5TC option Embed

Linux embarqué : construire son OS

Antoine Fraboulet, Tanguy Risset

prénom.nom@insa-lyon.fr
Labo CITI, INSA de Lyon, Dpt Télécom

19 août 2016

Références

Certains slides et illustrations proviennent de

- Free Electrons :
  `http://free-electrons.com/`
- Présentations BeagleBoard :
  `http://beagleboard.org/`
- Présentation raspberryPi :
  `http://www.raspberrypi.org/faqs`

Historique Linux embarqué

- Nombreuses versions de Linux embarqué pour différents types de matériel (Téléphones portable, PDA, set-top boxes, ...).
- Les caractéristiques de ces systèmes :
  - peu de RAM
  - mémoire Flash (pas de disque dur)
  - De nombreux périphériques spécifiques

$\Rightarrow$ Noyau Linux léger, souvent temps réel

- Quelques version historiques de Linux embarqué :
  - Distribution Linux Familiar (PDA)
  - Open Moko (smart phones)
  - Busy Box (commandes essentielles Unix en un seul fichier)
  - Open Zaurus (PDA Sharp)
  - ...

Construire un Linux embarqué

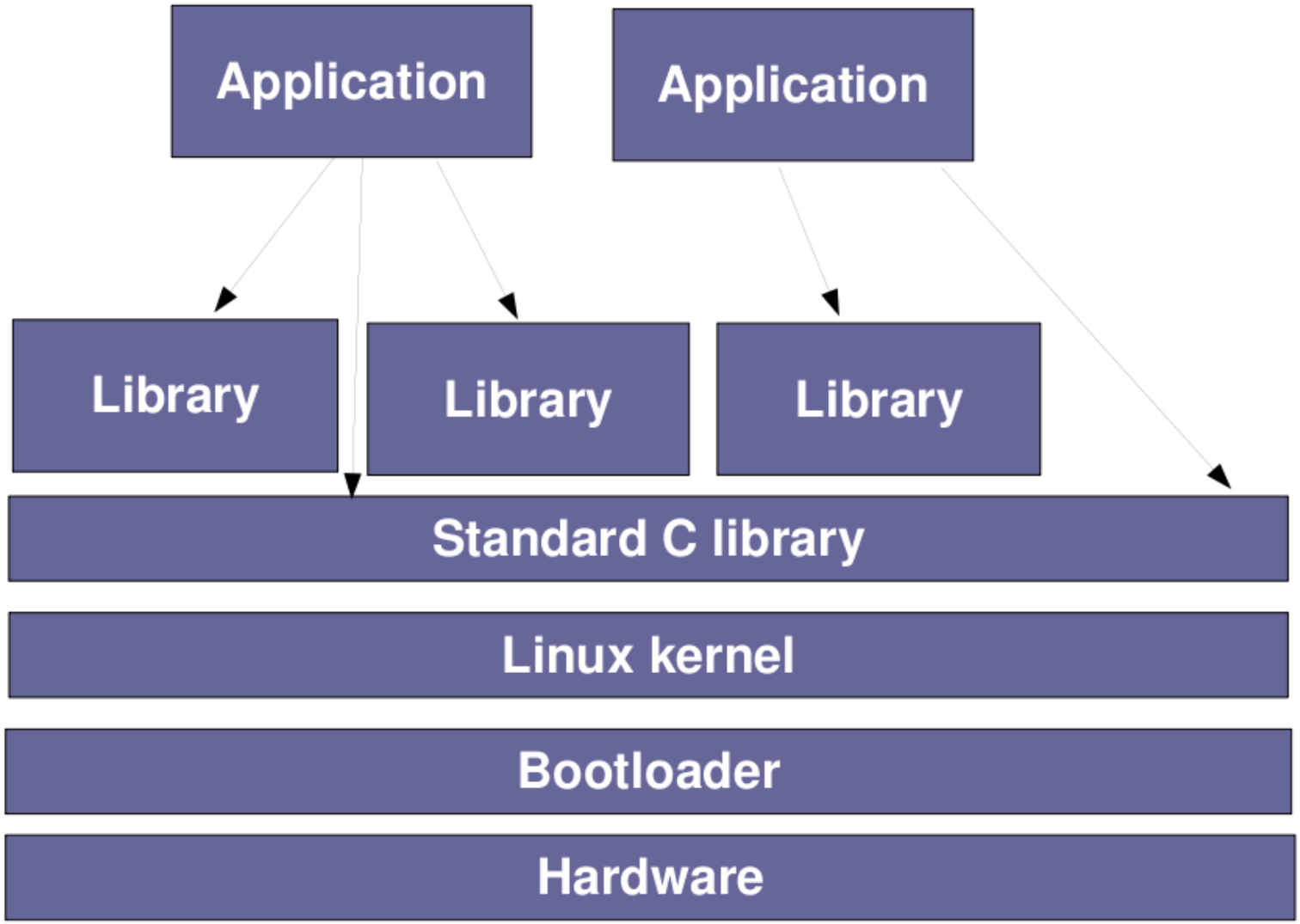
- Devant le nombre croissant de plate-formes apparaissant, les développeurs ont mis en commun leurs efforts pour développer des plate-formes de construction de linux embarqué.

