Buildroot

Distribution Linux Embarquée

Auteur : Julien Gaulmin <jga@cogitux.eu>

1. But

Nous cherchons ici à créer un système Linux embarqué minimaliste mais en utilisant la distribution Buildroot pour construire notre chaîne de compilation croisée et notre *rootfs*.

Cette fois-ci nous travaillerons sur une vrai carte avec SOC à savoir une célèbre Raspberry-Pi.

Nous mettrons en oeuvre des outils graphiques grâce à la bibliothèque QT5.

2. Prise en main de Buildroot

Buildroot est basé sur un ensemble de Makefiles qui automatisent la création de la chaîne de compilation croisée, du *rootfs* (logiciels y compris). Comme le noyau Linux, il intègre plusieurs interfaces de configuration des composants (texte, semi-graphique et graphique).

2.1. Extraction de l'archive des sources de Buildroot

YYYY.MM en fonction de la version fournie.

```
tar -jxf buildroot-YYYY.MM.tar.bz2
```

2.2. Appliquer la configuration par défaut pour la plateforme Raspberry-Pi 3 avec la bibliothèque QT5 incluse

```
cd buildroot-YYYY.MM

make list-defconfigs
     Built-in configs:
     …
       raspberrypi3_64_defconfig            - Build for raspberrypi3_64
       raspberrypi3_defconfig               - Build for raspberrypi3
       raspberrypi3_qt5we_defconfig         - Build for raspberrypi3_qt5we
       raspberrypi4_defconfig               - Build for raspberrypi4
       raspberrypi_defconfig                - Build for raspberrypi
     …

make raspberrypi3_qt5we_defconfig
```

*Note : Si vous avez une version 2011.12 de la Raspberry-Pi, choisissez la cible "raspberrypi_defconfig" et ajoutez manuellement (via "make menuconfig") les outils hardware "**rpi-userland**", la lib "**Qt5**" et "**eudev**" pour la gestion du peuplement de "/dev".*

2.3. Modifier la configuration pour nos besoins

Dans la configuration de Buildroot 2019.11, nous allons tout d'abord supprimer le paquet "**qt5webengine**" car il ne compile pas correctement. Par contre nous allons ajouter "**qt5cinex**" qui constitue une démo des capacités de la bibliothèque QT5.

```
make menuconfig
```

```
Target packages  --->
      Graphic libraries and applications (graphic/text)  --->
            [*] qt5cinex
                  [*]   High-definition version (aka RPi Edition)
            [*] Qt5  --->
                  [ ]   qt5webengine
```

Note : Pensez à sauvegarder en quittant l'outil de configuration.

2.4. Lancer la compilation complète de Buildroot

Buildroot va dans un premier temps télécharger et compiler les éléments nécessaires à la construction de sa chaîne de développement croisée puis compilera les logiciels sélectionné avant de construire le *rootfs*.

```
make
```

Note : Cette compilation prend environ trois heures dans une VM VirtualBox avec 4 CPU Core i5 Gen8 avec 4Go de RAM.

2.5. Dossiers de sortie Buildroot

Toutes les sorties générées par Buildroot sont dans son dossier “output” dont voici le contenu après la compilation :

```
tree -L 2 output/
output/
├── build
│   ├── … on trouve ici l’ensemble des briques
│   └──    logicielles qui composent notre rootfs …
├── host
│   ├── … on trouve ici l’ensemble des logiciels
│   ├──    nécessaires pour la compilation croisée
│   ├──    et la fabrication des images …
│   ├── bin
│   │   ├── …
│   │   ├── arm-linux-g++ -> toolchain-wrapper
│   │   ├── arm-linux-g++.br_real ->
arm-buildroot-linux-gnueabihf-g++.br_real
│   │   └── …
│   └── …
├── images … ici images binaires : firmware, noyau, device tree,
│   ├──       partitions …
│   ├── bcm2710-rpi-3-b.dtb
│   ├── bcm2710-rpi-3-b-plus.dtb
│   ├── bcm2710-rpi-cm3.dtb
│   ├── boot.vfat
│   ├── rootfs.ext2
│   ├── rootfs.ext4 -> rootfs.ext2
│   ├── rpi-firmware
│   ├── sdcard.img
│   └── zImage
├── staging -> …/arm-buildroot-linux-gnueabihf/sysroot
└── target
    ├── … contenu partiel du rootfs (presque complet aux points
    ├──    d’entrées des pilotes prêt) …
    ├── THIS_IS_NOT_YOUR_ROOT_FILESYSTEM
    └── …
```

