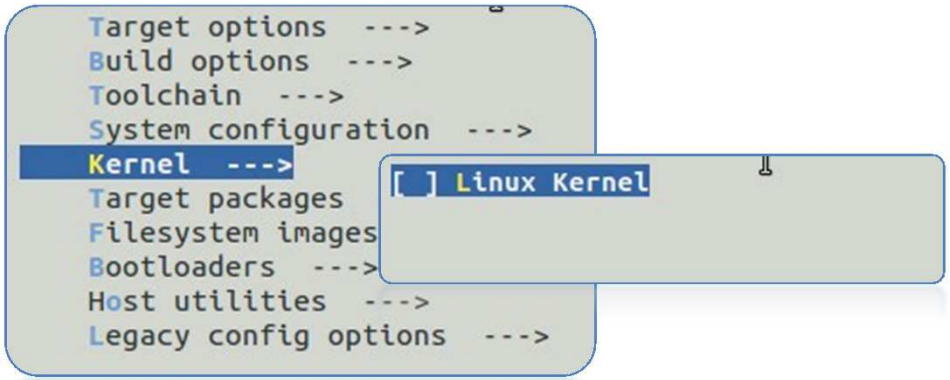
- ***eglibc*** *(Embedded GNU C Library) : variante configurable et optimisée pour l'embarqué de librairie standard du C glibc (en 2009, Debian annonça la migration de la glibc vers la eglibc pour leurs systèmes). eglibc est depuis 2011 un composant du projet Yocto. Cette librairie est un projet très bien maintenue offrant une meilleure empreinte mémoire que la glibc mais néanmoins plus large que uClibc.*

- ***uClibc*** *(microcontroller libc) : projet optimisé pour l'embarqué initialement prévu pour uCLinux (variante de Linux pouvant tourner sur processeur sans MMU). Cette bibliothèque est la plus compacte, elle est hautement configurable au détriment des services proposés et des performances.*

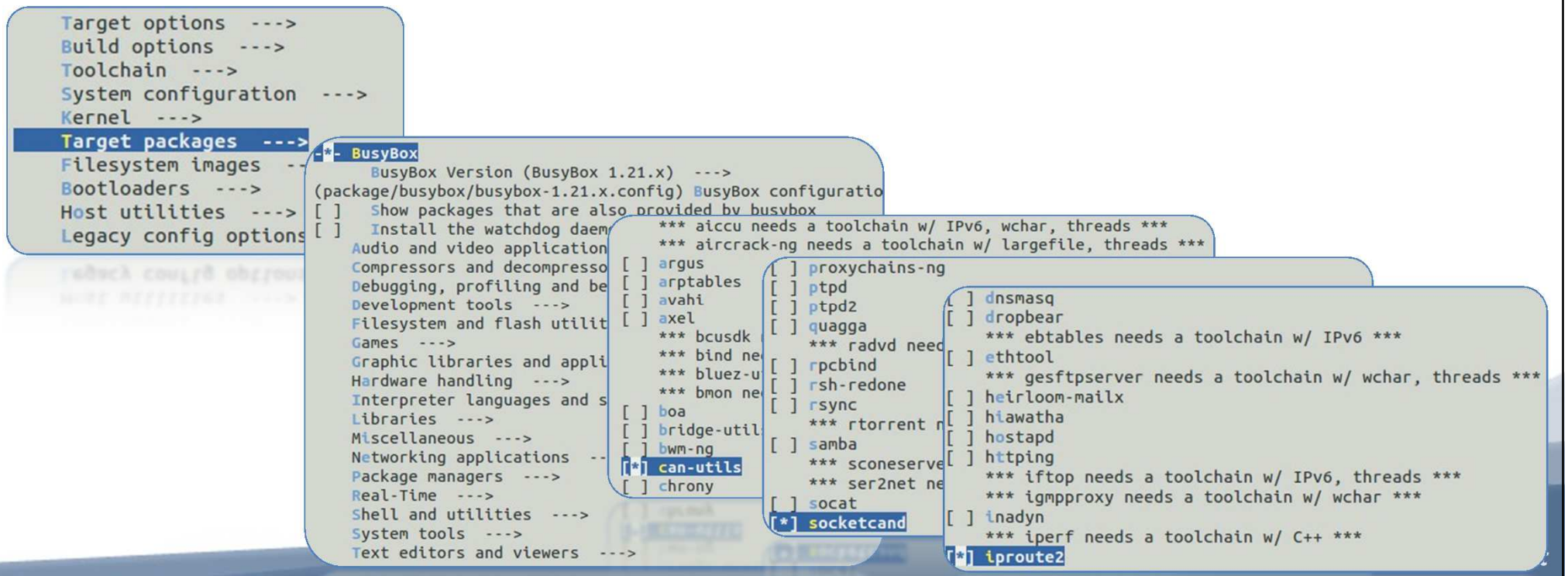
- ***System configuration :*** *configuration du système durant la phase d'amorçage (login, mdp, /dev stratégie de management …). Il faut savoir qu'en user space, le répertoire /dev peu être géré statiquement (plus performant mais moins évolutif) ou dynamiquement (cf. distributions dans le monde du PC, utilise souvent udev).*



