

Chapter 1

Utilisation basique de Buildroot



Objectifs:

- Télécharger Buildroot
- Configurer un système minimal avec Buildroot pour la Raspberry Pi 3
- Lancer la compilation
- Flasher et tester le système généré

1.1 Setup

Comme spécifié dans la documentation Buildroot¹, vous avez besoin de certains paquets/dépendances :

```
sudo apt install sed make binutils gcc g++ bash patch \  
gzip bzip2 perl tar cpio python unzip rsync wget libncurses-dev
```

Lisez les spécificités de votre carte RPi3 sur différents sites notamment [le site de la Raspberry Fondation](#)

1.2 Télécharger Buildroot

Comme nous allons réaliser du développement Buildroot, on va télécharger les sources depuis le répertoire Git :

```
git clone git://git.busybox.net/buildroot
```

Aller dans le nouveau dossier **buildroot** créé.

Nous allons utiliser la branche *2019.08* de cette release.

```
git checkout 2019.08
```

¹<https://buildroot.org/downloads/manual/manual.html#requirement-mandatory>

1.3 Configuration de Buildroot

Si vous regardez le dossier `configs/`, vous verrez qu'il y a un fichier `raspberrypi3_defconfig`, qui est une configuration pour construire un système pour une Raspberry Pi 3. Faites donc un:

```
make raspberrypi3_defconfig
```

Mais comme on souhaite apprendre Buildroot, on va regarder la configuration un peu plus en détails. Démarrez l'utilitaire de configuration Buildroot :

```
make menuconfig
```

Il est aussi possible d'utiliser les autres outils comme `nconfig`, `xconfig` ou `gconfig`. Maintenant, regardons les points importants Buildroot:

- **Menu Target Options**
 - **Target Architecture** : ARM (little endian)
 - **Target Architecture Variant** : D'après le [site web de la RPi3](#), ils utilisent un CPU Broadcom BCM2837 qui est basé sur ARM Cortex-A53
- **Menu Build options** : Regardez notamment le lieu où sauver la configuration Buildroot
- **Menu Toolchain**
 - Par défaut, Buildroot compile sa propre toolchain. Ça prend un peu de temps même si pour ARMv8 il y a une toolchain pré-buildée fournie par ARM.
- **Menu Kernel**
 - Par défaut, le kernel le plus récent dans la release Buildroot est utilisé. Ici, il s'agit d'un kernel spécifique, sur github d'où le **Custom Tarballs**
 - Faites attention à la configuration kernel utilisé Using an in-tree defconfig file avec bcm2709 comme configuration. Correspond au fichier `bcm2709_defconfig` dans le kernel: https://github.com/raspberrypi/linux/blob/rpi-4.19.y/arch/arm/configs/bcm2709_defconfig
 - Sur ARM, beaucoup de plateformes utilisent un *Device Tree* pour décrire le hardware. Vous pouvez regarder son DT : <https://github.com/raspberrypi/linux/blob/rpi-4.19.y/arch/arm/boot/dts/bcm2710-rpi-3-b.dts>,
- **Menu Target packages**. Surement le plus important car c'est le menu qui permet d'installer les paquets souhaités dans le rootfs final. On verra dans un autre lab un exemple.
- **Menu Filesystem images**. Pour savoir quel type d'image on veut générer.
- **Menu Bootloaders**
 - Le plus populaire pour ARM est *U-Boot*. Pour la RPi3 n'a pas besoin d'un bootloader pour executer et charger le kernel donc pas besoin!

1.4 Compilation

Simplement lancez `make` ou redirigez la sortie avec la commande `tee`:

```
make 2>&1 | tee build.log
```

1.5 Flasher la carte SD

Une fois la compilation terminée, on va pouvoir flasher notre système sur notre carte SD. Mais avant, encore faut-il l'identifier ! Regardez la sortie de `cat /proc/partitions` et chercher votre carte SD. En général, un lecteur de carte SD donnera `mmcblk0`, alors qu'un lecteur externe USB, ça sera du type `sdX` (i.e.sdb, sdc, etc.). **Attention : /dev/sda est généralement le disque dur de votre laptop!**

Si votre carte SD est `/dev/mmcblk0`, les partitions seront `/dev/mmcblk0p1`, `/dev/mmcblk0p2`, etc. Si c'est `/dev/sdc`, les partitions seront `/dev/sdc1`, `/dev/sdc2`, etc.

