Systèmes d'exploitation pour l'embarqué

UV 5.2 - Exécution et Concurrence

Paul Blottière

ENSTA Bretagne

2018 / 2019

https://github.com/pblottiere

Amélioration continue

Contributions

- Dépôt du cours : `https://github.com/pblottiere/embsys`
- Souhaits d'amélioration, erreurs, idées de TP, ... : ouverture d'Issues
- Apports de corrections : Pull Request

Linux embarqué : compilation et outils

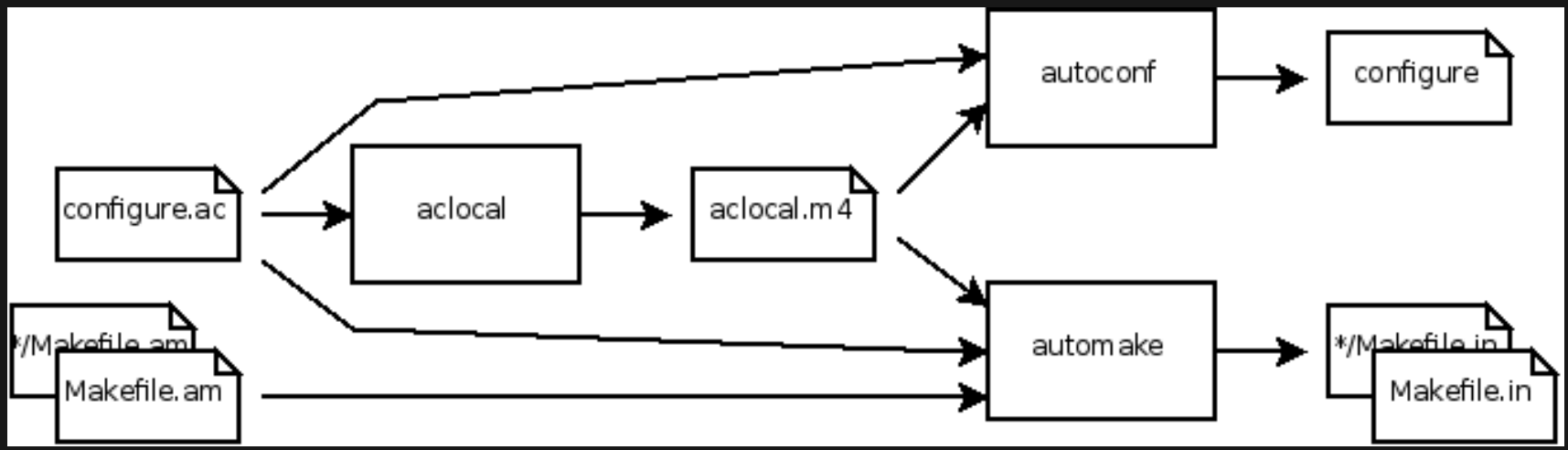
Plan

1. Autotools et menuconfig
2. Une distribution minimale
3. Cross-compilation
4. Compilation du kernel
5. Busybox
6. U-Boot
7. Automatisation (Buildroot, Yocto Project, ...)
8. QEMU

Autotools et menuconfig (1)

Les autotools

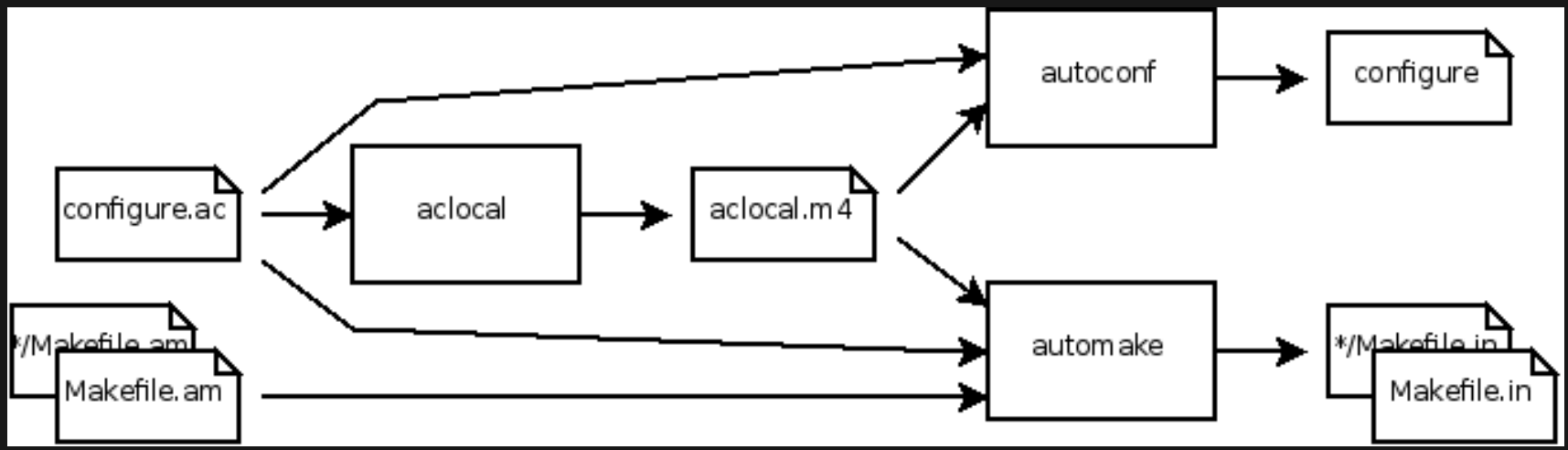
Ensemble d’outils de build du projet GNU.



Autotools et menuconfig (1)

Les autotools

Ensemble d'outils de build du projet GNU.



=> très utilisé, aux côté de CMake!

Autotools et menuconfig (2)

Les autotools

Utilisation classique sans paramétrage :

```
> ./configure
...
> make
...
> sudo make install
...
```

Autotools et menuconfig (2)

Les autotools

Utilisation classique sans paramétrage :

```
> ./configure
...
> make
...
> sudo make install
...
```

=> par défaut, l'installation se fera avec le prefix / :

- /usr/bin
- /usr/lib
- /usr/share
- ...

Autotools et menuconfig (3)

Les autotools

La phase de configuration, via l'exécution de *./configure*, prend de nombreux paramètres en entrée :

- - *-build* : système d'exécution courant
- - *-host* : système où le résultat de la compilation sera exécuté
- - *-prefix* : prefix d'installation utilisé lors du *make install*
- - *-enable-<FEATURE>* : active la fonctionnalité
- - *-disable-FEATURE* : désactive la fonctionnalité
- - *-help* : affiche la liste des commandes disponibles
- ...

Autotools et menuconfig (4)

Les autotools

Un des rôles du script *configure* est de vérifier la présence de toutes les dépendances et de construire les Makefile associés à partir des Makefile.in

Autotools et menuconfig (4)

Les autotools

Un des rôles du script *configure* est de vérifier la présence de toutes les dépendances et de construire les Makefile associés à partir des Makefile.in

```
> ./configure
checking build system type... x86_64-pc-linux-gnu
checking host system type... x86_64-pc-linux-gnu
...
checking for ncursesw/curses.h... no
checking ncurses.h usability... yes
checking ncurses.h presence... yes
checking for ncurses.h... yes
configure: creating ./config.status
config.status: creating Makefile
```

Autotools et menuconfig (5)

Menuconfig

ncurses / New Curses : API de développement d'interface graphique simple en mode texte et exécutable dans un terminal.

Autotools et menuconfig (5)

Menuconfig

ncurses / New Curses : API de développement d'interface graphique simple en mode texte et exécutable dans un terminal.

kconfig : langage de configuration développé initialement par Linus Torvalds pour configurer le kernel à travers une IHM utilisant ncurses.

Autotools et menuconfig (5)

Menuconfig

ncurses / New Curses : API de développement d'interface graphique simple en mode texte et exécutable dans un terminal.

kconfig : langage de configuration développé initialement par Linus Torvalds pour configurer le kernel à travers une IHM utilisant ncurses.

=> désormais, kconfig est utilisé par beaucoup d'autre projets de par sa légèreté et sa facilité d'utilisation :

- kernel
- busybox
- crosstool-ng
- cross-builder
- ...

Autotools et menuconfig (6)

Menuconfig

Par héritage, le lancement d'une IHM de configuration utilisant Kconfig se fait en exécutant la commande :