$\Rightarrow$ CrossTools

$\Rightarrow$ OpenEmbedded

  - Construire un noyau simple pour l'architecture cible
  - Sélectionner seulement les paquets nécessaires
  - Mettre en place le système de boot
  - Construire un environnement de compilation croisée

Architecture globale



Matériel embarqué

- Le matériel embarqué est différent du matériel classique :
  - Diverses architectures de processeur : ARM, MIPS, PowerPC... et x86 aussi
  - Stockage en mémoire Flash (de type NAND ou NOR), limitée en capacité (quelques MB à quelque centaine de MB)
  - RAM à capacité limitée (Quelques MB à quelques dizaines de MB)
  - Différents bus spécifiques : I2C, SPI, CAN...
- Existence de cartes de développement avec des facilités de téléchargement du code et de debug.

Exemple

Beaglboard

- Arm Cortex A8 1 GHz
- DaVinci digital media processor (DM3730, ARM+DSP)
- Puce graphique 3D
- 512 MB of DDR SDRAM
- 4GB SD-Card
- I^2C, SPI, DVI-D, S-Video, 4 port USB Hub, Stereo In/Out, Ethernet 10/100...

- **Rasberry Pi**
- Broadcom BCM2835, 700 MHz ARM avec FPU
- GPU Videocore 4
- RAM 512 Mo.
- 4GB SD-Card
- video RCA 2 port USB Hub, Stereo Out, Ethernet 10/100...

Configuration minimale pour Linux embarqué

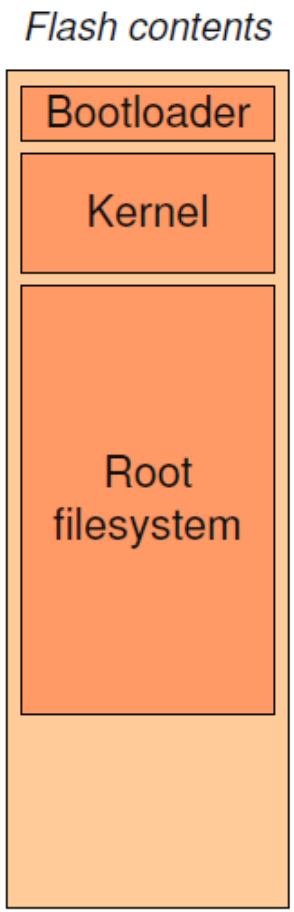
- Un CPU supporté par GCC et le noyau linux
  - CPU 32 bits
  - Les CPU sans MMU sont aussi supportés à travers le projet uClinux
- Au moins 4 à 8 MB de RAM
- Au moins 2 à 4 GB de stockage mémoire
- Linux n'est pas adapté aux systèmes basés sur les petits micro-controleurs comme l'EZ430 (quelque centaines de kB de Flash et RAM), pour ces systèmes là :
  - Pas d'OS
  - OS dédié (Contiki, FreeRTOS...)