Pour flasher votre système, suivez la commande indiquée dans le README de la RPi3 : <https://git.buildroot.net/buildroot/tree/board/raspberrypi/readme.txt?h=2019.08>:

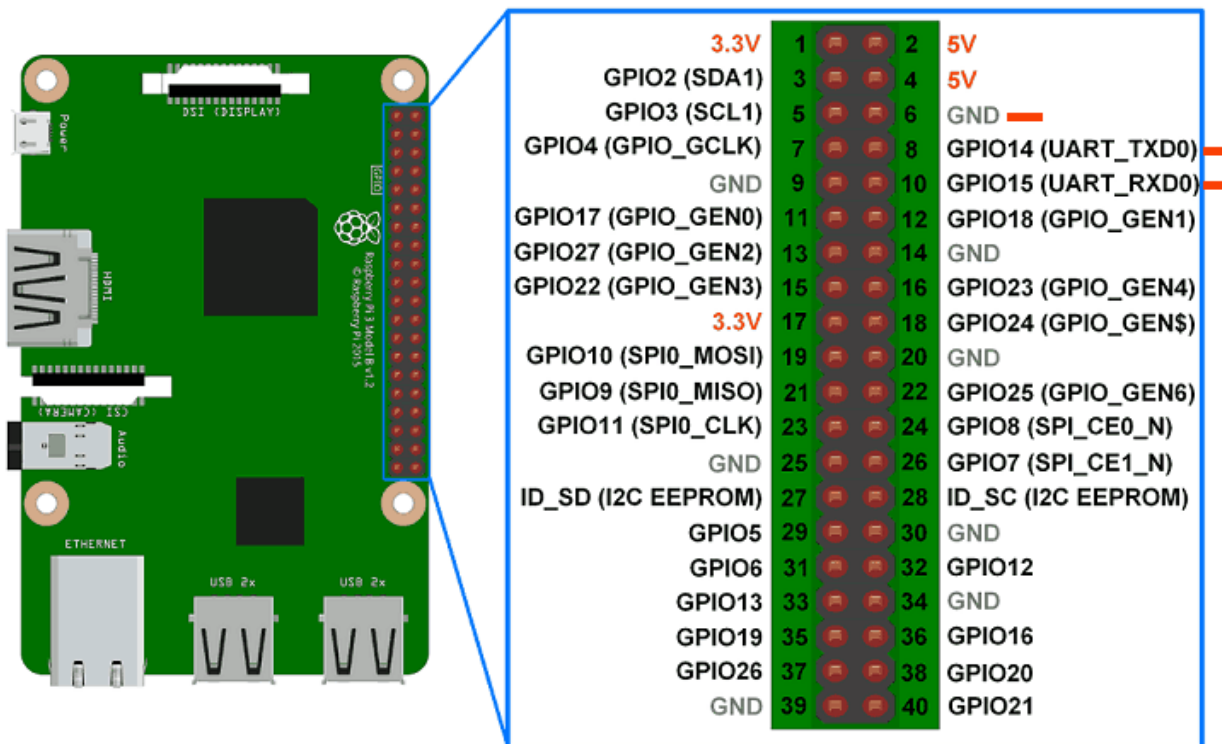
```
sudo dd if=output/images/sdcard.img of=/dev/mmcblk0 bs=1M
```

Ou `sdc` ou `sdb` au lieu de `mmcblk0` si besoin.

1.6 Démarrage du système

Insérez la carte SD dans la RPi3.

Branchez la série selon les pins de votre RPi3 :



Allumez via USB votre board et vous devriez voir le système booter !

Le login est `root` et profitez pour explorer le système. Lancer `ps` pour regarder combien de processus sont en cours d'exécution, qu'est-ce que Buildroot a généré dans `/bin`, `/lib`, `/usr` et `/etc`.

1.7 Explorer le log de build

De retour sur votre machine de build, regardez la sortie du log `build.log`. Buildroot est un peu verbose mais il affichera chaque message important d'un prefixe `>>>`. Pour avoir une idée générale de ce qui a été fait, vous pouvez lancer:

```
grep ">>>" build.log
```

Vous voyez les différents paquets être téléchargés, extraits, patchés, configurés compilés et installés.