2.6. Installer l'image générée sur une sdcard

Vous pouvez introduire la sdcard dans votre PC hôte et faire en sorte qu'elle soit accessible dans votre VM si vous ne travaillez pas en natif. Vérifiez ensuite à quels points d'entrées est associée la sdcard.

```
lsblk
     NAME         MAJ:MIN RM   SIZE RO TYPE  MOUNTPOINT
     …
     sdd          8:48   1   3,8G  0 disk
      ├──sdd1       8:49   1        32M  0 part  /media/julien23/363D-F234
      └──sdd2       8:50   1   360M  0 part  /media/julien23/cbb88f0d-…
```

On voit que le système hôte a monté des partitions existantes de la sdcard, il faut donc les démonter avant de copier l'image sdcard créée par Buildroot.

```
sudo umount /dev/sdd*

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

2.7. Tests

Vous pouvez ensuite introduire votre sdcard dans la carte Raspberry-Pi et démarrer en connectant un écran et un clavier

2.8. Questions et manipulations

Q1) Constatez que le clavier est en qwerty (disposition par défaut). Comment faire pour le passer en français en utilisant Busybox ?

Q2) Imaginez où intégrer cette modification durablement dans votre *rootfs*. Nous le mettrons en oeuvre plus tard en utilisant le mécanisme d'*overlay* de Buildroot.

Q3) Lancez l'application "CinematicExperience-demo" qui nous donne un aperçu des capacités graphiques de QT5 avec le service noyau EGL.

3. Amélioration de notre *rootfs*

3.1. La configuration Buildroot

Buildroot est construit autour d'un ensemble de Makefiles et d'un outil de configuration identique à celui du noyau Linux et de nombreux logiciels open source complexes.

```
make help
     … Nous donne les infos sur les recettes principales fournies par les
     Makefiles de Busybox …
```

On accède à la configuration semi-graphique via la commande maintenant connue :

```
make menuconfig
     Target options  --->
     Build options  --->
     Toolchain  --->
     System configuration  --->
     Kernel  --->
     Target packages  --->
     Filesystem images  --->
     Bootloaders  --->
     Host utilities  --->
     Legacy config options  --->
```

Note : On peut faire des recherches dans les menu de configuration en utilisant la touche /.

Les configuration sont regroupées par thématiques (cible, chaîne de compilation croisée, noyau, système cible, paquets de la cible…).

Pour accéder aux configurations du noyau, de la µClibC ou de Busybox, on peut appeler leur propres menus de configuration :

```
make linux-menuconfig
make busybox-menuconfig
make uclibc-menuconfig
```

3.2. Serveur SSH

Nous allons ajouter le paquet "**dropbear**" (Target packages > Networking applications > Dropbear) qui implémente un serveur SSH léger. Il nous permettra d'accéder à notre cible par le réseau de manière sécurisée.

3.3. Customisations du systèmes

Q4) Ajoutez un **mot de passe root** dans le menu "System configuration" (sans quoi nous ne pourrons pas nous connecter par SSH) et profitez-en pour mettre le ***rootfs* en lecture-seule** (dans le même menu).

3.4. Multimédia

Q5) Ajoutez le paquet “**omxplayer**” qui nous permettra de lire des vidéos en profitant des accélérations matérielles du module graphique du SOC Raspberry-Pi.

3.5. Débogage

Q6) Ajoutez le paquet “**gdbserver**” qui nous permettra de faire du débogage à distance.

3.6. Serveur web

Q7) Ajoutez le serveur web “**lighttpd**” (avec option “pcre support”) et l’interpréteur “**php**” qui nous permettra de faire des pages dynamiques. Ajoutez aussi le serveur web de Busybox pour comparer.

3.7. Finalisation

Q8) Lancer la compilation de votre système et installez le tout sur votre sdcard.

Q9) Testez l’accès SSH et vérifiez que le *rootfs* est bien en lecture-seule.

4. Buildroot *rootfs overlay*

4.1. Dossier *overlay*

Buildroot propose un mécanisme vous permettant de personnaliser votre *rootfs* en ajoutant vos propres productions à la fin de la création du *rootfs*.

Créez un dossier “buildroot_overlay” et indiquez son chemin absolu dans le menu “System configuration > Root filesystem overlay directories”.

Si vous souhaitez ajouter un fichier “fr.kmap” dans le dossier “/etc” de votre *rootfs*, créez un sous-dossier “etc” de votre dossier “buildroot_overlay” et ajoutez votre fichier dedans. Lancez la commande “make” pour mettre à jour votre *rootfs* et ses images.

4.2. Questions et manipulations

Q10) Ajoutez à votre *overlay* les modifications imaginées précédemment pour avoir un clavier français.

Q11) Ajoutez une vidéo dans le dossier “root” de votre *overlay*, elle nous servira à tester “omxplayer” (ex : Blender cycles demos).

Q12) Testez tout ça (clavier, lecture vidéo, etc) sur votre cible.

5. Compilation croisée

5.1. Chaîne de compilation croisée

Buildroot a construit sa propre chaîne de compilation croisée pour générer les exécutables de notre cible. Celle-ci se trouve par défaut dans “output/host/”.