Composants logiciels

- Une chaîne de compilation croisée
  - Compilateur qui tourne sur la machine de développement mais qui génère du code pour la plate-forme embarquée.
- Bootloader
  - Démarré par le matériel directement, responsable des initialisations de base, du chargement et de l'exécution du noyau
- Noyau Linux
  - Responsable de la gestion des différents services pour les applications (processus, mémoire virtuelle, pile réseau, pilote de périphérique, ...)
- Librairie C
  - Interface entre le noyau linux et les applications de l'espace utilisateur
- Librairies et application
  - Votre programme....

Système de fichier Root

- Dans un système Linux, plusieurs systèmes de fichiers peuvent être montés et créer un hiérarchie globale de fichiers et répertoires
- Un système de fichier particulier, le système de fichier `root` est monté en “`/` ”.
- Sur les Linux embarqués, le système de fichier `root` contient toutes les librairies, applications, et données du système.
- Construire ce système de fichier est l’une des tâche essentielle de l’intégration de Linux sur la plate-forme embarquée.
- En général le noyau est séparé du système de fichier `root`



Chaîne de compilation croisée

- La chaîne de compilation classique est une *chaîne de compilation native*
- Pour les systèmes embarqués, on utilise une chaîne de compilation croisée (cross-compiling toolchain).
- Elle est généralement composée de
  - Binutils : `as`, `ld`, `objdump`, `nm`, `strip`, ...
  - En-têtes du noyau (*kernel header*)
  - Compilateur GCC
    - peut cibler différents CPU : ARM, AVR, Blackfin, CRIS, FRV, M32, MIPS, MN10300, PowerPC, SH, v850, i386, x86_64, IA64, Xtensa, etc
  - Bibliothèque C/C++
    - Composant essentiel, permet l'interaction entre les programmes et le noyau
    - De nombreuses déclinaisons :`glibc`, `uClibc`, `eglibc`, `dietlibc`, `newlib`
    - La `libC` est choisie en même temps que la chaîne de compilation croisée, `gcc` étant compilé avec.
  - Débugger GDB

Librairie C

- `glibc` GNU Lib C $\simeq$ 2.5MB pour ARM
- `eglibc` embedded `glibc`
- `uClibc`, librairie très légère
- `Dietlibc` ($\simeq$ 70KB)
- `Newlib`
- Exemples de tailles de programmes :

C program	*Compiled with shared libraries*		*Compiled statically*	
	glibc	*uClibc*	*glibc*	*uClibc*
Plain "hello world" (stripped)	5.6 K (glibc 2.9)	5.4 K (uClibc 0.9.30.1)	472 K (glibc 2.9)	18 K (uClibc 0.9.30.1)
Busybox (stripped)	245 K (older glibc)	231 K (older uClibc)	843 K (older glibc)	311 K (older uClibc)

Construire une chaîne de compilation

- Bien distinguer
  - La machine sur laquelle est construite la chaîne de compilation (*build machine*)
  - La machine sur laquelle la chaîne de compilation va être exécutée (*host machine*)
  - La machine sur laquelle les binaires générés par la chaîne de compilation vont être exécutés (*target machine*).
- En général, *host* et *build* machines sont les mêmes.
- Choisir la `libc`
- Choisir les versions des différents composants
- Configurer la chaîne :
  - format binaire, ABI (*Application Binary Interface*)
  - calcul flottant ?
  - Locale, IPV6, etc...