- ***Kernel :*** *récupération, application des patches et configuration du kernel Linux (compression éventuelle) et du device tree. Cette étape, notamment l'application des patches, pouvant être difficilement configurable en fonction de la stratégie de récupération des patches, nous préférerons garder le processus de compilation du kernel dans un projet annexe.*



- ***Target packages :*** *customisation du rootfs cible. En 2013, près de 900 packages sont supportés par BuildRoot. Rajouter les composants nécessaires à du prototypage d'application CAN (rappel, "/" permet une recherche rapide et une validation des dépendances sous interface ncurses).*



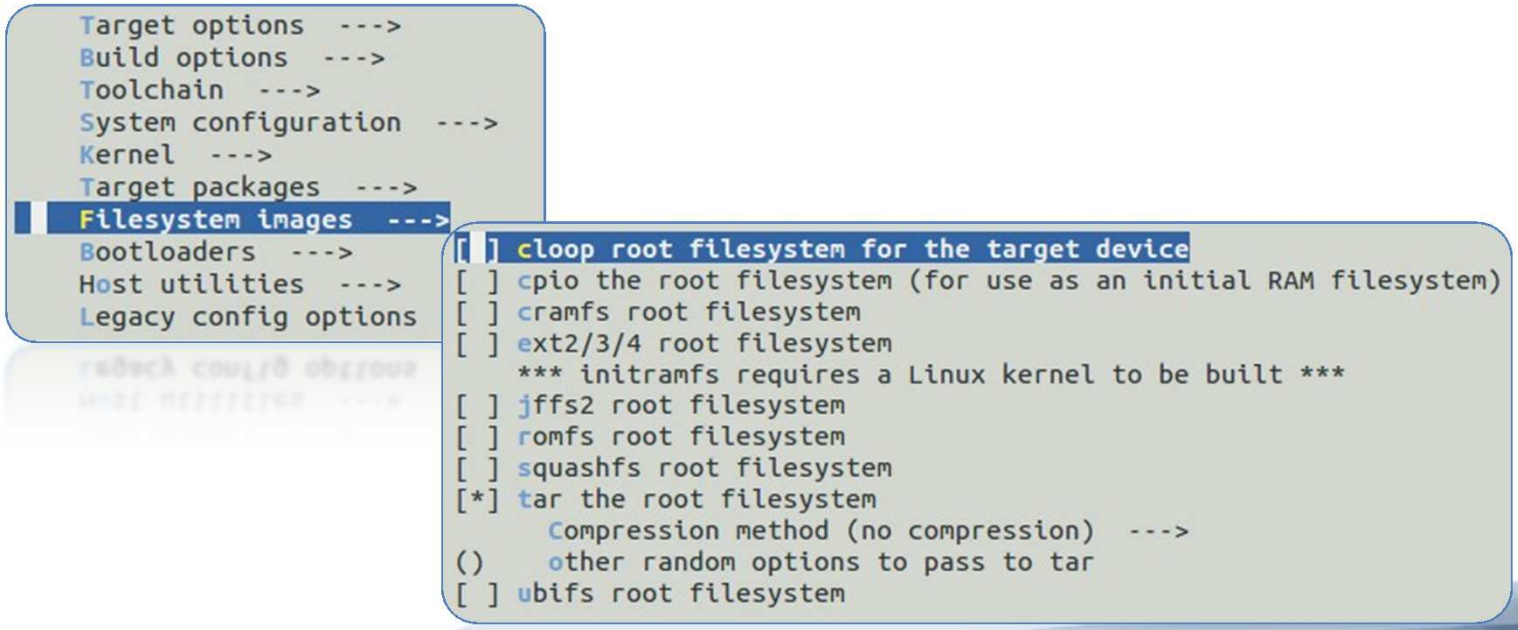
Observons deux des principales stratégies rencontrées afin de gérer un jeu de packages binaires exécutables :