```
./output/host/bin/arm-buildroot-linux-gnueabihf-gcc -v
Using built-in specs.
COLLECT_GCC=/media/dev/Embedded/buildroot-2019.11-raspberrypi3_qt5we/output/host/bin/arm-buildroot-linux-gnueabihf-gcc.br_real
COLLECT_LTO_WRAPPER=/media/dev/Embedded/buildroot-2019.11-raspberrypi3_qt5we/output/host/libexec/gcc/arm-buildroot-linux-gnueabihf/8.3.0/lto-wrapper
Target: arm-buildroot-linux-gnueabihf
Configured with: ./configure
--prefix=/media/dev/Embedded/buildroot-2019.11-raspberrypi3_qt5we/output/host
--sysconfdir=/media/dev/Embedded/buildroot-2019.11-raspberrypi3_qt5we/output/host/etc --enable-static --target=arm-buildroot-linux-gnueabihf
--with-sysroot=/media/dev/Embedded/buildroot-2019.11-raspberrypi3_qt5we/output/host/arm-buildroot-linux-gnueabihf/sysroot --enable-__cxa_atexit --with-gnu-ld
--disable-libssp --disable-multilib --disable-decimal-float
--with-gmp=/media/dev/Embedded/buildroot-2019.11-raspberrypi3_qt5we/output/host
--with-mpc=/media/dev/Embedded/buildroot-2019.11-raspberrypi3_qt5we/output/host
--with-mpfr=/media/dev/Embedded/buildroot-2019.11-raspberrypi3_qt5we/output/host --with-pkgversion='Buildroot 2019.11'
--with-bugurl=http://bugs.buildroot.net/ --disable-libquadmath --enable-tls
--enable-threads --without-isl --without-cloog --with-abi=aapcs-linux
--with-cpu=cortex-a53 --with-fpu=neon-vfpv4 --with-float=hard --with-mode=arm
--enable-languages=c,c++
--with-build-time-tools=/media/dev/Embedded/buildroot-2019.11-raspberrypi3_qt5we/output/host/arm-buildroot-linux-gnueabihf/bin --enable-shared
--disable-libgomp
Thread model: posix
gcc version 8.3.0 (Buildroot 2019.11)
```

5.2. Questions et manipulations

Q13) “Cross-compilez” un programme “helloworld” et placez le dans le dossier “/root” de votre image cible en utilisant votre dossier *overlay*.

Q14) Testez votre exécutable préalablement au reflashage en le transférant dans le “/tmp” de la cible via SCP (transfert de fichiers par SSH).

6. WiFi Raspberry-Pi

6.1. *Firmwares* propriétaire Broadcom

Les SOC BCM283x de Broadcom utilisés sur les Raspberry-Pi nécessitent des composants logiciels propriétaires pour faire fonctionner certains sous-ensembles (WiFi, Bluetooth, vidéo, etc).

Dans Buildroot on trouve les paquets pour ces *firmwares* dans “Target packages > Hardware handling > Firmware” et pour les plateformes Raspberry-Pi ils se nomment “rpi-*”.

6.2. Configuration du WiFi

Outre le *firmware* pour gérer le matériel, nous avons besoin du paquet “wpa_supplicant” pour gérer la connexion et l’authentification WiFi.

Vous pouvez utiliser la commande “wpa_passphrase” pour créer votre fichier de configuration “/etc/wpa_supplicant.conf” :

```
wpa_passphrase SSID_De_Mon_WiFi SuperMotDePasse1234
      network={
          ssid="SSID_De_Mon_WiFi"
          #psk="SuperMotDePasse1234"
          psk=88324845b0fc642b7a88f269c675c7afa0ef24b26cd7ba072288e8bdcd6fc1dc
      }
```

Ainsi, vous pouvez créer votre fichier “wpa_supplicant.conf” dans votre *overlay* :