Étapes de construction

- Une fois tous les choix effectués (ou.. pour toutes les configurations choisies) :
  - Extraire et installer les en-têtes du noyau
  - Extraire, configurer et installer les `binutils`
  - Extraire, configurer et installer une première version de `gcc` qui va générer les binaires pour la cible.
  - Extraire, configurer et compiler la `libc` en utilisant le compilateur construit à l'étape précédente
  - Configurer et compiler le cross-compilateur `gcc`

Construire une chaîne de compilation croisée

- On peut le faire soi-même... en théorie...
- On peut utiliser une chaîne de compilation pré-compilée
  - CodeSourcery, `http://www.codesourcery.com` (2012 : racheté par Mentor...)
  - `http://elinux.org/Toolchains`
  - Mais, aucune prise sur la configuration (paquets installés, target machine...)
- Ou on peut utiliser des outils de génération de chaînes de compilation
  - CrossTools, Le précurseur (Dan Kegel), plus maintenu `http://www.kegel.com/crosstool`
  - Crosstool-ng, `http://crosstool-ng.org/`
  - Buildroot, `http://buildroot.uclibc.org`
  - PTXdist, `http://www.pengutronix.de/software/ptxdist`
  - OpenEmbedded : historiquement le premier complet `http://www.openembedded.org/`
  - Yocto project : aujourd'hui, le plus en vue `https://www.yoctoproject.org/`

Le bootloader

- Sur les architectures embarquées, le boot de bas niveau est très dépendant du CPU et de la plate-forme.
  - Certaines cartes ont une Flash à partir de laquelle le CPU boote après un reset
  - Certains CPU ont un morceau de code intégré dans une ROM qui charge automatiquement une petite portion de Flash dans la RAM
- Il existe de nombreux bootloader open-source génériques, parmis eux :
  - U-Boot, Universal boot, standard de facto, maintenu par Denx `http://www.denx.de/wiki/UBoot`
  - Barebox, successeur d'U-boot, mais ne supporte pas encore autant de matériels

U-boot

- U-boot doit être installé dans la mémoire Flash pour être exécuté par le matériel :
  - La carte propose un moniteur de boot qui permet de flasher le bootloader de second niveau (c'est le cas sur la beagleboard)
  - U-boot est déjà installé et peut être utilisé pour se re-flasher (attention, peut rendre la carte inutilisable !)
  - La carte propose une interface JTAG qui permet d'écrire dans la Flash sans aucun system tournant sur la carte.
- Une fois installé, U-boot propose, par l'intermédiaire du port série, un certain nombre de commandes.
- U-boot est alors utilisé pour charger et booter sur une image noyau (`uImage` ou U-boot image), mais il peut aussi changer l'image noyau ou le système de fichier root présent sur le système.
- La communication avec l'hôte se fait par Ethernet (`tftp`) ou le port série

Linux : Séquence de boot traditionnelle

1. Bootloader
   - Exécute par le CPU à une adresse fixe en ROM/Flash
   - Initialize le support du matériel ou se trouve le noyau (Flash, réseau, SD-card...)
   - Charge l'image du noyau dans la RAM
   - Lance l'exécution du noyau
2. Noyau
   - Se décompresse lui-même
   - Initialise le noyau et les pilotes compilés statiquement utilisés pour accéder le système de fichier `root`
   - monte le système de fichier `root`
   - exécute le premier programme utilisateur (spécifié par le parametre `init` du noyau).
3. Premier programme utilisateur
   - Configure l'espace utilisateur et démarre les services systèmes

Mais...

