

TABLE DES MATIÈRES

1	Introduction	2
2	Le choix	3
2.1	A l'origine était le commencement	3
2.2	L'hôte en question	3
2.3	Prérequis	4
2.4	Le choix	4
2.5	C'est parti!	4
2.5.1	les paquets de l'hôte	4
2.5.2	configurer l'hôte : cas du port Série	5
2.5.3	configurer l'hôte : envoyer des fichiers en USB	6
2.6	OpenEmbedded : en plein dedans	6
2.6.1	OpenEmbedded : un peu de cuisine avec Bitbake	6
2.6.2	Installation de OpenEmbedded	8
2.6.3	resumé pour l'homme pressé	11

1 INTRODUCTION



La mini2440 en *Français*

La carte fraîchement sortie de son emballage vous donne envie de développer vous-même vos applications, de la customiser un peu. Après tout, c'est une carte faite pour ça plus qu'un téléphone auquel on ajoute des scripts. Seulement, voilà : c'est pas si simple que ça de se lancer dans le développement embarqué. Il faut expérimenter, apprivoiser la bête, même si c'est "FriendlyArm". Des cartes ARM ce n'est pas nouveau, il y en a plein, des plus ou moins professionnelles. Pourtant celle-là semble déclencher plus d'intérêt parmi les Geeks et développeurs de différents endroits : Angleterre, Allemagne, U.S., Thaïlande, Chine et... la France depuis peu de temps. Malgré cet intérêt croissant et commun, peu de blogs et/ou de tutoriels abordent simplement comment mettre en place un environnement complet pour arriver aux développements dont il est question. C'est le but de tutoriel qui est aussi dédié à être en Français. C'est le résultat d'informations que j'ai pu récolter sur les manuels, sites et aussi mes expériences afin qu'en lisant ce document on évite de perdre du temps à trop se questionner comme je l'ai fait au début.

2.1 A l'origine était le commencement

Ce qui motive en général de faire un tutoriel c'est l'absence d'informations sur un sujet. Ici, paradoxalement c'est la quantité d'informations qui m'a motivé : il faut pas mal de temps pour savoir ce que l'on souhaite vraiment faire. En effet, parmi les possibilités on peut :

- faire son propre bootloader
- customiser son propre noyau Linux
- choisir si l'on veut avoir un serveur d'affichage ou pas
- choisir QUEL serveur d'affichage l'on voudra
- ajouter ou retirer les applications livrées avec la carte
- ajouter/développer ses propres applications
- etc... aussi loin que l'imagination le permette

Si vous lisez ceci c'est que vous avez déjà fait votre choix...sur ce que vous voulez faire de la mini2440 pas forcément COMMENT vous allez le faire. Je m'explique : si vous voulez faire votre propre appli qui rendra fou de jalousie le voisin ou le collègue il va falloir d'abord savoir par quels processus vous allez devoir passer. En gros on veut faire un gâteau et on peut soit utiliser une préparation où il faut juste ajouter un oeuf et un peu de farine, ou le faire complètement soi-même. C'est cette dernière possibilité que j'ai décidé d'aborder ici car de loin elle est la meilleure : on a ainsi la liberté de choisir ce que l'on veut avoir et ce que l'on veut enlever, optimiser et dernir gros avantage : apprendre. Effectivement, l'avantage pédagogique de construire soi-même son système est énorme surtout qu'avec Linux cela vaudra aussi pour un PC.

2.2 L'hôte en question

Dans notre développement embarqué on va utiliser l'ordinateur comme hôte de développement, on pourrait le faire sur la mini2440 mais c'est bien moins pratique finalement car sur le PC la compilation sera moins lourde et puis l'on peut suivre les indications en même temps. L'hôte en question sera donc un PC et la distribution choisie une ubuntu 9.10 Karmic koala. Cela aurait pu en être une autre, mais c'est la plus répandue et donc vous avez plus de chance de l'avoir. Windows à été exclu de choix, pas seulement par sa nature propriétaire mais aussi parce que malgré ce qui est proposé dans la documentation de la FriendlyArm, il y a peu d'intérêt à développer sous Linux avec un hôte Windows dans une machine virtuelle, ça fait beaucoup d'intermédiaires pour rien.