```
cat rootfs_overlay/etc/wpa_supplicant.conf
      ctrl_interface=/var/run/wpa_supplicant
      ap_scan=1
      network={
          ssid="SSID_De_Mon_WiFi"

          #psk="SuperMotDePasse1234"
          psk=88324845b0fc642b7a88f269c675c7afa0ef24b26cd7ba072288e8bdcd6fc1dc
      }
```

Il faut aussi surcharger le fichier “/etc/network/interfaces” généré par Buildroot pour y ajouter la configuration de l’interface WiFi (wlan0) :

```
cat rootfs_overlay/etc/network/interfaces
      auto lo
      iface lo inet loopback

      auto eth0
      iface eth0 inet dhcp
        pre-up /etc/network/nfs_check
        wait-delay 15
        hostname $(hostname)

      auto wlan0
```

```
iface wlan0 inet dhcp
  pre-up wpa_supplicant -B -Dnl80211 -iwlan0 -c/etc/wpa_supplicant.conf
  post-down killall -q wpa_supplicant
  wait-delay 15
```

6.3. Questions et manipulations

Q15) Ajoutez les paquets “rpi-wifi-firmware” et “wpa_supplicant” (avec l’option “wpa_passphrase”) à votre configuration.

Q16) Mettez à jour “interfaces” et ajouter votre “wpa_supplicant.conf” dans votre *overlay*.

Q17) Générez votre image et testez la sur la cible.

7. Débogage à distance

7.1. Débogueur croisé

Par défaut, le débogueur n'est pas compilé lors de la création de la chaîne de compilation croisée Buildroot. Vous pouvez l'activer via le menu "Toolchain > Build cross gdb for the host" de Buildroot (l'option "TUI support" est recommandée). Vous pouvez ensuite relancer la compilation des composants manquant avec la commande "make".

7.2. Débogage de votre programme "helloworld"

Compilez votre programme "helloworld.c" avec le compilateur croisé de Buildroot sans oublier d'intégrer les symboles de débogage (`-g`) :

```
host$ ./output/host/bin/arm-buildroot-linux-gnueabihf-gcc -g -o
../buildroot_overlay/root/helloworld.gdb ../src/helloworld.c
```

Transférez le binaire sur la cible et exécutez le avec "gdbserver" :

```
host$ scp ../buildroot_overlay/root/helloworld.gdb root@ip_raspberry:/tmp

target$ chmod 755 /tmp/helloworld.gdb
target$ gdbserver :1234 /tmp/helloworld.gdb
```

Lancer le débogueur croisé et connectez le au "gdbserver" de votre cible. Vous pouvez alors le contrôler à distance.

```
host$ ./output/host/bin/arm-buildroot-linux-gnueabihf-gdb --tui --directory ..
../buildroot_overlay/root/helloworld.gdb
     (gdb) target remote 192.168.23.252:1234
     (gdb) list
     (gdb) break 5
     (gdb) cont
     (gdb) backtrace
     (gdb) info reg
     (gdb) next
     (gdb) cont
```

7.3. Questions et manipulations

Q18) Récupérez le programme "buggy.c", compilez le et envoyez le sur la cible pour le déboguer à distance.

8. Leds et GPIO

8.1. Présentation

Le noyau Linux propose une interface unifiée d'accès aux leds et aux GPIO via le pseudo système de fichiers "Sysfs". Il conserve ainsi la philosophie Unix du "tout fichier".

Cette interface est documentée dans les fichiers "led-class.txt" et "gpio.txt" du noyau Linux dont des extraits utiles sont repris dans le document "Raspberry-Pi_Annexes.pdf".

```
# ls /sys/class/leds/
led0

# ls /sys/class/leds/led0/
brightness  device  max_brightness  power  subsystem  trigger  uevent

# cat /sys/class/leds/led0/brightness
0

# echo 1 >/sys/class/leds/led0/brightness

# echo 0 >/sys/class/leds/led0/brightness

# cat /sys/class/leds/led0/trigger
[none] rc-feedback kbd-scrolllock kbd-numlock kbd-capslock kbd-kanalock
kbd-shiftlock kbd-altgrlock kbd-ctrllock kbd-altlock kbd-shiftllock
kbd-shiftrlock kbd-ctrlllock kbd-ctrlrlock timer oneshot heartbeat backlight
gpio cpu cpu0 default-on input panic mmc0