- Cette séquence implique que tous les pilotes de périphériques sont compilés statiquement dans le noyau.
- Cette hypothèse est valide dans le monde embarqué ou le noyau est paramétré en fonction du matériel, elle l'est beaucoup moins pour les PC de bureau.
- Le système de boot actuel contient donc le mécanisme `initramfs` (ex-`initrd`)
  - Un petit système de fichier contenu dans le noyau lui-même
  - Il peut détecter le matériel, charger les modules du noyau nécessaires
  - Il monte alors le système de fichier `root` et exécute l'init

Connaissance pour Linux embarqué

- La mise ne place d'un linux embarqué nécessite de connaître un peu le mécanisme de boot, notamment pour parametrer correctement la commande de lancement du noyau.
- Il faut aussi connaître l'administration linux (gestion de paquets, installation driver, etc.)
- Enfin suivant le type de stockage physique disponible, le système de fichier peut être spécifique
- Pour énormément de plates-formes existantes, les configurations requises sont disponibles dans les outils de construction de chaîne de compilation comme OpenEmbedded

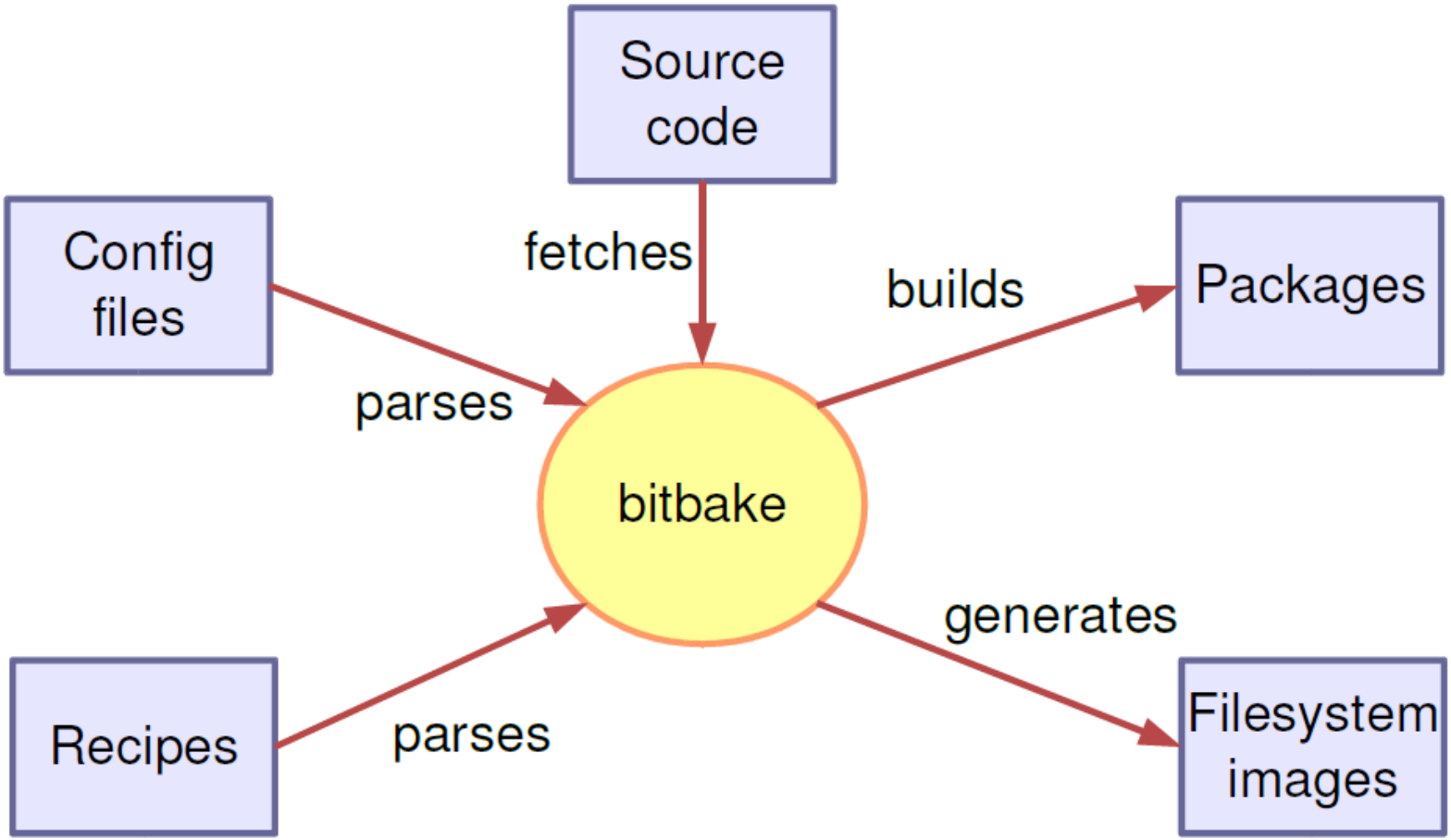
OpenEmbedded

- Initié par les développement mis en commun de la distribution OpenZaurus pour les PDA Sharp Zaurus et de la distribution Familiar pour PDAs.
- Rejoint ensuite par de nombreuses distributions : OpenSimpad, GPE Phone Edition, Ångström, OpenMoko...
- Basé sur `bitbake` : un énorme Makefile mondial dérivé du système de paquetage Gentoo.
- OpenEmbedded contenait en 2011
  - 2097 packages dans `recipes/`
  - 7294 versions de packages dans `recipes/<tool>/*.bb`
  - 305 machines définies dans `conf/machine`
  - 34 distributions définies dans `conf/distro`
- ...et en 2014
  - 2143 packages dans `recipes/`
  - 7226 versions de packages dans `recipes/<tool>/*.bb`
  - 314 machines définies dans `conf/machine`
  - 31 distributions définies dans `conf/distro`