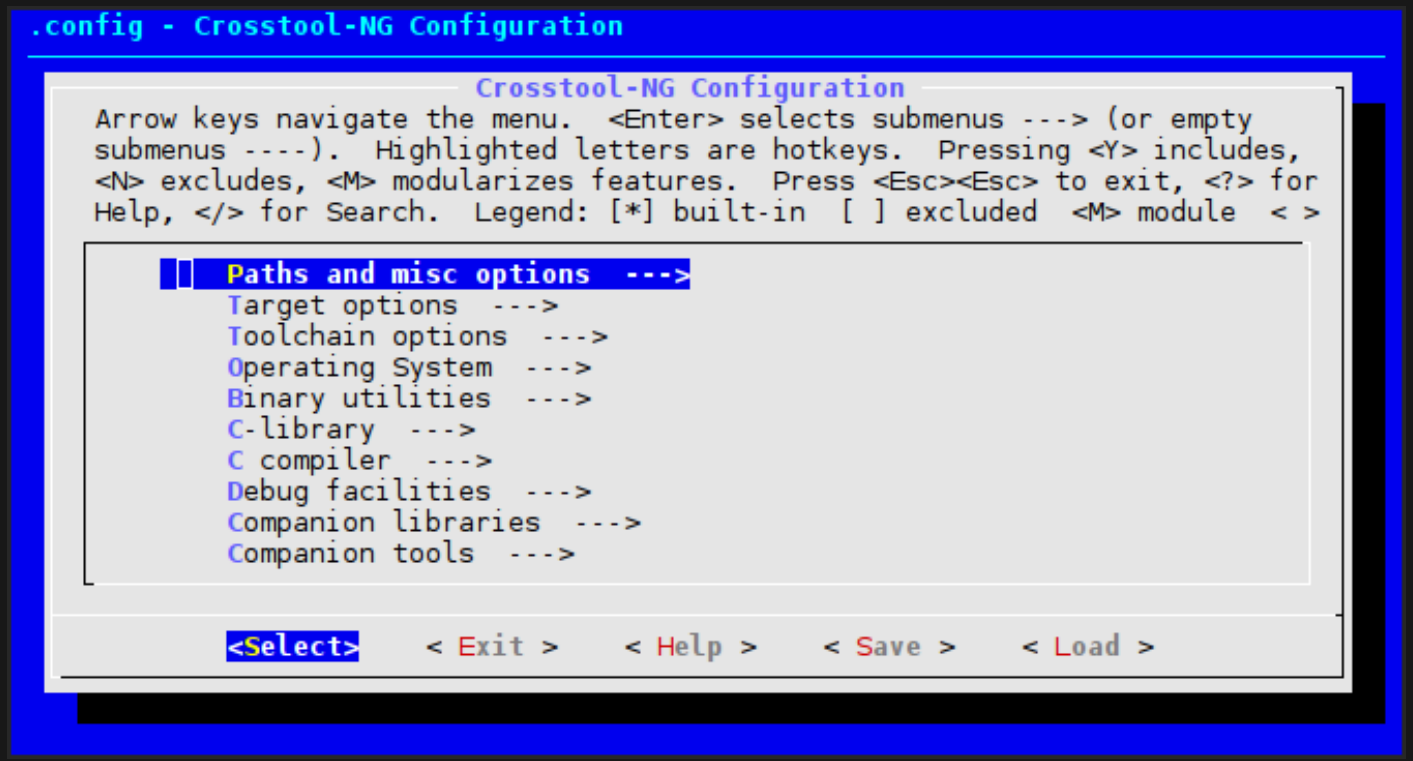
```
> make menuconfig
```

Autotools et menuconfig (6)

Menuconfig

Par héritage, le lancement d'une IHM de configuration utilisant Kconfig se fait en exécutant la commande :

```
> make menuconfig
```



Autotools et menuconfig (7)

Menuconfig

En sortie, un fichier de configuration, utilisé au moment du make, est généré à partir des éléments configurés dans l'interface :

```
CONFIG_OID_REGISTRY=m
CONFIG_UCS2_STRING=y
CONFIG_FONT_SUPPORT=y
# CONFIG_FONTS is not set
CONFIG_FONT_8x8=y
CONFIG_FONT_8x16=y
CONFIG_ARCH_HAS_SG_CHAIN=y
CONFIG_ARCH_HAS_PMEM_API=y
```

Autotools et menuconfig (7)

Menuconfig

En sortie, un fichier de configuration, utilisé au moment du make, est généré à partir des éléments configurés dans l'interface :

```
CONFIG_OID_REGISTRY=m
CONFIG_UCS2_STRING=y
CONFIG_FONT_SUPPORT=y
# CONFIG_FONTS is not set
CONFIG_FONT_8x8=y
CONFIG_FONT_8x16=y
CONFIG_ARCH_HAS_SG_CHAIN=y
CONFIG_ARCH_HAS_PMEM_API=y
```

=> il existe souvent des fichiers de configuration prédéfinis pour des buts bien particuliers!

Autotools et menuconfig (8)

Menuconfig

Un exemple de script utilisant le langage KConfig :

```
config MODVERSIONS
        bool "Set version info on module symbols"
        depends on MODULES
        help
        Usually, modules have to be recompiled
        whenever you switch to a new kernel.  ...
```

Une distribution minimale (1)

Kernel et RFS

Une distribution minimale basée sur le noyau Linux nécessite seulement deux éléments :

- un kernel
- un RFS contenant les binaires et les librairies de base

Une distribution minimale (2)

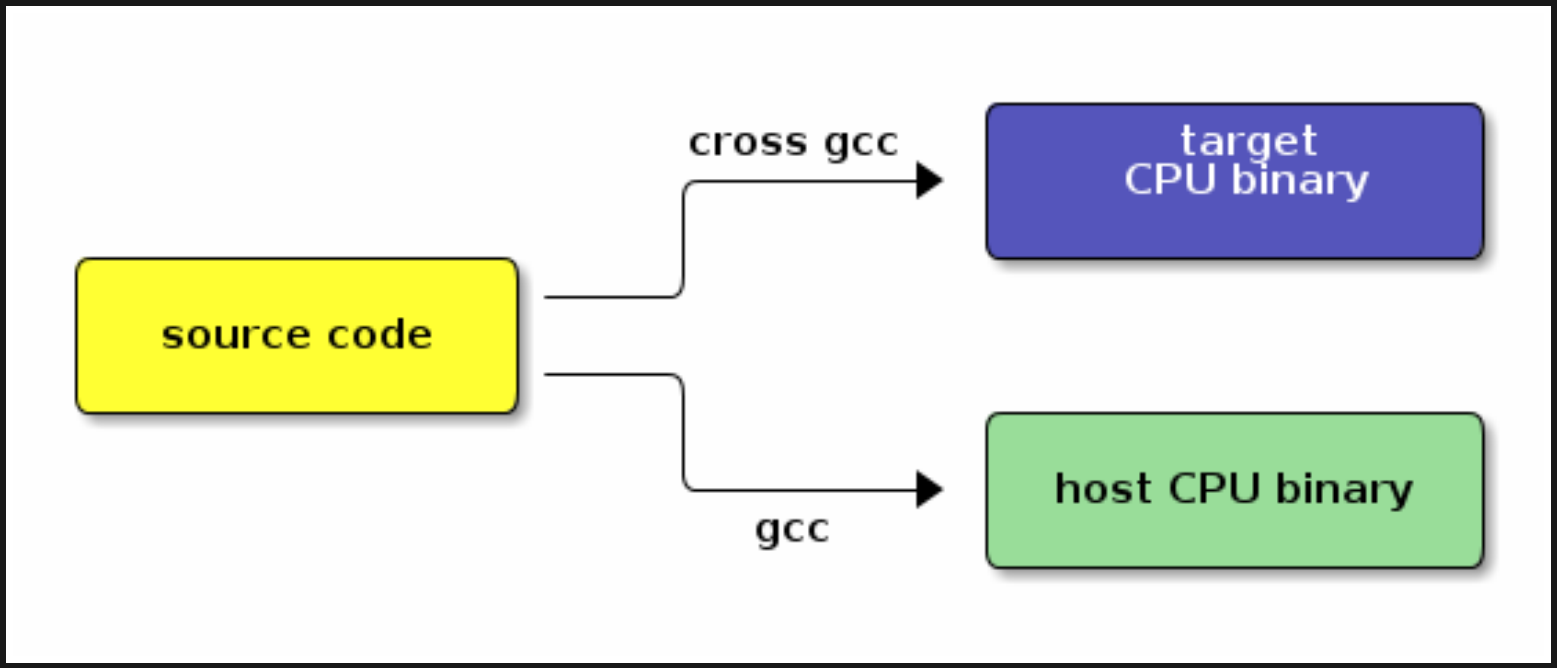
Cross compiler

Pour compiler l'ensemble, un élément est indispensable : **un compilateur croisé**.

Une distribution minimale (2)

Cross compiler

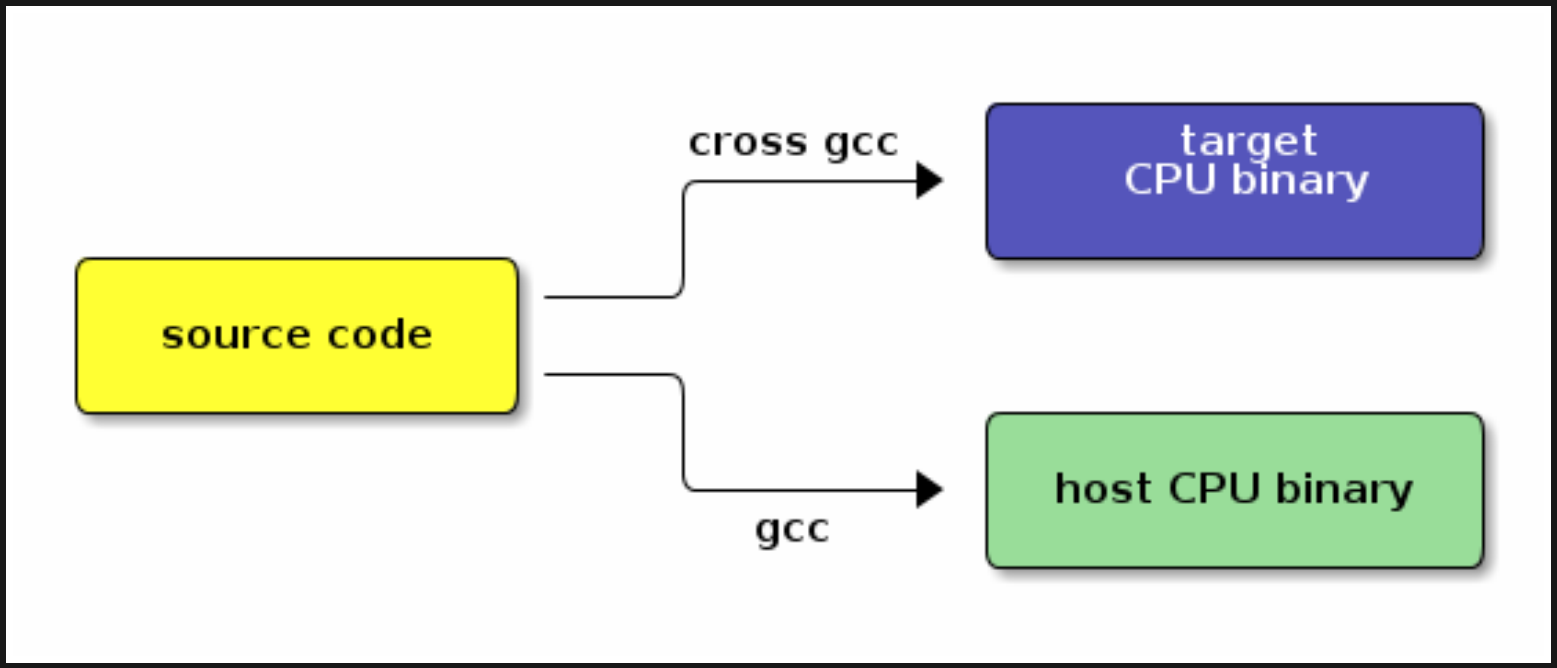
Pour compiler l'ensemble, un élément est indispensable : **un compilateur croisé**.