# echo cpu >/sys/class/leds/led0/trigger

# cat /sys/class/leds/led0/trigger
none rc-feedback kbd-scrolllock kbd-numlock kbd-capslock kbd-kanalock
kbd-shiftlock kbd-altgrlock kbd-ctrllock kbd-altlock kbd-shiftllock
kbd-shiftrlock kbd-ctrlllock kbd-ctrlrlock timer oneshot heartbeat backlight
gpio [cpu] cpu0 default-on input panic mmc0
```

8.2. Questions et manipulations

Q19) En vous appuyant sur l'extrait de "led-class.txt" du noyau Linux, configurez la LED "ACT" de votre cible pour la piloter à la demande.

Q20) Configurez ensuite la LED "ACT" pour qu'elle indique la santé du système (*heartbeat*). Comment faire pour automatiser cette fonctionnalité au démarrage ?

Q21) En vous appuyant sur l'extrait de "gpio.txt" du noyau Linux, configurez la PIN11 du Raspberry-Pi (GPIO17 du SOC) en sortie et servez-vous en pour piloter une LED.

Q22) Configurez à présent la PIN12 du Raspberry-Pi en entrée et lisez le statut de cette entrée tout en faisant varier physiquement son état à l'aide d'un fil (en utilisant la masse et le +3,3V alternativement).

Q23) Écrire un programme en langage C pour attendre et réagir de façon optimale à des fronts montants sur la PIN15 (appels système `poll()` par exemple).

Q24) Quels sont les avantages de l'appel système `poll()` par rapport à une lecture périodique de l'état d'une GPIO ?

9. Serveur web

9.1. Installation

Nous avons déjà installé les paquets “**lighttpd**” et “**php**” tout à l’heure, il faut maintenant les configurer et créer une première page web.

9.2. Configuration via l’*overlay*

Le fichier “/etc/lighttpd/modules.conf” contient la liste des modules utilisés par “lighttpd”. Nous allons copier le fichier par défaut dans notre arborescence d’*overlay* et activer “fastcgi” pour pouvoir utiliser “php” :

```
cat rootfs_overlay/etc/lighttpd/modules.conf
     …
     server.modules = (
     …
       "mod_fastcgi",
     )
     …
     ##
     ## FastCGI (mod_fastcgi)
     ##
     include "conf.d/fastcgi.conf"
     …
```

Le fichier “/etc/lighttpd/conf.d/fastcgi.conf” contient la configuration spécifique du module “fastcgi”, il faut le créer dans notre arborescence d’*overlay* :

```
cat rootfs_overlay/etc/lighttpd/conf.d/fastcgi.conf
     fastcgi.server = ( ".php" => ((
         "bin-path" => "/usr/bin/php-cgi",
         "socket" => "/var/tmp/php.socket"
     )))
```

La racine du serveur (“server_root”) définie dans le fichier de configuration “lighttpd.conf” est “/var/www”, il faut donc y placer un fichier “index.php” de démo pour voir si tout fonctionne. La fonction “phpinfo()” de PHP est idéale pour ça :

```
cat rootfs_overlay/var/www/index.php
     <?php phpinfo(); ?>
```

9.3. Questions et manipulations

Q25) Générez votre image avec ces modifications et la mettre en place sur votre cible.

Q26) Sur l'hôte, accédez à votre cible à l’aide d’un navigateur en entrant l’IP de celle-ci dans la barre d’adresse. Vous devriez voir la page d’info PHP apparaître.

Q27) Mettez en oeuvre une petite page web pour pilote les GPIO.

10. Le problème de “qt5webengine”

10.1. Problème

Si on ajoute le paquet “qt5webengine”, la compilation échoue pour un problème d’en-tête `<stdlib.h>`.

10.2. Solution

Il s’avère que le problème a été résolu par un *patch* sur l’archive du paquet. Buildroot possède un mécanisme pour appliquer des *patches* après la phase d’extraction du paquet. Le paquet est extrait dans `output/build/qt5webengine-5.12.5/` et les *patches* à appliquer pour ce paquet sont dans `package/qt5/qt5webengine/5.12.5/` .

```
wget -O package/qt5/qt5webengine/5.12.5/0001-pkg_config-Fixes-when-use_sysroot-false.patch https://raw.githubusercontent.com/buildroot/buildroot/6cfe21ae90ccd88254bf5240b2b29d74794f45fe/package/qt5/qt5webengine/5.12.5/0001-pkg_config-Fixes-when-use_sysroot-false.patch
... Pour télécharger le patch dans le dossier du paquet ...

make menuconfig
... Ajoutez qt5webengine ...

make qt5webengine
... Pour lancer la compilation et l’installation du paquet dans le rootfs ...
```