Introduction à Linux embarqué - Buildroot

Mylène Josserand

Développeuse et formatrice Linux embarqué

Cours INSA, Octobre 2019

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

Dessin issu d'une photographie de Samuel Blanc

Configuration & compilation

Mylène Josserand
josserand.mylene@gmail.com

Organisation du build

- Toutes les sorties de compilation vont dans `output/`
- Le fichier de configuration est un `.config`
- `buildroot/`
  - `.config`
  - `arch/`
  - `package/`
  - `output/`
  - `fs/`
  - ...

Config entière vs. *defconfig*

- `.config` = fichier de config *entier* ➔contient les valeurs de toutes les options
  Le `.config` par défaut a +3000 lignes
  ➔Pas pratique et lisible
- Un `defconfig` stocke seulement les options qui n'ont pas de valeurs par défaut
  ➔Plus facile à lire et modifiable par des humains

defconfig: exemple

- La configuration Buildroot par défaut, le *defconfig* est vide: tout est par défaut.
- Si change pour ARM, le *defconfig* aura juste une ligne :

```
BR2_arm=y
```

- Si ajoute en plus le paquet `stress`, le *defconfig* sera deux lignes :

```
BR2_arm=y
BR2_PACKAGE_STRESS=y
```

Utilisation et création d'un *defconfig*

- Buildroot permet de charger un *defconfig* issu du dossier `configs/` via : `make <foo>_defconfig`
  - Ca va surcharger le `.config`, si il y en a un
- Pour créer un *defconfig* : `make savedefconfig`
  - Fichier pointé par l'option `BR2_DEFCONFIG`
  - Par défaut, pointe sur le `defconfig` original

defconfigs existants

- Il existe un *defconfigs* pour différentes plateformes supportées :
  - RaspberryPi, BeagleBone Black, CubieBoard, Microchip evaluation boards, Minnowboard, quelques i.MX6 boards
- Pour les lister : `make list-defconfigs`
- La plupart des *defconfigs* sont minimaux : compile seulement a toolchain, bootloader, kernel and minimal root filesystem.

```
$ make qemu_arm_vexpress_defconfig
$ make
```

- Des instructions sont souvent disponibles dans `board/<boardname>`, e.g.: `board/raspberrypi/readme.txt`.

Quelques tips

- Nettoyer une cible
  - Nettoyer tout le dossier d'output mais garder la configuration :

```
$ make clean
```

  - Tout nettoyer dont configuration et téléchargement des sources :

```
$ make distclean
```

- Pour avoir une compilation plus verbose
  - Par défaut, beaucoup de commandes sont cachées
  - `V=1` : pour avoir toutes les commandes

```
$ make V=1
```

Dessin issu d'une photographie de Samuel Blanc

Buildroot organisation (source/build)

Mylène Josserand
josserand.mylene@gmail.com

Source tree

Source tree (1/3)

- `Makefile`
  - top-level `Makefile`, gère la configuration et orchestre le build
- `Config.in`
  - top-level `Config.in`, options générales + inclusion de tous les autres `Config.in`

Source tree (2/3)

- `toolchain/` : paquets pour générer ou utiliser une toolchain
- `system/`
  - `skeleton/` le squelette du rootfs
  - `Config.in`, options pour des fonctionnalités système comme le système de démarrage, etc.
- `linux/` : le paquet du kernel Linux
- `package/` : tous les paquets user space (2200+)
- `fs/` : Logique pour générer une image rootfs dans différents formats (`ext2`, `squashfs`, `tar`, `ubifs`, etc)
- `boot/` : paquets de bootloarder (`barebox`, `uboot`, etc)

Source tree (3/3)

- `configs/`
  - configurations par défaut pour de nombreuses plateformes
  - Similaire aux defconfigs du kernel
  - `atmel_xplained_defconfig`, `beaglebone_defconfig`, `raspberrypi_defconfig`, etc.
- `board/`
  - Fichiers sépcifiques aux boards (config kernel, patches kernel, ...)
  - Vont de paire avec un *defconfig* dans `configs/`
- `utils/` : des utilitaires pour des developpeurs Buildroot
- `docs/` : documentation en AsciiDoc pouvant être de différents formats
  `https://buildroot.org/downloads/manual/manual.html`