Une distribution minimale (2)

Cross compiler

Pour compiler l'ensemble, un élément est indispensable : **un compilateur croisé**.



=> la première étape est donc de compiler le compilateur croisé.

Compilateur croisé (1)

Contenu

Une chaîne de compilation est difficile à mettre en oeuvre from scratch car elle contient de très nombreux éléments :

- le compilateur en tant que tel : gcc-<ARCH>
- les outils fournis par GNU Binutils : ld, as, nm, ...
- la libc : glibc, uclibc ou eglibc

Compilateur croisé (1)

Contenu

Une chaîne de compilation est difficile à mettre en oeuvre from scratch car elle contient de très nombreux éléments :

- le compilateur en tant que tel : gcc-<ARCH>
- les outils fournis par GNU Binutils : ld, as, nm, ...
- la libc : glibc, uclibc ou eglibc

Il existe des chaînes sur étagère, robustes et éprouvées :

- Crosstool (vieillisant)
- Crosstool-NG
- la chaîne de ELDK
- la chaîne de Buildroot
- la chaîne de Yocto Project

Compilateur croisé (2)

Crosstool-NG

Auteur : Yann Morin / Fançais / ENIB.

Compilateur croisé (2)

Crosstool-NG

Auteur : Yann Morin / Fançais / ENIB.

Les architectures supportées : ARM, AVR, PPC, x86, ...

Compilateur croisé (2)

Crosstool-NG

Auteur : Yann Morin / Fançais / ENIB.

Les architectures supportées : ARM, AVR, PPC, x86, ...

À travers la configuration de Crosstool-ng, on peut choisir l'architecture cible, la version de gcc, la libc, la version de la libc, ...

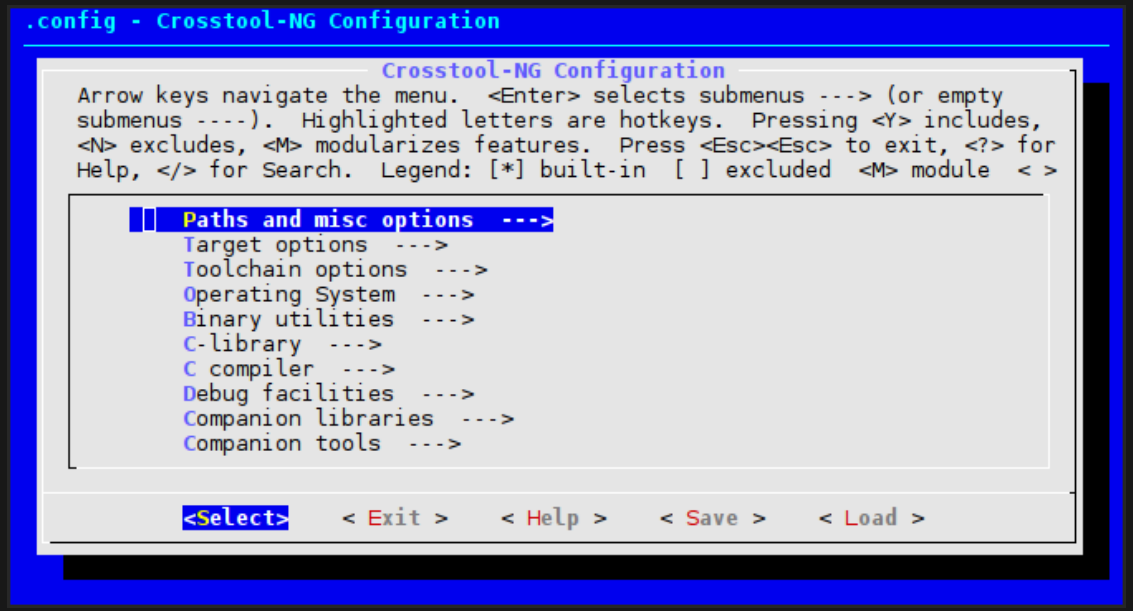
Compilateur croisé (2)

Crosstool-NG

Auteur : Yann Morin / Fançais / ENIB.

Les architectures supportées : ARM, AVR, PPC, x86, ...

À travers la configuration de Crosstool-ng, on peut choisir l'architecture cible, la version de gcc, la libc, la version de la libc, ...



Compilateur croisé (3)

ELDK

Embedded Linux Development Kit :

- chaîne de compilation croisée pour PPC, ARM ou MIPS
- distribution Linux associée à l'architecture cible

Compilateur croisé (3)

ELDK

Embedded Linux Development Kit :

- chaîne de compilation croisée pour PPC, ARM ou MIPS
- distribution Linux associée à l'architecture cible

=> on peut très bien n'utiliser que la chaîne de compilation et construire nous même la distribution.

Compilateur croisé (3)

ELDK

Embedded Linux Development Kit :

- chaîne de compilation croisée pour PPC, ARM ou MIPS
- distribution Linux associée à l'architecture cible

=> on peut très bien n'utiliser que la chaîne de compilation et construire nous même la distribution.

=> la chaîne de compilation de ELDK vient avec des librairies précompilées. On ne peut donc pas choisir les versions de gcc ou de la glibc

Compilateur croisé (3)

ELDK

Embedded Linux Development Kit :

- chaîne de compilation croisée pour PPC, ARM ou MIPS
- distribution Linux associée à l'architecture cible

=> on peut très bien n'utiliser que la chaîne de compilation et construire nous même la distribution.

=> la chaîne de compilation de ELDK vient avec des librairies précompilées. On ne peut donc pas choisir les versions de gcc ou de la glibc

Compilateur croisé (4)

ISA et ABI

Instruction Set Architecture : définit une interface de communication entre une brique logicielle et la partie matérielle.

Compilateur croisé (4)

ISA et ABI

Instruction Set Architecture : définit une interface de communication entre une brique logicielle et la partie matérielle.

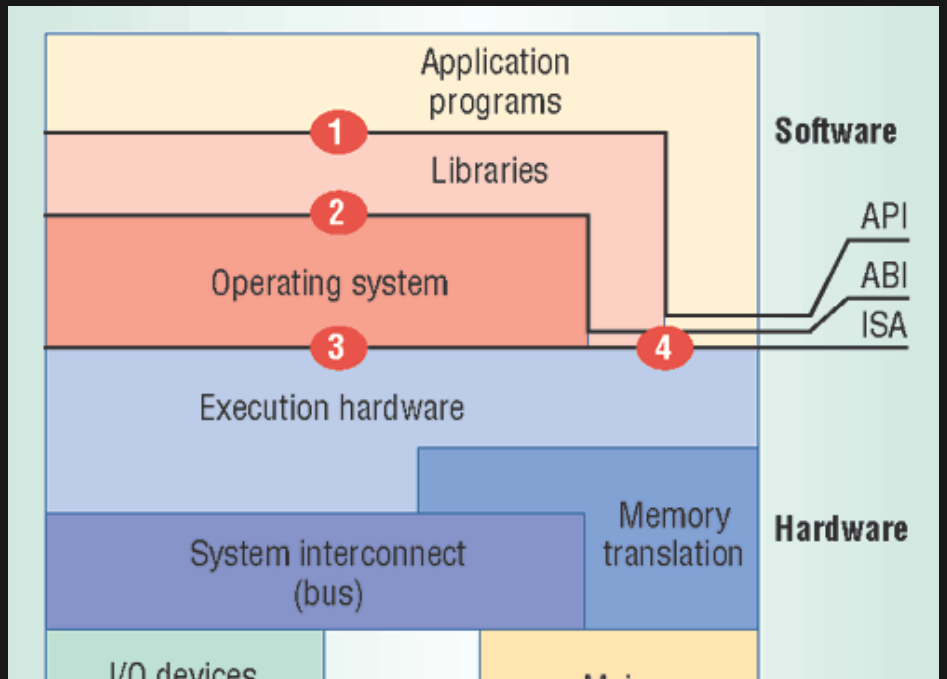
Application Binary Interface : définit au niveau binaire une interface de communication entre plusieurs briques logicielles (librairies, OS, ...).

Compilateur croisé (4)

ISA et ABI

Instruction Set Architecture : définit une interface de communication entre une brique logicielle et la partie matérielle.

Application Binary Interface : définit au niveau binaire une interface de communication entre plusieurs briques logicielles (librairies, OS, ...).



Compilateur croisé (5)

ISA et ABI

Il existe plusieurs ABI par architecture. On doit donc spécifier l'ABI souhaitée au moment de la compilation de la chaîne de cross-compilation!

Compilateur croisé (5)

ISA et ABI

Il existe plusieurs ABI par architecture. On doit donc spécifier l'ABI souhaitée au moment de la compilation de la chaîne de cross-compilation!

Le choix de l'ABI a une conséquence directe sur la taille du code généré, l'utilisation de la mémoire, ...

Compilateur croisé (5)

ISA et ABI

Il existe plusieurs ABI par architecture. On doit donc spécifier l'ABI souhaitée au moment de la compilation de la chaîne de cross-compilation!

Le choix de l'ABI a une conséquence directe sur la taille du code généré, l'utilisation de la mémoire, ...

Par exemple pour ARM :

- OABI : Old-ABI
- EABI : Embedded-ABI

Compilateur croisé (6)

FPU

Floating Point Unit : unité de calcul en virgule flottante.

=> dans un processeur, il peut y avoir une partie dédiée aux opérations sur flottant.

Compilateur croisé (6)

FPU

Floating Point Unit : unité de calcul en virgule flottante.

=> dans un processeur, il peut y avoir une partie dédiée aux opérations sur flottant.