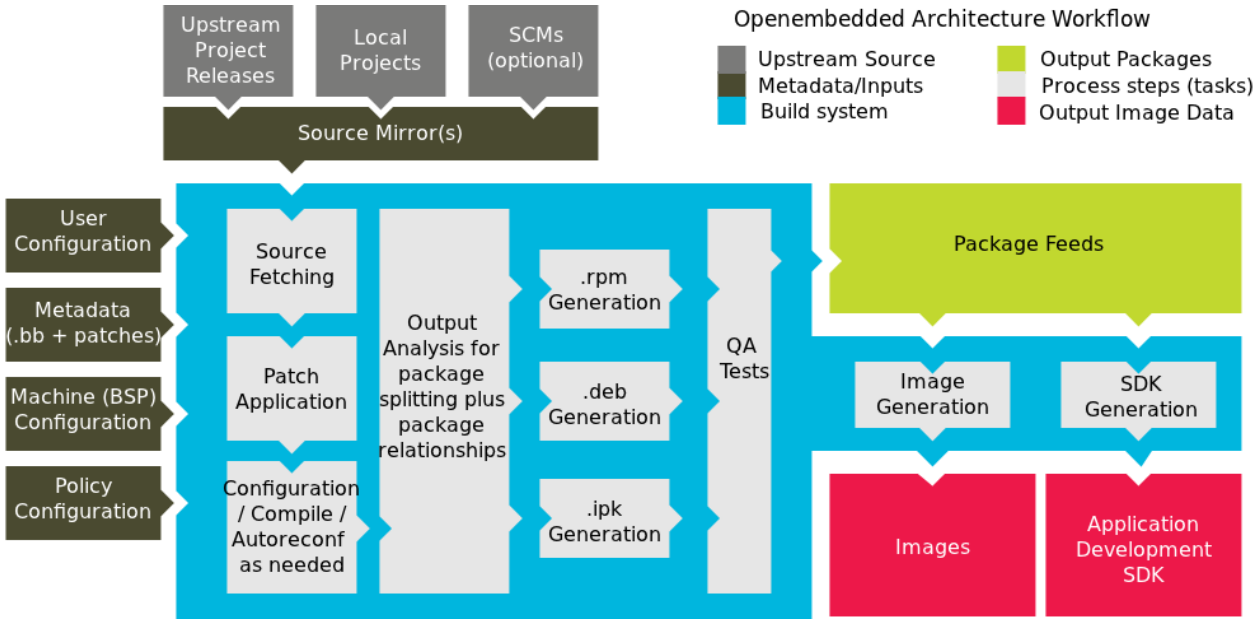
BitBake

- BitBake est le composant permettant à OpenEmbedded d'exister
- Implémenté en python à partir de l'outils de gestion des la distribution Gentoo (`emerge`)
- BitBake peut :
  - Télecharger les sources sur internet (.tgz, svn, git,...)
  - Appliquer les patches
  - Construire les composants
  - Construire le compilateur et le cross-compilateur
  - Configurer, compiler la libc
  - Créer le système de fichier `root`,
  - Créer et installer les packages additionel necessaires (système de gestion de paquets `ipkg`)

BitBake


Source code
Config files
fetches
builds
Packages
parses
bitbake
generates
Recipes
parses
Filesystem images

The Yocto project



- Depuis 2011 le projet Yocto (`https://www.yoctoproject.org/`) et OpenEmbbeded partage une certain nombre de meta-data (openembedded-core).
- Le projet Yocto se focalise sur la mise en place de chaine de compilation solide et facile à utiliser pour un nombre restreint de platformes.

Utilisation d'OpenEmbedded : beaglboard

- Procédure suivie pour la mise en place de ce cours
- Distribution Angstrom : `http://www.angstrom-distribution.org/`
- `git clone git://gitorious.org/angstrom/angstrom-setup-scripts.git` (installe `bitbake` et `openembedded`, tout est fait dans le repertoire `angstrom-setup-scripts`.)
- `export MACHINE=beagleboard`
- `bitbake console-image`
- .....
- Quelques heures et 5 Gigas plus tard ....

Déploiement d'Angstrom avec OpenEmbedded

- dans le repertoire `angstrom-setup-scripts/build/tmp-angstrom_2008_1/deploy/glibc/images/beagleboard`
  - fichiers `u-Boot, uImage`