- ***Standard :*** *chaque package est un objet binaire exécutable indépendant, solution équivalente aux distributions GNU/Linux rencontrées dans le monde du PC.*

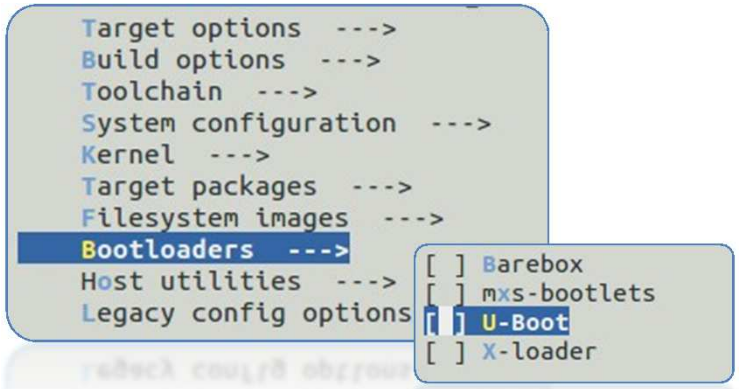
- ***BusyBox :*** *objet binaire exécutable unique, configurable à la compilation et pouvant encapsuler des centaines de commandes systèmes (via liens symboliques). Cette solution permet de minimiser grandement l'empreinte mémoire du jeu de commande cible.*

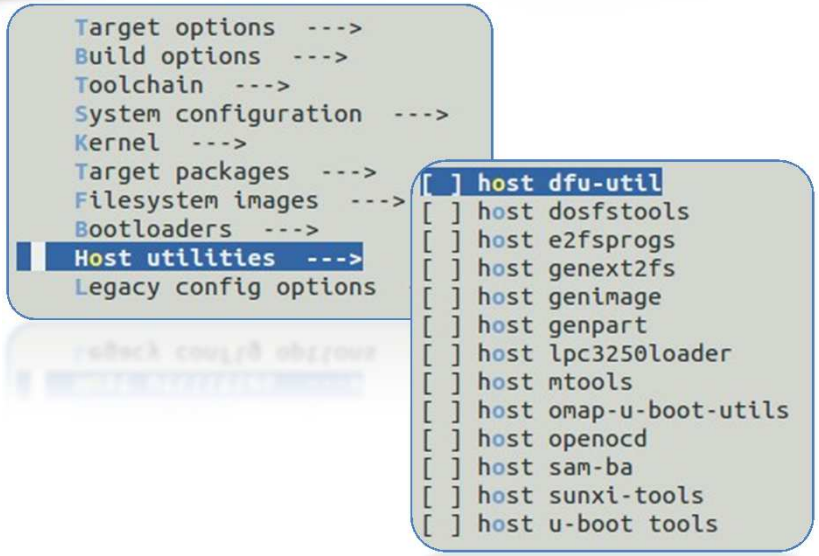
```
lrwxrwxrwx  1 vmlinux vmlinux      7 févr.  8 00:43 ash -> busybox
-rwxr-xr-x  1 vmlinux vmlinux 601288 févr.  8 00:54 busybox
lrwxrwxrwx  1 vmlinux vmlinux      7 févr.  8 00:43 cat -> busybox
lrwxrwxrwx  1 vmlinux vmlinux      7 févr.  8 00:43 catv -> busybox
lrwxrwxrwx  1 vmlinux vmlinux      7 févr.  8 00:43 chattr -> busybox
lrwxrwxrwx  1 vmlinux vmlinux      7 févr.  8 00:43 chgrp -> busybox
lrwxrwxrwx  1 vmlinux vmlinux      7 févr.  8 00:43 chmod -> busybox
lrwxrwxrwx  1 vmlinux vmlinux      7 févr.  8 00:43 chown -> busybox
lrwxrwxrwx  1 vmlinux vmlinux      7 févr.  8 00:43 cp -> busybox
```