Si un processeur ne possède pas de FPU (dit *hardware FPU*), le calcul sur nombres flottant est exécuté logiciellement par le compilateur lors de la génération en langage machine.

=> on parle alors non pas de **hardfpu**, mais de **softfpu**!

Compilateur croisé (7)

OABI vs EABI : un problème de FPU

OABI part du principe que le processeur possède une FPU. Or, toutes les cartes sous ARM n'en possèdent pas nécessairement.

Compilateur croisé (7)

OABI vs EABI : un problème de FPU

OABI part du principe que le processeur possède une FPU. Or, toutes les cartes sous ARM n'en possèdent pas nécessairement.

Dans le cas d'une demande d'un calcul sur flottant avec une EABI mais sur un processeur n'ayant pas de FPU, une exception sera levée par le processeur, puis attrapée par le kernel qui va lui même utiliser sa fonction de *softfpu*.

Compilateur croisé (7)

OABI vs EABI : un problème de FPU

OABI part du principe que le processeur possède une FPU. Or, toutes les cartes sous ARM n'en possèdent pas nécessairement.

Dans le cas d'une demande d'un calcul sur flottant avec une EABI mais sur un processeur n'ayant pas de FPU, une exception sera levée par le processeur, puis attrapée par le kernel qui va lui même utiliser sa fonction de *softfpu*.

=> terriblement inefficace car même chose sur tous les calculs sur les flottants!

Compilateur croisé (8)

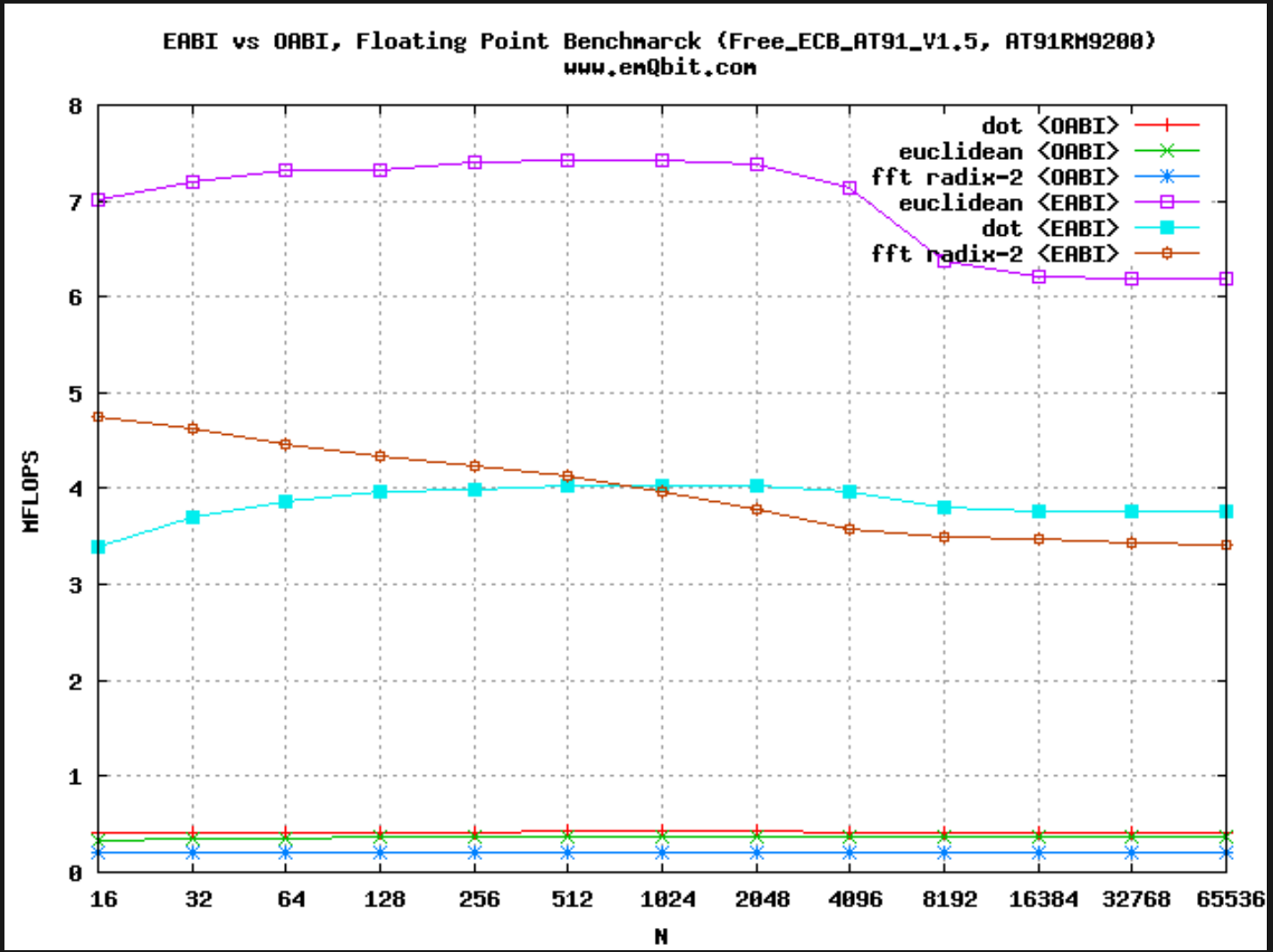
OABI vs EABI : un problème de FPU

EABI permet de gérer la fonction de *softfpu* dans l'espace utilisateur au lieu de l'espace kernel, ce qui augmente les performances.

=> pour cela, il faut passer des options à gcc au moment de la compilation.

Compilateur croisé (9)

OABI vs EABI : un problème de FPU



[Benchmark OABI vs EABI]

Compilation du kernel (1)

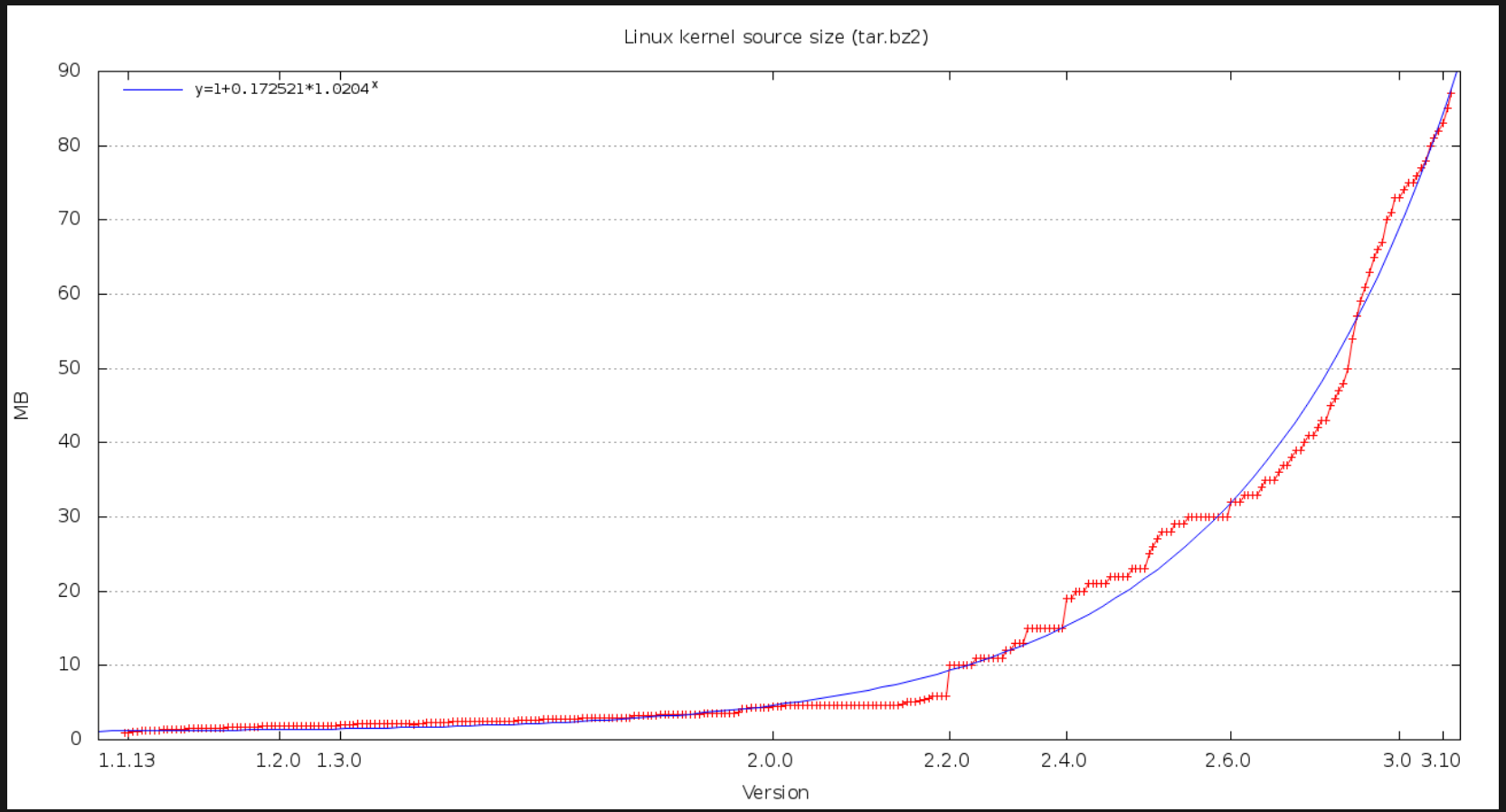
Les sources

Le code source de Linux peut être récupéré sur le github de Mr Torvalds : https://github.com/torvalds/linux.

Compilation du kernel (1)

Les sources

Le code source de Linux peut être récupéré sur le github de Mr Torvalds : https://github.com/torvalds/linux.