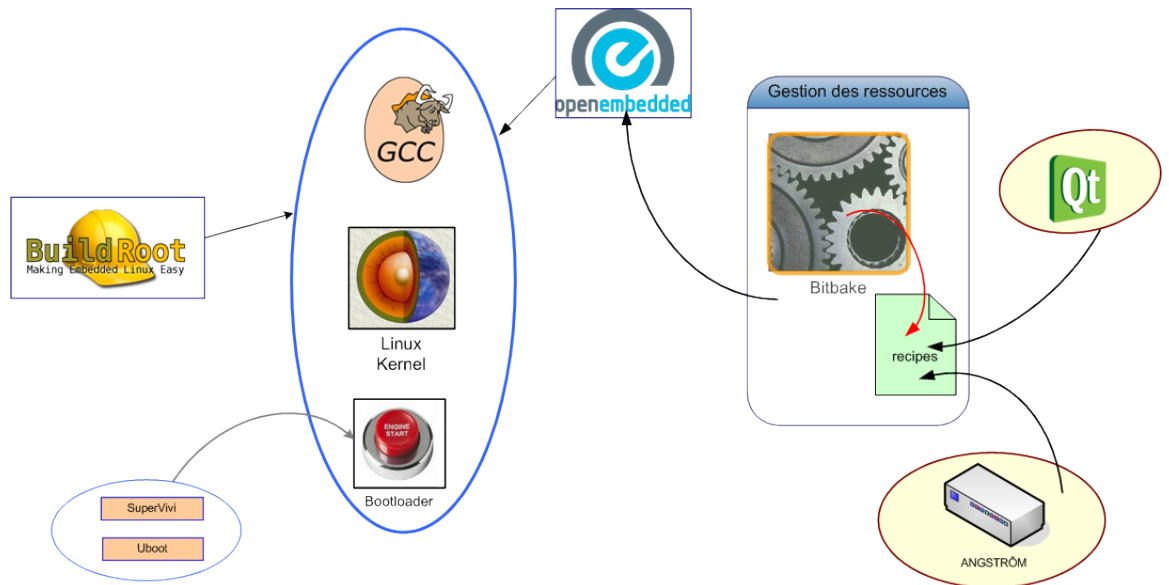
2.3 Prérequis

Même si le but est d'être assez simple, dans ce type de développement il faut avoir et savoir un minimum de choses, je considère donc :

- que vous avez déjà installé votre ubuntu 9.10
- que vous avez des notions de commandes Linux et bash
- que vous avez un minimum d'acquis en électronique
- de la patience de la motivation, et de la patience aussi

Cela dit, ce qui suit est fait pour être réellement simple ; assez pour s'en sortir tout seul.

2.4 Le choix



Comme je l'ai dit trop de choix empêche d'avancer : trop d'informations tue l'information. Eh oui : par quoi on commence ? on installe directement le kernel, on fait comment pour le compiler, et avec quoi ? Crosstools, Buildroot, OpenEmbedded, Emdebian, Angström, etc il y en a partout. Qui fait quoi, c'est là à mon avis le plus délicat à choisir. Il ne sera pas question ici de détailler toutes les possibilités, j'aborderai surtout ce que j'ai expérimenté car c'est ce qui a fonctionné. *Vous emmener du carton d'emballage au sourire de satisfaction à la fin de l'installation.*

Cela n'empêche une petite explication pour éclaircir tout ça.

- La mini2440 s'appuie sur Linux (Kernel/noyau Linux est un pléonasme !) pour fonctionner ainsi quand quelqu'un développe un driver on peut tous l'utiliser

- Ce noyau s'occupe de toutes les activités de la carte, à condition qu'elle ait démarré, c'est ce dont s'occupe le bootLoader ; j'ai donné en exemple Supervivi qui est livré avec la carte et Uboot qui est le plus connu. Il y en a d'autres comme RedBoot de RedHat par exemple
- pour pouvoir installer nos applications il faut pour les développer le compilateur GCC de R.Stallman reste ce qu'il y a de mieux. Avoir un compilateur sur son PC hôte (cross compilateur) reste le mieux

En partant de ces éléments fondamentaux on cerne mieux notre besoin. Cependant, il faut gérer pas mal d'autres choses comme les bibliothèques, les dépendances entre elles bref toutes les ressources qui sont nécessaires à un projet de développement. Et c'est là que des outils comme OpenEmbedded, BuildRoot ou autres deviennent utiles.

Sur le synoptique j'ai voulu resumer cette ambivalence : on peut soit le faire soi même en décompressant un gcc, un kernel etc ... soit passer par un environnement qui va gérer cela et même plus pour nous. En fait on décrit notre besoin, soit par un menu en "faisant notre marché" pour BuildRoot, soit avec un fichier de descriptions (recipe) pour OpenEmbedded.

Pour cela j'ai opté pour OpenEmbedded plutôt que BuildRoot car il a l'avantage d'être plus ouvert par son système de paquetages et malgré ce qui est dit sur Buildroot, j'ai trouvé que la personnalisation y est plus compliquée quand on veut se préparer un système "aux petits oignons", mais cela n'engage que moi.