- ***Filesystem images :*** *type d'image du rootfs à générer (ext2, ext3, ext4 …) et compression éventuelle. Dans notre cas, nous générerons un rootfs sous forme d'une simple archive de type tarball sans processus de compression (facilite le déplacement, le partage et l'installation par la suite).*



- ***Bootloaders :*** *sélection du bootloader (3rd stage bootloader) et éventuellement du second étage de bootloader, dans notre cas le MLO proposé par TI. Concernant notre projet, nous possédons déjà les sources et un binaire précompilé des différents bootloaders.*





- ***Host utilities :*** *installation éventuelle d'utilitaires pour la machine host. Non nécessaire dans notre cas.*

- ***Legacy config options :*** *options de configuration BuildRoot retirée entre la version courante et les versions antérieures. Non utilisé dans notre cas.*

*Observons le contenu du répertoire de sortie de BuildRoot (**~/elinux/work/buildroot/buildroot<current-version>/output/**) :*

- ***build :** répertoire correspondant aux points d'extractions et de compilation des packages, et contenant donc les sources.*
- ***host :** répertoire correspondant aux points d'extractions et d'installation d'utilitaires pour le host. **host/usr/<toolchain-prefix>/sysroot/** contient le sysroot utilisé par la toolchain.*
- ***staging :** lien symbolique vers le **sysroot** (rootfs-like minimaliste utilisé par la chaîne de cross-compilation pour les fichiers d'en-tête et les bibliothèques)*
- ***target :** point d'installation des bibliothèques et composants cibles*
- ***toolchain :** vide si utilisation d'une toolchain externe*
- ***images :** répertoire d'installation du rootfs (tarball, ext2, ext4 …) et images binaires pour la cible (kernel, device tree, bootloaders …)*

Une fois la configuration effectuée, la sauvegarder côté host et lancer le processus d'exécution. Extraire ensuite le rootfs généré vers la MMC/SDcard :

```
cp .config  board/beaglebone/br-bbb-config
make
cd output/images/
tar –xvf rootfs.tar –C /media/rootfs
```

Modifier le fichier de configuration uEnv.txt de U-Boot en adaptant le point d'entrée du rootfs à notre système :

```
optargs=quiet init=/sbin/init
```

*Vous constaterez une erreur au démarrage du système, le processus **init** ne trouve pas le chemin vers les bibliothèques dynamiques :*

```
[    0.864441] pinctrl-single 44e10800.pinmux: could not request pin 21 on device pinctrl-single
/sbin/init: error while loading shared libraries: libc.so.6: cannot open shared object file: No such file
[    1.210266] Kernel panic - not syncing: Attempted to kill init! exitcode=0x00007f00
```

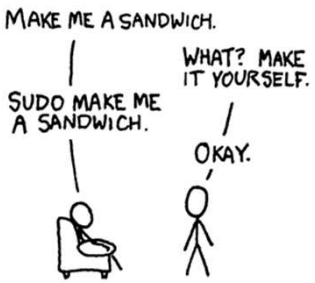
*Importer le **sysroot** utilisé par la chaîne de compilation vers le rootfs précédemment généré. Rebooter le système … et vous voilà sur votre premier système Linux créé from scratch !*

```
cd  ~/elinux/work/buildroot/buildroot<current-version>/output/staging/
tar –vcf sysroot.tar .
cd ../images/
mkdir  rootfs
tar –xvf rootfs.tar –C rootfs/
tar –xvf ../staging/sysroot.tar –C rootfs/
cd rootfs/
tar –vcf rootfs.tar .
tar –xvf rootfs.tar –C /media/rootfs
```

A ce stade là, il ne reste plus qu'à croiser les doigts et tester les composants installés. Commençons par les services ***canutils*** *(maybe in few minutes it's coffee time … again) :*

http://www.armadeus.com/wiki/index.php?title=CAN_bus_Linux_driver

```
Welcome to Ensicaen BSP
beagleboneblack login: root
# cansend can0 500#R
write: Network is down
# ip link set can0 up type can bitrate 125000
# ifconfig can0 up
# cansend can0 500#R
# can
can-calc-bit-timing  cangen               cansend
canbusload           cangw                cansniffer
candump              canlogserver
canfdtest            canplayer
```



A faire :

- ***Partie sur Yocto :*** *Le projet manquant encore un peu de maturité et d'investissement de la part de certains acteurs en début 2014, notamment Freescale, cette partie sera créée dans un futur proche mais avec un peu de prudence.*

Merci de votre attention !