Compilation du kernel (2)

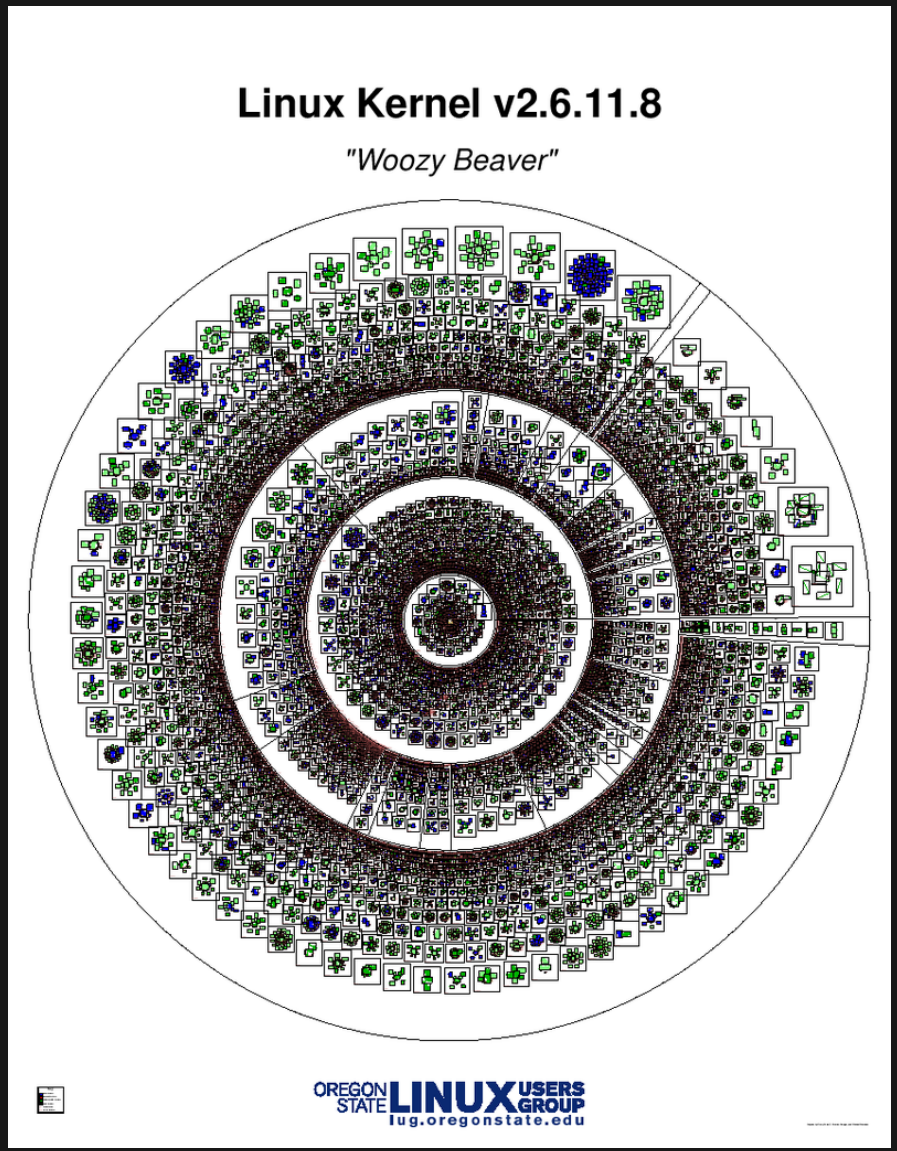
Arborescence

Les répertoires principaux :

- arch : code spécifique aux architecture matérielles
- Documentation : informations au format texte
- drivers : les pilotes de périphériques (i2c, gpio, ...)
- include : les headers
- kernel : les sources du kernel à proprement parlé
- net : le code des couches réseau
- ...

Compilation du kernel (3)

Les couches



[Kernel sources]

Compilation du kernel (4)

Configuration

Une configuration par défaut est disponible pour les différentes architectures matérielles : ce sont les fichiers <ARCH>_defconfig.

Compilation du kernel (4)

Configuration

Une configuration par défaut est disponible pour les différentes architectures matérielles : ce sont les fichiers <ARCH>_defconfig.

Par exemple :

- arch/x86/configs/i386_defconfig
- arch/x86/configs/x86_64_defconfig
- arch/arm/configs/mini2440_defconfig
- arch/powerpc/configs/ppc40x_defconfig

Compilation du kernel (5)

Configuration

Pour utiliser une configuration d'une architecture différente de celle courante :

```
> uname -a
Linux debian 4.2.0-1-amd64 4.2.6-1 x86_64 GNU/Linux
> make sunxi_defconfig
***
*** Can't find default configuration
*** "arch/x86/configs/sunxi_defconfig"!
***
scripts/kconfig/Makefile:108: recipe for target
'sunxi_defconfig' failed
> make ARCH=arm sunxi_defconfig
#
# configuration written to .config
#
```

Compilation du kernel (6)

Compilation

Une fois configuré, on peut compiler le kernel :

```
> make ARCH=arm CROSS_COMPILE=/path/to/compiler
...
BUILD   arch/x86/boot/bzImage
Setup is 15708 bytes (padded to 15872 bytes).
System is 6008 kB
CRC 734d60c8
Kernel: arch/x86/boot/bzImage is ready  (#1)
```

Compilation du kernel (6)

Compilation

Une fois configuré, on peut compiler le kernel :

```
> make ARCH=arm CROSS_COMPILE=/path/to/compiler
...
BUILD   arch/x86/boot/bzImage
Setup is 15708 bytes (padded to 15872 bytes).
System is 6008 kB
CRC 734d60c8
Kernel: arch/x86/boot/bzImage is ready  (#1)
```

=> le fichier **bzImage** est la partie statique du kernel!

Compilation du kernel (7)

Les modules

Les modules (fichiers .ko) doivent être installés sur un RFS :

```
> mkdir fake_rfs
> make modules_install INSTALL_MOD_PATH=./fake_rfs
scripts/kconfig/conf  --silentoldconfig Kconfig
INSTALL crypto/echainiv.ko
INSTALL drivers/thermal/x86_pkg_temp_thermal.ko
...
INSTALL net/netfilter/xt_nat.ko
DEPMOD  4.4.0-rc5
> ls fake_rfs/lib/modules/4.4.0-rc5/
... kernel ... modules.alias modules.dep ...
> ls fake_rfs/lib/modules/4.4.0-rc5/kernel/
crypto  drivers  fs  net
```

Busybox (1)

Qu'est ce?

Dans le cadre d'un système embarqué, l'environnement d'outils et d'applications fournis à travers le RFS doit :

- être de volume réduit
- avoir une consommation en ressources réduite
- être portable sur diverses architectures matérielles

Busybox (1)

Qu'est ce?

Dans le cadre d'un système embarqué, l'environnement d'outils et d'applications fournis à travers le RFS doit :

- être de volume réduit
- avoir une consommation en ressources réduite
- être portable sur diverses architectures matérielles

=> pour cela, le projet Busybox est utilisé dans quasiment tous les équipements basés sur des distributions customisées.

Busybox (2)

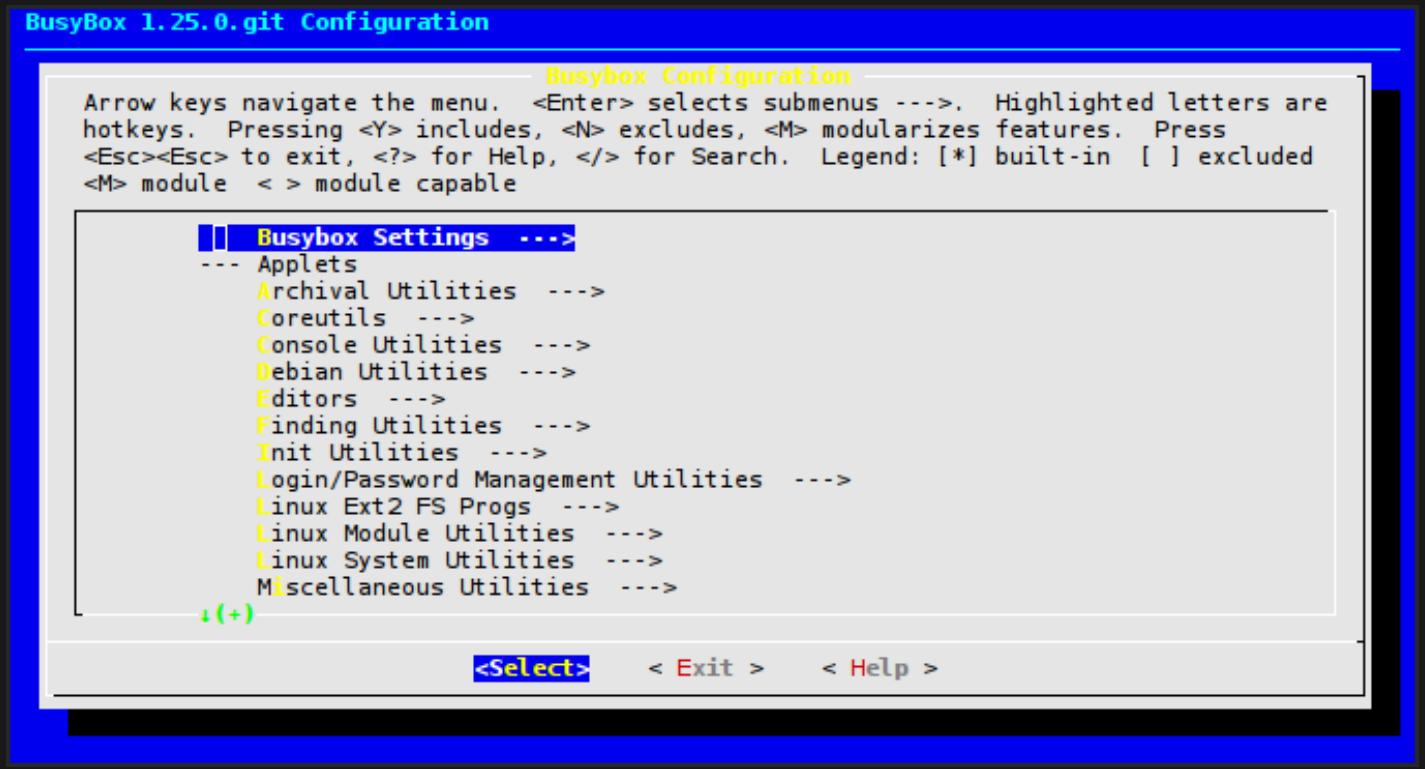
Configuration

La configuration de Busybox utilise aussi KConfig et est disponible grâce à la commande **make menuconfig**.