D'ailleurs on peut trouver deux très bons articles sur Buildroot dans le Linux Mag H.S. 47.

2.5 C'est parti !

Maintenant que les choix sont faits, on se lance.

2.5.1 les paquets de l'hôte

Avant de commencer il faut avoir :

- Git
- bitbake
- et tout une liste que la commande suivante permet d'installer car sur Ubuntu 9.10 ce n'est pas présent
ça donne :

```
sudo apt-get install git bison flex e2fsprogs m4
curl cvs subversion unzip python-psycopy texinfo
texi2html diffstat openjade docbook-utils quilt
groff linuxdoc-tools patch linuxdoc-tools
```

Concernant bitbake, vous pouvez l'installer par cette commande, mais cela n'aura aucun intérêt car on utilisera une version mieux

adaptée que celle livrée dans les dépôts Ubuntu. La version 1.12 de bitbake empêche de compiler les scripts, il faut au moins la 1.18

2.5.2 configurer l'hôte : cas du port Série

On veut pouvoir télécharger nos programmes et aussi dialoguer avec la carte mais depuis pas mal de temps, les ports série sont absents des PC, malgré la volonté des constructeurs de les éradiquer le besoin est toujours présent. C'est pour cela que l'on trouve des convertisseurs USB <->série. Sous Linux il n'y pas (plus) grand chose à installer, on branche le cable qui sera détecté avec le bon driver.

Pour le retrouver on regarde du côté des devices avec :

```
ls /dev/ttyUSB0
```

et non pas /dev/ttyS0 car ce n'est pas un driver COM classique.

Je n'ai pas rencontré de difficulté avec l'Ubuntu mais si jamais votre cable n'était pas détecté, vérifiez que vous avez bien le driver **ftdi** installé

L'autre avantage d'être sous Linux c'est que l'on a tout ce qu'il faut avec nous : trouver un terminal série sous Seven gratuit n'est pas si simple, tandis que sous Linux c'est courant.

J'utilise le bon vieux minicom, mais si vous y êtes allergique, il y à aussi GTKterminal qui fonctionne avec les mêmes paramètres donnés ici.

dans le terminal, on fait :

```
minicom -s
```

la première fois pour configurer, sinon on n'entre pas le "-s"

Avec ses petits doigts musclés on choisi les paramètre suivants :

```
A - Serial Device      : /dev/ttyS0
B - Lockfile Location  : /var/lock
C - Callin Program     :
D - Callout Program    :
E - Bps/Par/Bits       : 115200 8N1
F - Hardware Flow Control : No
G - Software Flow Control : No
```

Vérifiez bien le paramètre -F- (HW flow) car sinon vous ne pourrez pas saisir quoique ce soit.

2.5.3 configurer l'hôte : envoyer des fichiers en USB

Avec le câble USB fourni on va envoyer nos fichiers vers la carte.

C'est pour l'instant la solution utilisée, mais ce n'est pas forcément la meilleure car sous Linux on peut utiliser le NFS que je trouve bien pratique. J'en parlerai si nécessaire dans un prochain tuto, mais sinon il y a plein de sites qui en parlent. Pour l'instant on veut aller rapidement à l'essentiel, il y a déjà pas mal de choses à faire.

Sur le DVD livré avec la carte, on ne trouve que la version windows du logiciel pour télécharger en USB : [DNW](#) , mais grâce à la dynamique de la communauté dont je parlais au début, il y a une version sous Linux disponible

The [Lien USBPush](#)

Le monsieur se rend en console à l'endroit du téléchargement, il va décompresser le fichier en question et il va compiler en installant au préalable la librairie nécessaire (à moins qu'il aime chercher longtemps pourquoi ça ne compile pas) Ca donne :

```
tar vzxvf s3c2410_boot_usb-20060807.tar.bz2
sudo apt-get install libusb-dev
make
cp usbpush /usr/bin
```

2.6 OpenEmbedded : en plein dedans

Bon, à partir de là on a le minimum vital pour s'amuser un peu, et c'est là qu'il va falloir résister pour ne pas s'arracher les cheveux. Mais de quoi je parle?! , en suivant ce qui est dit ici il n'y a aucun problème; il faut vraiment bien suivre par contre parce que le moindre petit écart ne pardonne pas.