Build tree

Build tree

- `output/`
- Répertoire global
  - `build/`
    - `buildroot-config/`
    - `busybox-1.22.1/`
    - `host-pkgconf-0.8.9/`
    - `kmod-1.18/`
    - `build-time.log`
  - Lieu d'extraction de toutes les archives
  - Lieu de compilation de chaque paquet
  - Ajout en plus de fichiers de *stamp* créés par BR
  - Variable: `BUILD_DIR`

Build tree: output/host

- output/
  - host/
    - lib
    - bin
    - sbin
    - <tuple>/sysroot/bin
    - <tuple>/sysroot/lib
    - <tuple>/sysroot/usr/lib
    - <tuple>/sysroot/usr/bin
  - Contient des outils pour l'hôte (cross-compiler, etc.) et le *sysroot* pour la toolchain
  - Variable: HOST_DIR
  - Variable for the *sysroot*: STAGING_DIR
  - <tuple> est un identifiant pour l'architecture, le vendeur, l'os, etc : arm-unknown-linux-gnueabihf.

Build tree: output/target

- output/
  - target/
    - bin/
    - etc/
    - lib/
    - usr/bin/
    - usr/lib/
    - usr/share/
    - usr/sbin/
    - THIS_IS_NOT_YOUR_ROOT_FILESYSTEM
    - ...
  - Le routfs **cible**
  - Pas complètement prêt pour la cible: permissions, fichiers device, etc
  - Utilisé pour générer le rootfs final images/
  - Variable: TARGET_DIR

Build tree: output/images

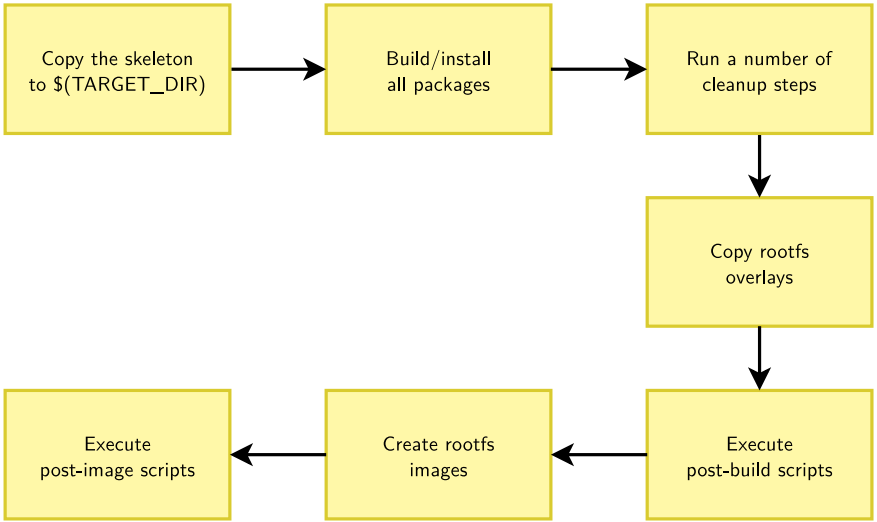
- output/
  - images/
    - zImage
    - armada-370-mirabox.dtb
    - rootfs.tar
    - rootfs.ubi
  - Contient les images finales: kernel, bootloader, rootfs image(s)
  - Variable: BINARIES_DIR