Chapter 2

Ajout du support CAN

Objectifs:

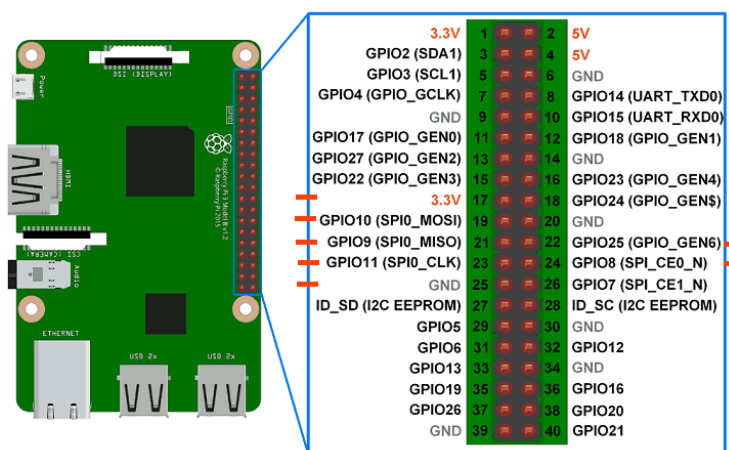
- Ajouter de nouveaux paquets à compiler
- Configurer un périphérique hardware (CAN) sous la RPi3
- Tester le nouveau système

2.1 Avant de commencer...

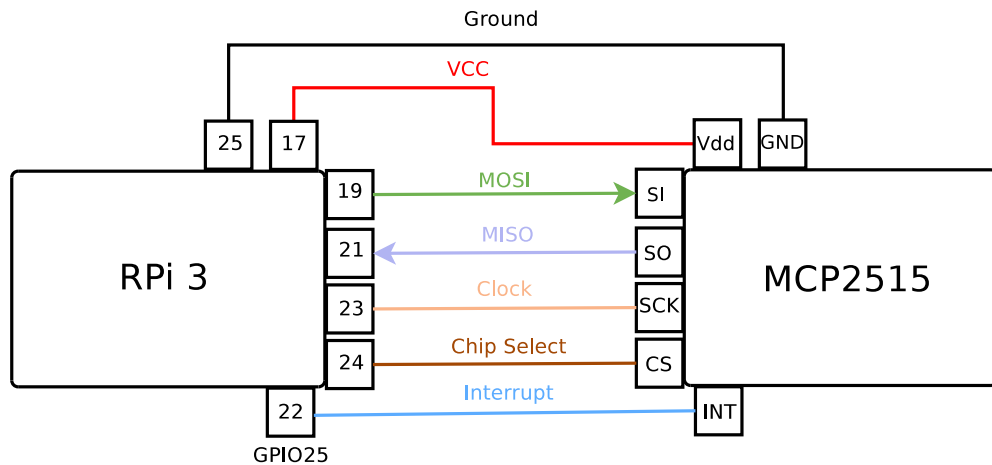
Le contrôleur CAN que nous allons utiliser est un [MCP2515](#) qui utilise un bus SPI.

Prendre un peu de temps pour lire la datasheet du MCP2515, regarder les différentes pins, le voltage en entrée, etc. Cela vous permettra de comprendre les pins suivantes utilisées.

Nous allons utiliser les pins SPI de la RPi :



Ce qui nous donnera une connexion SPI suivante :



2.2 Configuration

2.2.1 Configuration de la RPi3 : defconfig + device-tree

Pour ajouter le support d'un périphérique dans Linux, il faut:

- Ajouter un noeud dans le device tree selon le bus utilisé
- Activer le driver dans la configuration kernel

Device-tree overlay

La RPi a une particularité par rapport à certaines cartes de développement, elle utilise un fichier de configuration de type texte pour faire des modifications hardware via un firmware.

Rechercher ce fichier :

```
$ find . -iname config.txt
./package/rpi-firmware/config.txt
```

Lire les fichiers contenus dans le dossier `board/raspberrypi3/`.

Regarder comment est-il possible de modifier le fichier `config.txt`, notamment avec ce qui est déjà effectué avec l'option `dtoverlay=pi3-miniuart-bt`. Les informations sur la syntaxe de ce fichier sont disponibles sur ce site http://elinux.org/RPi_config. Les `dtoverlay` permettent d'activer des parties hardware de la raspberry pi en modifiant le Device Tree.