  - Système de fichier `root` `Angstrom-console-image-glibc-ipk-2010.7-test-20110111-beagleboard.rootfs.tar.bz2`
  - Modules `modules-beagleboard.tgz`
- Une fois tout cela installé sur la carte SD.....

beaglboard booting Angstrom

```
Texas Instruments X-Loader 1.4.4ss (Jul 28 2010 - 16:59:13)
Beagle xM Rev A
Reading boot sector
Loading u-boot.bin from mmc
U-Boot 2010.03 (Aug 06 2010 - 11:28:56)
[....]
reading boot.scr
** Unable to read "boot.scr" from mmc 1:1 **
reading uImage
[...]
   Loading Kernel Image ... OK
OK

Starting kernel ...
Uncompressing Linux.....................................................................
[    0.000000] Linux version 2.6.32 (koen@dominion) (gcc version 4.3.3 (GCC) ) 0
[..............]
Starting GPE display manager: gpe-dm

.-------.
|       |                  .-.
|   |   |-----.-----.-----.| |   .----..-----.-----.
|       |     | __  |  ---'| '--.|  .-'|     |     |
|   |   |  |  |     |---  ||  --'|  |  |  '  | | | |
'---'---'--'--'--.  |-----''----''--'  '-----'-'-'-'
                -'  |
                '---'

The Angstrom Distribution beagleboard ttyS2
Angstrom 2009.X-test-20100104 beagleboard ttyS2

beagleboard login: root
root@beagleboard:~#
```

building linuxes for raspberryPi

- Build a Custom Raspberry Pi Distro with OpenEmbedded & Yocto : `http://www.pimpmypi.com/`
- OpenEmbedded meta-raspberrypi `http://www.raspberrypi.org/forums/viewtopic.php?f=56&t=59282`
- Building XBMC (media player for television) with OpenElec `http://derekmolloy.ie/raspberry-pi/`
- Depuis 2012, nous utilisons dans BED des version préformatées pour raspberryPi de la distribution Debian wheezy (Raspbian) (`http://www.raspberrypi.org/downloads/`)