Dessin issu d'une photographie de Samuel Blanc

Root filesystem dans Buildroot

Mylène Josserand
josserand.mylene@gmail.com

Construction d'un rootfs



Root filesystem overlay

- **root filesystem overlay** : Permet de customiser le contenu du rootfs : ajout de config, scripts, répertoires, etc
- Un *root filesystem overlay* est un répertoire où le contenu sera **copié au dessus du root filesystem**. Permet d'écraser des fichiers.
- Option `BR2_ROOTFS_OVERLAY` avec liste des chemins du rootfs overlay

```
$ grep ^BR2_ROOTFS_OVERLAY .config
BR2_ROOTFS_OVERLAY="board/myproject/rootfs-overlay"
$ ls -l board/myproject/rootfs-overlay
board/myproject/rootfs-overlay/etc/ssh/sshd_config
board/myproject/rootfs-overlay/etc/init.d/S99myapp
```

Scripts Post-build

- Parfois, le *root filesystem overlay* ne suffit pas →**scripts post-build**.
- Pour **customiser des fichiers existants**, **supprimer des fichiers** pour sauver de l'espace, etc
- Executé **avant** que l'image du rootfs soit créée
- `BR2_ROOTFS_POST_BUILD_SCRIPT` : liste du chemin de scripts
- Contient différents arguments

Script post-build: example

board/myproject/post-build.sh

```
#!/bin/sh
TARGET_DIR=$1
BOARD_DIR=board/myproject/

# Generate a file identifying the build (git commit and build date)
echo $(git describe) $(date +%Y-%m-%d-%H:%M:%S) > \
    $TARGET_DIR/etc/build-id

# Create /applog mountpoint, and adjust /etc/fstab
mkdir -p $TARGET_DIR/applog
grep -q "^/dev/mtdblock7" $TARGET_DIR/etc/fstab || \
    echo "/dev/mtdblock7\t\t/applog\tjffs2\tdefaults\t\t0\t0" >> \
    $TARGET_DIR/etc/fstab

# Remove unneeded files
rm -rf $TARGET_DIR/usr/share/icons/bar
```

Buildroot configuration

```
BR2_ROOTFS_POST_BUILD_SCRIPT="board/myproject/post-build.sh"
```

Scripts post-image

- `Script post-image` : Executé **à la fin** de la génération de l'image du rootfs
- Pour faire n'importe quelle action nécessaire en fin de build. Exemple:
  - Generer une image finale
  - Démarrer un processus de flashage
- `BR2_ROOTFS_POST_IMAGE_SCRIPT` : liste de *post-image* scripts

Déploiement des images

- Par défaut, Buildroot stockent les images dans `output/images`
- Déploiement doit être fait par l'utilisateur
- Si stockage amovible comme carte SD ou clef USB :
  - Création manuellement des partitions et extraction des composants
  - Utilisation d'un outil `genimage` pour tout gérer

Déploiement des images : genimage

- `genimage` : peut créer des image complète pour des périphériques blocks (carte SD, clef USB, disque dur)
- Exemple commun avec 2 partitions : une partition FAT partition pour bootloader & kernel, une partition ext4 pour le root filesystem
- Il existe un script d'aide `support/scripts/genimage.sh`
- De plus en plus utilisé →Image toute prête

Deploying the images: genimage example

genimage-raspberrypi.cfg

```
image boot.vfat {
  vfat {
    files = {
      "bcm2708-rpi-b.dtb",
      "rpi-firmware/bootcode.bin",
      "rpi-firmware/cmdline.txt",
      "zImage"
      [...]
    }
  }
}
```

```
image sdcard.img {
  hdimage {
  }
  partition boot {
    partition-type = 0xC
    bootable = "true"
    image = "boot.vfat"
  }
  partition rootfs {
    partition-type = 0x83
    image = "rootfs.ext4"
  }
}
```

defconfig

```
BR2_ROOTFS_POST_IMAGE_SCRIPT="support/scripts/genimage.sh"
BR2_ROOTFS_POST_SCRIPT_ARGS="-c board/raspberrypi/genimage-raspberrypi.cfg"
```

flash

```
dd if=output/images/sdcard.img of=/dev/sdb
```

Merci de votre attention !
Questions ? Commentaires ?

Mylène Josserand — josserand.mylene@gmail.com