Pour activer le bus SPI, il faut utiliser le `dtparam` et pour activer le hardware CS, ça sera via `dtoverlay` (voir la documentation [ici](#)):

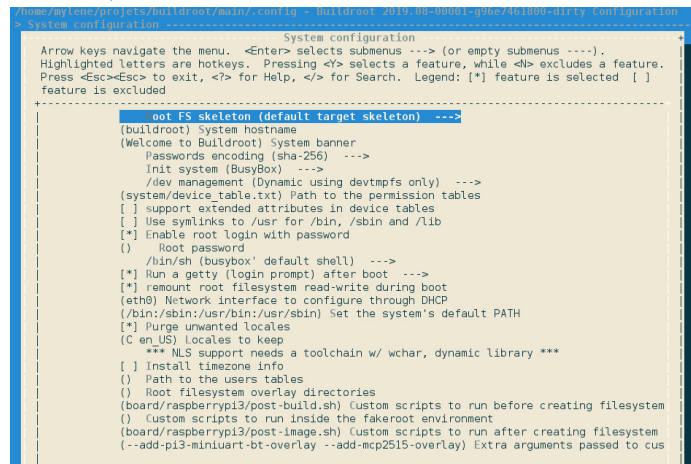
```
dtparam=spi=on
dtoverlay=spi0-hw-cs
```

En ce qui concerne le chip CAN, il faut utiliser un `dtoverlay`. Voici une [documentation](#) à lire concernant leur utilisation avec la node du contrôleur CAN que nous utiliserons `MCP2515`.

```
dtoverlay=mcp2515-can0,oscillator=16000000,interrupt=25
```

Toutes ces nouvelles valeurs devront être ajoutées dans le fichier `config.txt` en ajoutant du code dans le script `post-image.sh` de la RPi3. Prendre exemple sur les autres arguments pour ajouter une option `--add-mcp2515-overlay`.

Une fois que le script a été modifié, il faudra ajouter l'option dans la configuration Buildroot (cf variable `BR2_ROOTFS_POST_SCRIPT_ARGS`).



Configuration kernel

Par défaut, le driver pour le contrôleur CAN MCP2515 est déjà activé dans la configuration que l'on utilise du kernel.

Vérifier que la configuration MCP251X est activée (soit `builtin =y` soit en module `=m`) en utilisant la commande suivante :

```
make linux-menuconfig
```

Cela vous ouvrira le menuconfig spécifique au kernel (et non pas à Buildroot).

2.2.2 Configuration de Buildroot

Maintenant que la configuration côté Kernel et device-tree est réalisée, il faut activer des nouveaux paquets dans Buildroot pour pouvoir avoir des outils pour utiliser le CAN :

- **LIBSOCKETCAN** : Permet de contrôler un périphérique CAN via du code en C
- **CAN_UTILS** : Permet d'ajouter des outils en ligne de commande pour tester des périphériques CAN tel que `candump` et `cansend`.

Ouvrir le menuconfig de buildroot et chercher les paquets qui nous intéressent. Les activer, sauvegarder votre configuration et compiler une nouvelle image. Vérifier qu'ils sont bien compilés pour votre RPi3 avec :

```
$ find output/target/ -iname can*
```

```
mylene@dell-desktop:~/projets/buildroot/main ((96e7461800...))$ find output/target/ -iname can*
output/target/usr/bin/cangen
output/target/usr/bin/cansniffer
output/target/usr/bin/cangw
output/target/usr/bin/cansend
output/target/usr/bin/candump
output/target/usr/bin/canplayer
output/target/usr/bin/canbusload
output/target/usr/bin/can-calc-bit-timing
output/target/usr/bin/canlogserver
output/target/usr/bin/canfdtest
output/target/lib/modules/4.19.23-v7/kernel/net/can
output/target/lib/modules/4.19.23-v7/kernel/net/can/can-gw.ko
output/target/lib/modules/4.19.23-v7/kernel/net/can/can-bcm.ko
output/target/lib/modules/4.19.23-v7/kernel/net/can/can-raw.ko
output/target/lib/modules/4.19.23-v7/kernel/net/can/can.ko
output/target/lib/modules/4.19.23-v7/kernel/drivers/net/can
output/target/lib/modules/4.19.23-v7/kernel/drivers/net/can/can-dev.ko
```

```
$ find output/staging/ -iname can*
```

```
mylene@dell-desktop:~/projets/buildroot/main ((96e7461800...))$ find output/staging/ -iname can*
output/staging/usr/include/can_netlink.h
output/staging/usr/include/linux/can
output/staging/usr/include/linux/can.h
```

Maintenant, flasher votre carte SD avec votre nouvelle image, booter votre RPi3 avec le module CAN connecté comme il faut et tester votre interface `can0` avec :

```
$ ip link set can0 up type can bitrate 500000
$ ifconfig
$ cansend can0 01a#11223344AABBCCDD
$ candump can0
```