Et puis comme il est dit sur les paquets de gateaux :

Malgré tout le soin apporté à la fabrication des Produits, il peut arriver que le produit s'avère défectueux (défaut de fabrication ou de fonctionnement), ou encore qu'une erreur puisse s'y être glissée

Dans ce cas eh bien *RTFM* comme ils disent, ou en termes choisis : il faut prendre les docs de OpenEmbedded

[ICI](#) ou [LA](#)

Le mieux est encore de me prévenir que je modifie ce document en fonction, ça profitera aux autres qui liront après vous.

2.6.1 OpenEmbedded : un peu de cuisine avec Bitbake

bitbake ou comment cuire ses bits

Si vous vous rappelez les explications sur OpenEmbedded et les environnements de cross-compilation (si vous ne vous rappelez pas, c'est pas grave il

y a des remèdes pas mal pour l'Alzheimer maintenant), on a vu que Openembedded s'appuie sur BitBake pour mettre en place (i.e. : télécharger, installer et tout ce qui faut) l'environnement avec ses dépendances.

A propos de Bitbake

Bitbake est un script Python dérivé de Portage, le système de gestion des paquets utilisé par Gentoo assurant la gestion des dépendances d'un paquet : il se charge de les télécharger et les construire automatiquement si nécessaire. Bitbake représente la base de OpenEmbedded.

A propos des recettes de BitBake

Le fichier de recette est dans un répertoire recipes avec l'extension .BB
C'est le moyen pour décrire comment construire un paquet particulier. Cela inclut les dépendances du paquet, où trouver les sources de ce paquet, sa configuration, sa compilation.

cela donne un truc de la forme:

```
bitbake -b PifPof_1.0.bb
```

Quand je vous disais que c'est du gateau !!

Installation de bitbake

```
cd /home/david
mkdir oe
cd oe
mkdir build
mkdir build/conf
mkdir sources
mkdir tmp
wget http://download.berlios.de/bitbake/bitbake-1.8.18.tar.gz
tar zxvf bitbake-1.8.18.tar.gz
sudo mv bitbake-1.8.18 /usr/bin/bitbake
```

2.6.2 Installation de OpenEmbedded

```
cd /home/david/oe
git clone git://repo.or.cz/openembedded/mini2440.git
```

Certains disent que l'on a le temps de prendre un café après cette commande. Je trouve cela exagéré. J'ai à peine eu le temps de prendre mon café, après avoir regardé la trilogie Star-wars

le fichier de configuration

```
cp mini2440/mini2440_local_conf_example.conf ../build/conf/local.conf
```

On édite ce fichier et on prend le temps de bien vérifier que les chemins sont ok, particulièrement :

```
DL_DIR = "/home/david/oe/sources"
.....
BBFILES = "/home/david/oe/recipes/*/*.bb"
.....
# Uncomment this if you're building for an arch
# that uses emulated locale
# generation under qemu (mainly arm glibc) and
# have an external gcc 3.x compiler
# that OE recognises. This will mean the gcc-
# native build is skipped, speeding
# builds up.
#ASSUME_PROVIDED += "gcc3-native"
#ASSUME_PROVIDED += "e2fsprogs-native"
#ASSUME_PROVIDED += "git-native"
#ASSUME_PROVIDED += "m4-native"
#ASSUME_PROVIDED += "bison-native"
#ASSUME_PROVIDED += "flex-native"
#ASSUME_PROVIDED += "curl-native"
#ASSUME_PROVIDED += "ncurses-native"
#ASSUME_PROVIDED += "zlib-native"
#ASSUME_PROVIDED += "cvs-native"
.....
TMPDIR = /home/david/oe/tmp
```

Le mieux est de commenter ces restrictions, sinon on perd pas mal de temps dans les forums pour comprendre pourquoi la compilation bloque

L'homme de la Pampa, parfois rude, reste toujours courtois. Mais la vérité m'oblige à te le dire : ton Antoine commence à me les briser menu!

C'est pas parce que je mache tout qu'il faut juste répéter. Le fichier de conf comporte un moyen de vérifier que vous avez bien regardé et compris et comme j'approuve la méthode je vous laisse trouver. C'est très simple et au pire vous aurez un message lors de la compil qui s'arrêtera net.

Les variables d'environnement

On édite le fichier de conf du compte Linux local

```
sudo vim /home/david/.bashrc
```

et à la fin (c'est mieux et propre) on ajoute son chemin

```
BBPATH=/home/david/oe/:/home/david/oe/build  
PATH=\$PATH:/home/david/oe/bitbake/bin
```

On sauvegarde et on sort. Pour le prendre en compte :

```
source ~/.bashrc
```

Cette méthode est à mon avis bien mieux que celle que l'on peut voir ailleurs car un simple `EXPORT BBPATH` ou faire un `SOURCE FICHER_VARS` ne sont que des solutions ponctuelles : avec elles votre système ne fonctionnera plus au prochain démarrage car vous aurez perdu vos variables d'environnement

Pour les utilisateurs Ubuntu, il faut vérifier que l'on utilise bien bash et pas dash. Pour cela, :

```
sudo dpkg-reconfigure dash
```

et on répond : "NO"

DASH

DASH (the Debian Almquist Shell) remplace bash depuis Edgy. Ce remplacement est annoncé pour " améliorer les performances " car selon les développeurs de Canonical, bash crée trop d'instances au démarrage étant alors trop gourmand ce que dash est censé corriger.