Busybox (2)

Configuration

La configuration de Busybox utilise aussi KConfig et est disponible grâce à la commande **make menuconfig**.



Busybox (3)

Installation

Pour la compilation et l'installation :

```
> make ARCH=arm CROSS_COMPILE=/path/to/binary
HOSTCC  scripts/basic/fixdep
HOSTCC  scripts/basic/split-include
HOSTCC  scripts/basic/docproc
...
DOC     BusyBox.txt
DOC     busybox.1
DOC     BusyBox.html
> mkdir fake_rfs
> make install CONFIG_PREFIX=./fake_rfs/
./fake_prefix///bin/ash -> busybox
./fake_prefix///bin/base64 -> busybox
...
```

Busybox (4)

Liens symboliques

```
> ls fake_rfs
total 820
  drwxr-xr-x 2 4096 Dec 21 18:01 .
  drwxr-xr-x 5 4096 Dec 21 18:01 ..
  lrwxrwxrwx 1 7 Dec 21 18:01 ash -> busybox
  l-rwxr-xr-x 1 829680 Dec 21 18:01 busybox
  lrwxrwxrwx 1 7 Dec 21 18:01 cat -> busybox
  lrwxrwxrwx 1 7 Dec 21 18:01 base64 -> busybox
  ...
```

Busybox (4)

Liens symboliques

```
> ls fake_rfs
total 820
  drwxr-xr-x 2 4096 Dec 21 18:01 .
  drwxr-xr-x 5 4096 Dec 21 18:01 ..
  lrwxrwxrwx 1 7 Dec 21 18:01 ash -> busybox
  l-rwxr-xr-x 1 829680 Dec 21 18:01 busybox
  lrwxrwxrwx 1 7 Dec 21 18:01 cat -> busybox
  lrwxrwxrwx 1 7 Dec 21 18:01 base64 -> busybox
  ...
```

=> busybox est un binaire unique fournissant les commandes grâce à un jeu de liens symboliques

Busybox (4)

Liens symboliques

```
> ls fake_rfs
total 820
  drwxr-xr-x 2 4096 Dec 21 18:01 .
  drwxr-xr-x 5 4096 Dec 21 18:01 ..
  lrwxrwxrwx 1 7 Dec 21 18:01 ash -> busybox
  l-rwxr-xr-x 1 829680 Dec 21 18:01 busybox
  lrwxrwxrwx 1 7 Dec 21 18:01 cat -> busybox
  lrwxrwxrwx 1 7 Dec 21 18:01 base64 -> busybox
  ...
```

=> busybox est un binaire unique fournissant les commandes grâce à un jeu de liens symboliques

=> la taille du binaire est limitée par mutualisation des fonctions communes

U-Boot (1)

Les bootloaders

Un bootloader est notamment chargé d'exécuter le kernel et est capable d'utiliser les fonctions matérielles de base :

- la mémoire vive et mémoire flash
- les ports série
- le réseau filaire (Ethernet)

U-Boot (1)

Les bootloaders

Un bootloader est notamment chargé d'exécuter le kernel et est capable d'utiliser les fonctions matérielles de base :

- la mémoire vive et mémoire flash
- les ports série
- le réseau filaire (Ethernet)

Pour les plateformes x86, GRUB est généralement utilisé.

U-Boot (2)

JTAG

L'installation d'un bootloader sur carte peut être réalisée par liaison série (si un bootloader est déjà présent) ou bien par JTAG : **Join Test Action Group**.

U-Boot (2)

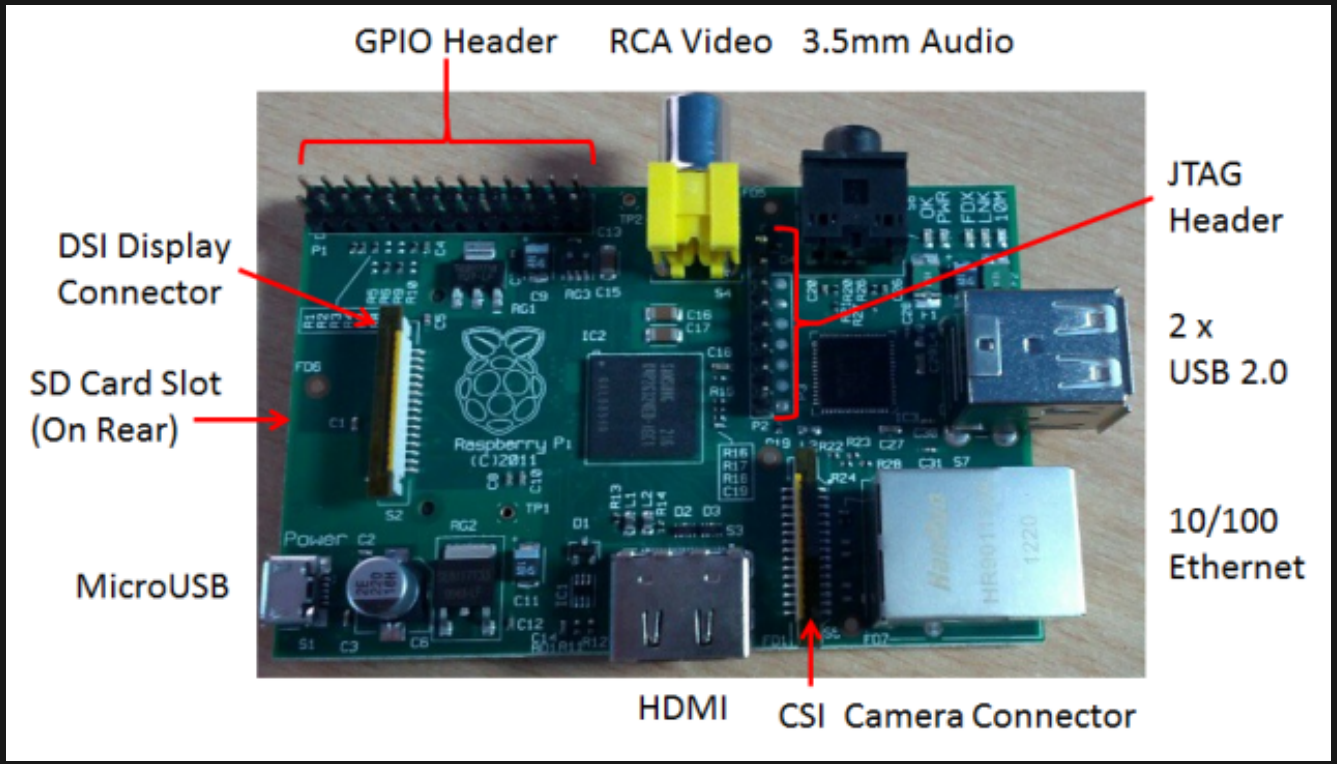
JTAG

L'installation d'un bootloader sur carte peut être réalisée par liaison série (si un bootloader est déjà présent) ou bien par JTAG : **Join Test Action Group**.

=> l'installation du bootloader est une étape critique et peut rendre une carte inutilisable!

U-Boot (3)

JTAG et Raspberry Pi



U-Boot (4)

JTAG et Xbox 360

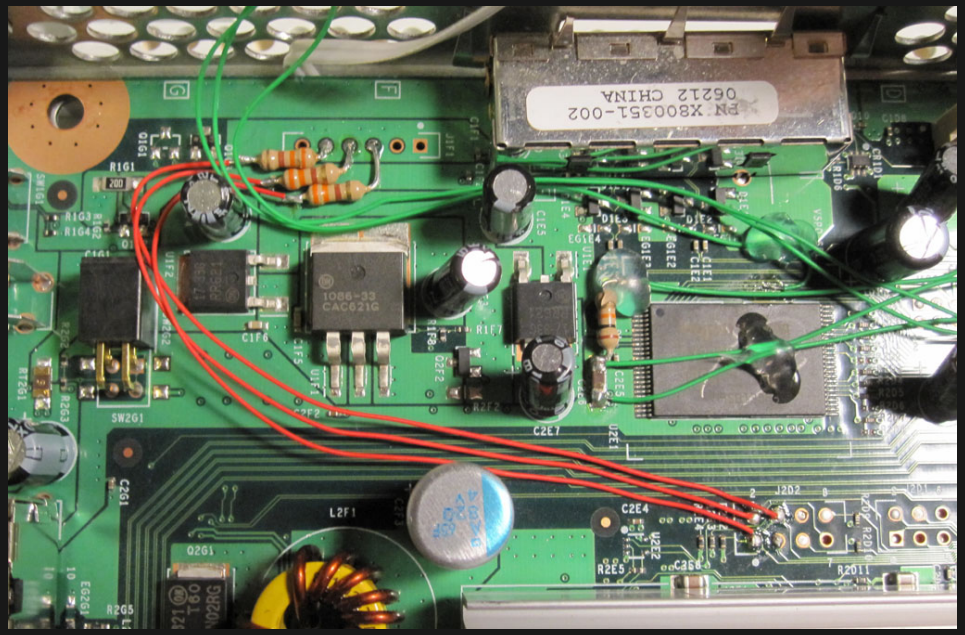
Un hack connu de la Xbox 360 est d'ajouter la possibilité de lire des jeux sur disque dur, clé USB ou par FTP.

U-Boot (4)

JTAG et Xbox 360

Un hack connu de la Xbox 360 est d'ajouter la possibilité de lire des jeux sur disque dur, clé USB ou par FTP.

Pour cela, il faut se connecter sur le port JTAG de la carte mère et charger une nouvelle image NAND!