Dernière étape : la construction

Courage, c'est quasiment fini car on a fait le plus difficile reste à lancer la construction de notre système.

Si vous voulez avoir QT4 parmi vos paquets, faites-donc :

```
vim /home/david/oe/recipes/images/mini2440-image.bb
```

pour modifier le fichier et rajouter qt4-embedded

```
IMAGE_INSTALL = "task-base-extended
\${ANGSTROM_EXTRA_INSTALL}
psplash-zap qt4-embedded
esekeyd u-boot-utils tslib-calibrate tslib-tests
i2c-tools i2c screen rsync nfs-utils-client
directfb gdbserver directfb mtd-utils
rsvg pango \"
```

Maintenant on se lance :

- retournez dans le répertoire "build" du début
- lancez la commande :

```
bitbake -v task-base
```

Il y a d'autres images comme X11, GPE OPIE mais comme on a précisé QT4 et que l'on veut le minimum celle-ci suffit

Après un autre long moment, vous êtes en mesure de faire votre compilation croisée puisque vous avez maintenant les compilateurs ad-hoc et toutes les bibliothèques nécessaires.

Alors soyons fous :

```
bitbake -v console-image
```

"C'est la fin, oui mon ami la fin"

Et voilà, vous avez enfin tout ce que vous aviez besoin réunis sous tous les formats utiles, on peut sortir le Champomy et appeler les fi ah oui! j'ai oublié de vous dire où l'on trouve nos images :

C'est simplement là où l'on a déclaré notre répertoire temporaire, c'est à dire :

```
/home/david/oe/tmp/deploy/glibc/images/mini2440
```

"C'est quoi cette bouteille de lait?"

"OK mais on en fait quoi de ces images?, pourquoi il y en à 3?, etc" vous demandez-vous l'oeil agard et les mains tremblantes tel un Mick Jaeger en pleine transe après son tilleul-menthe quotidien.

- console-image.jffs2 pour flasher directement dans la NAND de la carte avec le bootloader
- console-image.tar.gz pratique car le rootfs est déjà inclus (utile aussi en NFS). Pour l'utiliser avec une carte SD, créez une partition dessus et :

```
tar xzvfp /home/david/oe/tmp/deploy/  
glibc/images/mini2440/console-image.  
tar.gz /mnt/mmc1/
```

- console-image.ext3 est orienté partition et peut se mettre sur une carte SD avec un

```
dd if=console-image.ext3 of=/dev/mmc1
```

A noter que sur Karmic Koala il est rapporté des soucis de montage pour la SD, la plupart sur portables.

2.6.3 résumé pour l'homme pressé

```
cd /home/david  
mkdir oe  
cd oe  
mkdir build  
mkdir build/conf  
mkdir sources  
mkdir tmp  
wget http://download.berlios.de/bitbake/bitbake  
-1.8.18.tar.gz  
tar zvxf bitbake-1.8.18.tar.gz  
mv bitbake-1.8.18 bitbake
```

```
cd /home/david/oe  
git clone git://repo.or.cz/openembedded/mini2440.  
git
```

```
cp mini2440/mini2440_local_conf_example.conf ../  
build/conf/local.conf
```

```
sudo vim /home/david/.bashrc
```

```
BBPATH=/home/david/oe/:/home/david/oe/build
```

```
source ~/.bashrc
```

```
sudo dpkg-reconfigure dash
```

```
vim openembedded/recipes/images/mini2440-image.bb
```

```
IMAGE_INSTALL = "task-base-extended  
\${ANGSTROM_EXTRA_INSTALL}  
psplash-zap qt4-embedded  
esekeyd u-boot-utils tslib-calibrate tslib-tests  
i2c-tools i2c screen rsync nfs-utils-client  
directfb gdbserver directfb mtd-utils  
rsvg pango"
```

```
bitbake -v task-base
```

```
bitbake -v console-image
```

```
/home/david/oe/tmp/deploy/glibc/images/mini2440
```

BIBLIOGRAPHIE

[1] Cette section va disparaitre