U-Boot (5)

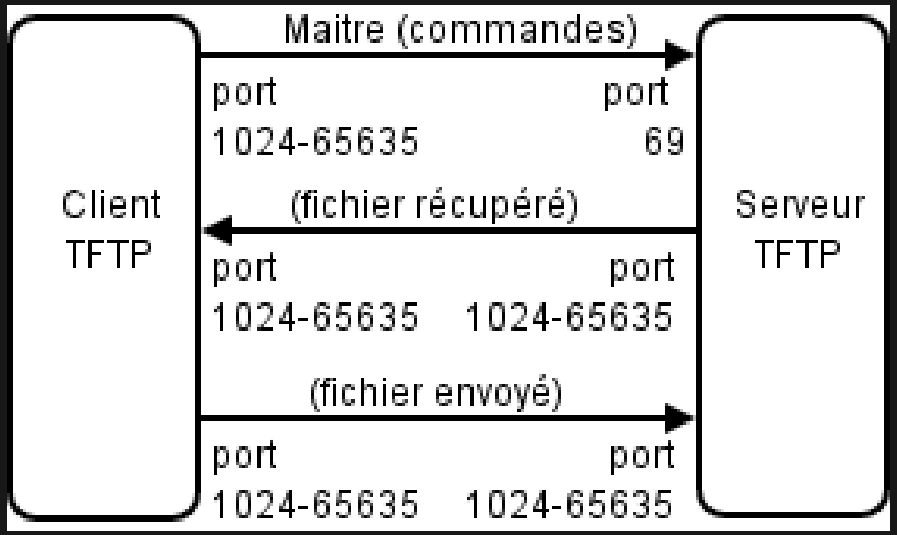
TFTP

Trivial File Transfert Protocol : connexion UDP (port 69 par défaut) permettant le transfert de fichiers.

U-Boot (5)

TFTP

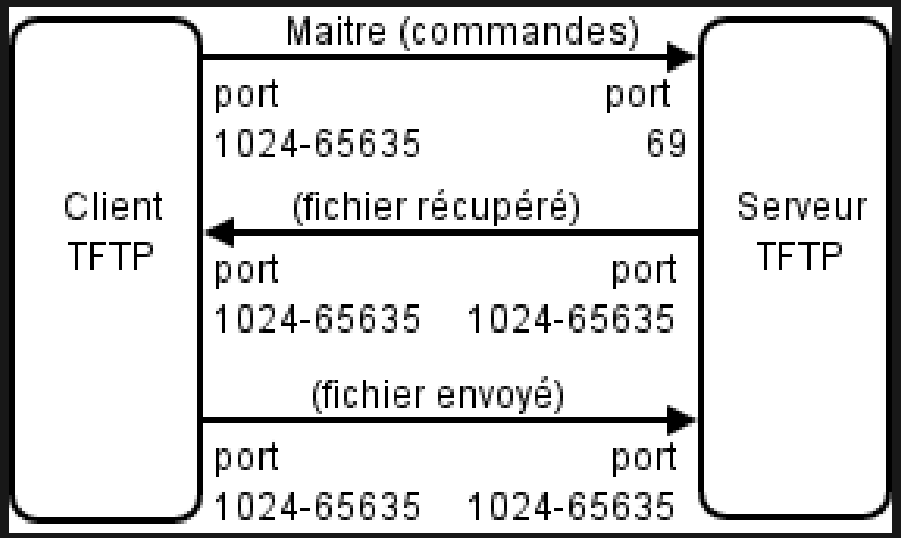
Trivial File Transfert Protocol : connexion UDP (port 69 par défaut) permettant le transfert de fichiers.



U-Boot (5)

TFTP

Trivial File Transfert Protocol : connexion UDP (port 69 par défaut) permettant le transfert de fichiers.



=> très utilisé en embarqué!

U-Boot (6)

Universal Bootloader

Dans le monde de l'embarqué, en dehors des cartes x86, U-Boot est le bootloader le plus répandu.

U-Boot (6)

Universal Bootloader

Dans le monde de l'embarqué, en dehors des cartes x86, U-Boot est le bootloader le plus répandu.

Utilisation classique :

- connexion à la carte par liaison série et accès au prompt U-Boot
- configuration du server TFTP
- téléchargement de l'image kernel et du RFS par TFTP et écriture sur mémoire flash
- modification de la commande de démarrage
- boot

U-Boot (7)

prompt

Le prompt de U-Boot offre de très nombreuses commandes et permet d'enregistrer sur la flash des variables d'environnement.

U-Boot (7)

prompt

Le prompt de U-Boot offre de très nombreuses commandes et permet d'enregistrer sur la flash des variables d'environnement.

```
U-Boot 2011.12 (Dec 19 2012 - 14:53:34)

CPU:   Freescale VyBrid 600 family rev1.0 at 0 MHz
Board: Vybrid
DRAM:  256 MiB
WARNING: Caches not enabled
NAND:  512 MiB
MMC:   FSL_SDHC: 0
In:    serial
Out:   serial
Err:   serial
Net:   Link UP timeout
FEC0, FEC1
Hit any key to stop autoboot:  0
pcm052 u-boot>
```

U-Boot (8)

Commandes et variables d'environnement

Liste non exhaustive :

- **run** : exécution d'une macro
- **bootm** : démarrage d'une image stockée en mémoire
- **bootcmd** : macro de démarrage (exécutée automatiquement quand **bootdelay** arrive à 0 en autoboot)
- **boot** : équivaut à **run bootcmd**
- **setenv** : affectation d'une variable d'environnement
- **saveenv** : sauvegarde de l'environnement sur la flash
- **printenv** : affichage des variables d'environnement et des macros

Automatisation (1)

Buildroot

Buildroot est un ensemble de Makefile automatisant le process de build d'une distribution Linux embarquée.

Automatisation (1)

Buildroot

Buildroot est un ensemble de Makefile automatisant le process de build d'une distribution Linux embarquée.

Pour la phase de cross-compilation, soit Buildroot en construit une, soit on lui indique d'utiliser une chaîne custom (comme celle de Crosstool-ng).

Automatisation (1)

Buildroot

Buildroot est un ensemble de Makefile automatisant le process de build d'une distribution Linux embarquée.

Pour la phase de cross-compilation, soit Buildroot en construit une, soit on lui indique d'utiliser une chaîne custom (comme celle de Crosstool-ng).

Configurable à la *menuconfig*.

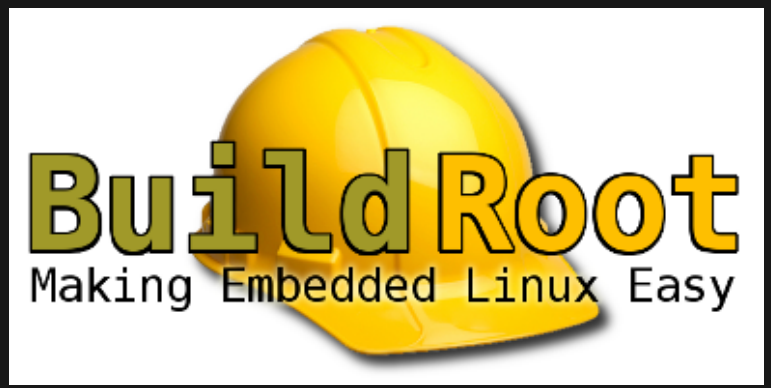
Automatisation (1)

Buildroot

Buildroot est un ensemble de Makefile automatisant le process de build d'une distribution Linux embarquée.

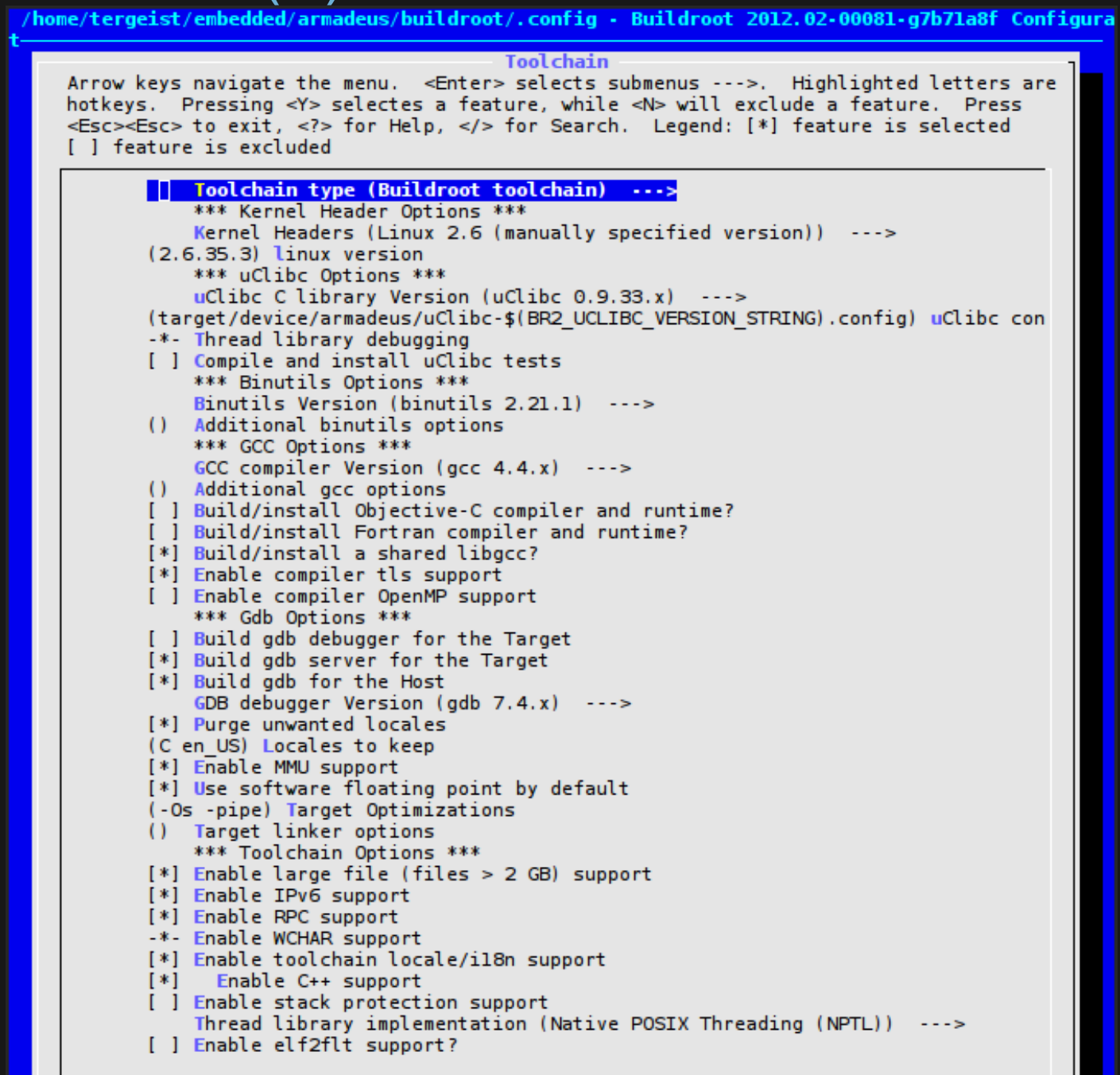
Pour la phase de cross-compilation, soit Buildroot en construit une, soit on lui indique d'utiliser une chaîne custom (comme celle de Crosstool-ng).

Configurable à la *menuconfig*.



Automatisation (2)

Buildroot



Automatisation (3)

Buildroot

Les répertoires principaux :

- **configs** : fichiers de configuration pour la compilation
- **downloads** : les archives téléchargées sont ici
- **output/build** : les paquets et librairies compilés
- **output/host** : chaîne de cross-compilation
- **output/images** : les images compilées
- **package** : les règles de compilation

Automatisation (4)

Buildroot

Les options principales du make :

- **clean** : nettoie tout ce qui a été compilé
- **toolchain** : contruit seulement la chaîne de compilation croisée
- **busybox-menuconfig** : configuration de busybox
- **uclibc-menuconfig** : configuration de la uclibc
- **linux-menuconfig** : configuration du kernel

Automatisation (4)

Buildroot

Buildroot créé des fichiers *.stamp_* pour se repérer dans les étapes de construction/compilation :

```
> cd output/build/busybox-1.23.1
> ls .stamp_*
.stamp_built       .stamp_downloaded  .stamp_patched
.stamp_configured  .stamp_extracted   .stamp_installed
```

Automatisation (4)

Yocto Project

Système de build Open Source plus long à prendre en main mais offrant davantage de fonctionnalités que Buildroot.

Open Embedded: à l'origine de Yocto.

Automatisation (5)

Yocto Project

Architecture moins statique que Buildroot.

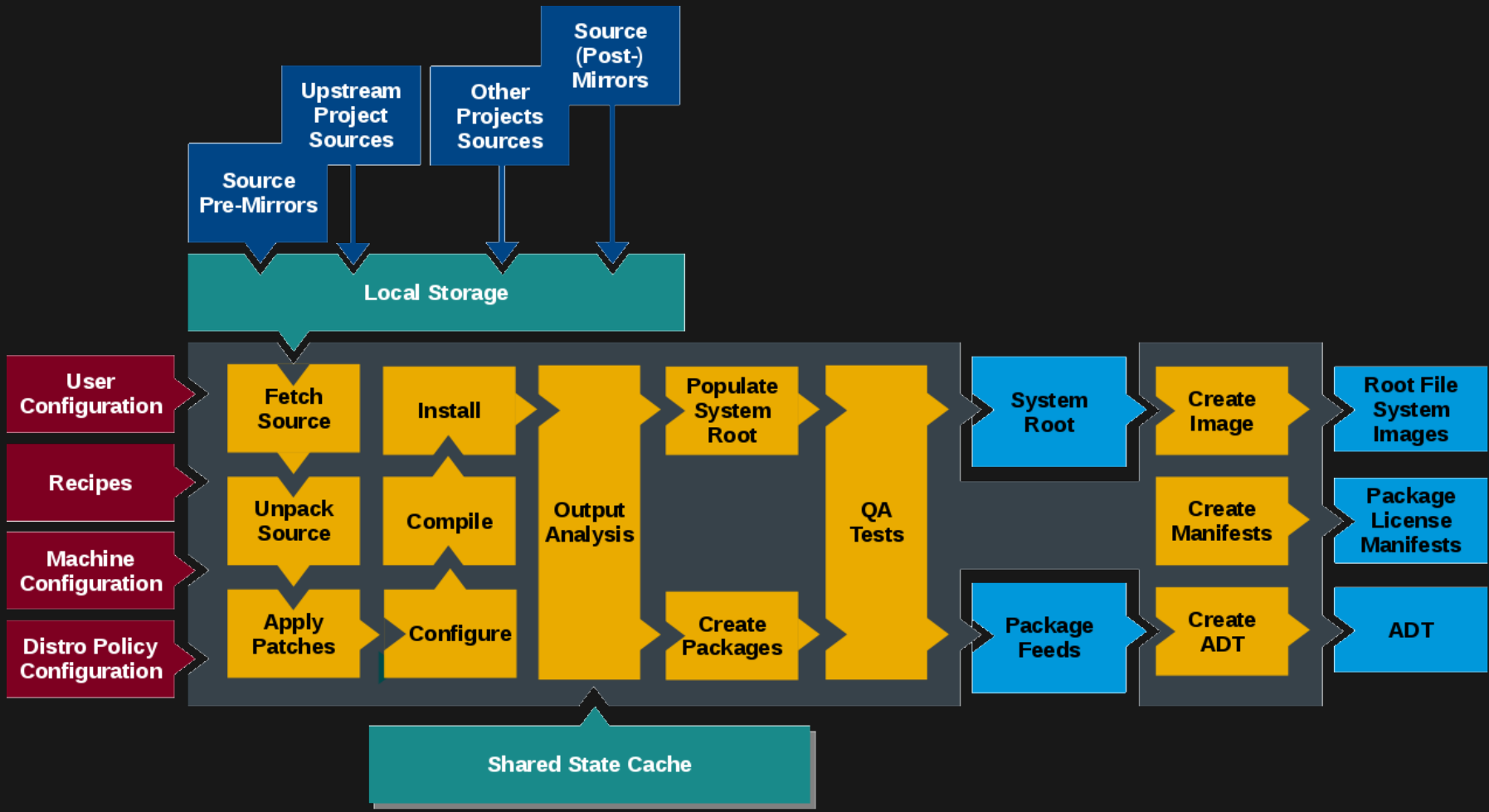
Très utilisé dans l'industrie.

Les éléments de base:

- **Poky**: Platform Builder
- **BitBake**: Moteur de Build
- **OpenEmbedded Core**: Recipes, Layers et Class de base

Automatisation (6)

Yocto Project



Automatisation (7)

Yocto Project

```
> git clone -b thud \
    git://git.yoctoproject.org/poky.git poky-thud
> ls poky-thud
bitbake         LICENSE  meta-poky     meta-skeleton
oe-init-build-env README.LSB   README.qemu
documentation   meta     meta-selftest
meta-yocto-bsp README.hardware    README.poky
scripts
```

Automatisation (8)

Yocto Project

Layer:

- **meta-poky**, **meta-yocto-bsp**, ...
- Contient des recettes **recipes**
- Séparation logique de fonctionnalités
- Peut écrire des Layer custom

Automatisation (9)

Yocto Project

Recipes

- 1 ou plusieurs Recettes par Layer
- Organisées en sous-répertoire: **meta-/**
- Contient des fichier BitBake **.bb**

```
BBLAYERS ?= " \
  /home/user/poky/meta \
  /home/user/poky/meta-yocto \
  /home/user/poky/meta-yocto-bsp \
  /home/user/poky/meta-raspberrypi \
```

Automatisation (10)

Yocto Project

Configurer le build en modifiant le fichier **conf/local.conf**:

- **MACHINE**: la cible
- **IMAGE_INSTALL**: ajouter la recette d'une layer
- **PACKAGE_CLASSES**: RPM, DEB ou IPK
- **DL_DIR**: répertoire de téléchargement
- ...

Modifier les Layers à utiliser:

- Modifier **BBLAYERS** dans **conf/bblayers.conf**
- Utiliser **bitbake-layers add-layer**

Automatisation (11)

Yocto Project

```
$ . ./oe-init-build-env build
$ bitbake core-image-minimal
$ bitbake meta-toolchain
$ ls build/tmp/deploy/images
...
```

QEMU (1)

Principe

Émulateur de matériel permettant de simuler des architectures différentes de celle courante : x86, PPC, ARM SPARC.

QEMU (1)

Principe

Émulateur de matériel permettant de simuler des architectures différentes de celle courante : x86, PPC, ARM SPARC.

Très utilisé en embarqué car permet de tester les images compilées (kernel, rootfs et uboot) directement sur la machine de travail :

- gain de temps important
- facilité d'utilisation
- permet de se passer du matériel pour la phase de mise au point

QEMU (2)

Utilisation

Il existe un binaire **qemu-system-<ARCH>** pour chaque architecture supportée : qemu-system-arm, qemu-system-ppc, qemu-system-x86_64, ...

QEMU (2)

Utilisation

Il existe un binaire **qemu-system-<ARCH>** pour chaque architecture supportée : qemu-system-arm, qemu-system-ppc, qemu-system-x86_64, ...

Les options principales :

- **-M** : machine à émuler
- **-m** : quantité de RAM
- **-kernel** : image kernel à utiliser
- **-initrd** : ramdisk

QEMU (2)

Utilisation

Il existe un binaire **qemu-system-<ARCH>** pour chaque architecture supportée : qemu-system-arm, qemu-system-ppc, qemu-system-x86_64, ...

Les options principales :

- **-M** : machine à émuler
- **-m** : quantité de RAM
- **-kernel** : image kernel à utiliser
- **-initrd** : ramdisk

```
> qemu-system-arm -M versatilepb -m 128M -kernel \
  uImage -initrd rootfs
```

Conclusion

Il existe de très nombreux outils dans le monde des systèmes embarqués. Le tout est simplement de s'y retrouver!

Références

- Free Electrons
- Linux Embarqué - Pierre Ficheux