Développement Linux embarqué : 5 étapes pour commencer

Julien Grossholtz(https://openest.io/author/openest/)
décembre 17, 2019(https://openest.io/2019/12/17/)

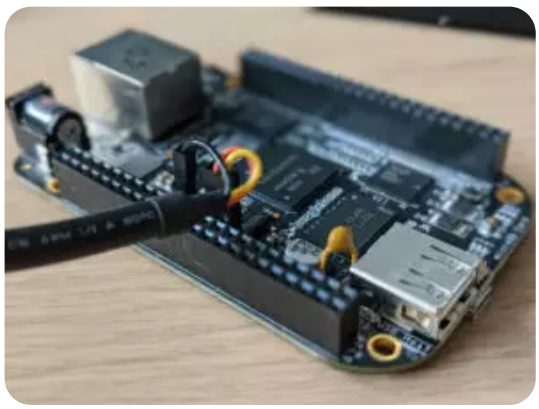
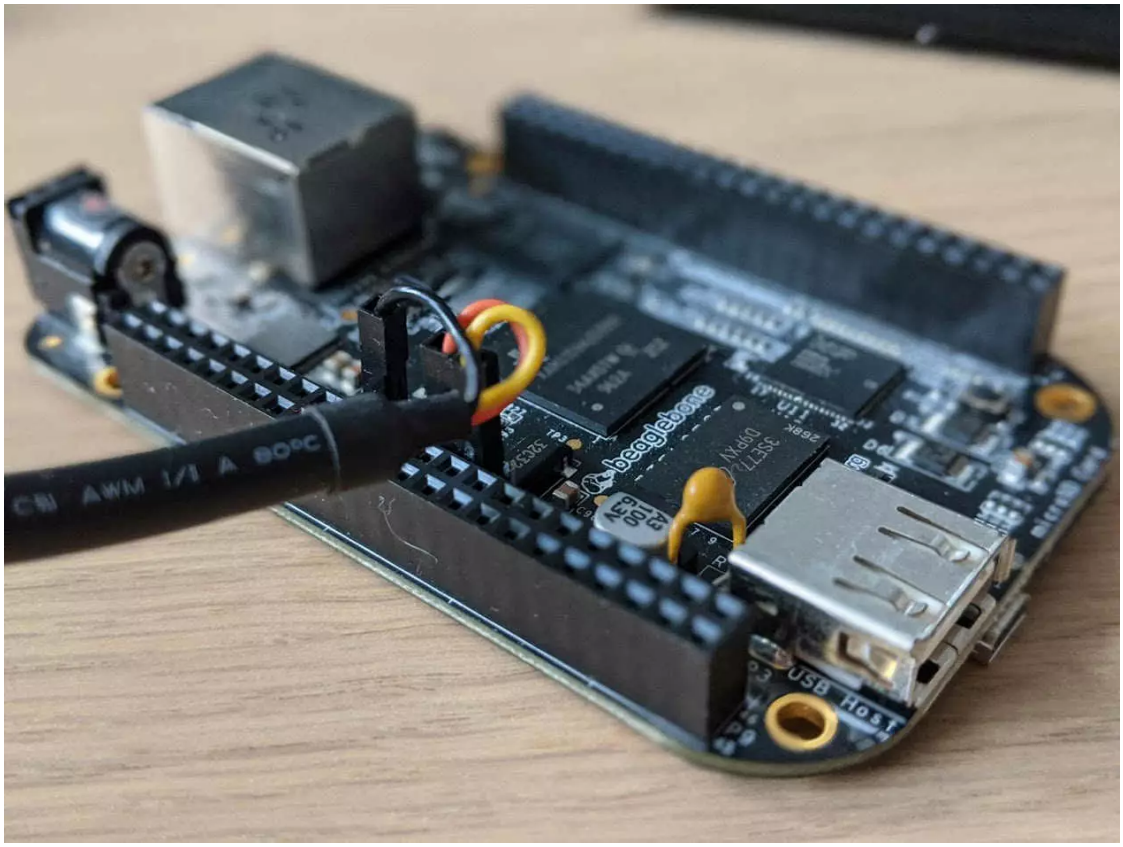


Table des matières

Voici le guide de démarrage rapide pour le développement **Linux embarqué** dont vous avez besoin pour générer un système embarqué de qualité industrielle pour la carte qui se trouve sur votre bureau. Le **développement Linux embarqué** est un domaine vaste et il y a beaucoup de documentation pour vous aider avec tous les détails ce tutoriel veut simplement vous donner des étapes claires pour être efficaces dans la création de votre système.



La Beaglebone Black est une excellente carte pour apprendre Linux embarqué

Construire un système embarqué complet en utilisant Buidroot

Buildroot et Yocto sont les outils Linux embarqué les plus importants. Vous pourriez les comparer à des **distributions Linux embarquées**, mais comme vous le verrez, elles sont bien plus que cela.

Pour ce billet de blog nous utilisons Buildroot. En effet, chez Openest nous pensons qu'il est plus facile à utiliser lorsque l'on débute et beaucoup plus rapide à mettre en place. Si vous préférez Yocto, faites-nous savoir dans les commentaires, nous mettrons probablement à jour cet article plus tard.

Clonons le code source de Buildroot et allons dans ses sources :

```
git clone git://git.buildroot.net/buildroot
cd buildroot
git tag
[...]
```

Explorons les tags git disponibles (actuellement 2019.11) et récupérons cette version :

```
git checkout
```

Nous voulons maintenant trouver votre **Board Support Package** Linux embarquée dans le dossier « board ». Il est classé par fabricant de cartes, c'est donc relativement facile de trouver le vôtre. Pour la suite de ce tutoriel, nous utiliserons la **Beaglebone Black (http://beagleboard.org/black)** que vous trouverez dans le répertoire « board/beaglebone ». De là, je vous suggère de lire le fichier readme.txt que vous trouverez dans ce répertoire.

Maintenant, appliquez la configuration correspondante et construisez votre système :

```
make beaglebone_defconfig
make
```

Dans le cas où vous avez un **Raspberry-Pi 4 (https://www.raspberrypi.org/products/raspberry-pi-4-model-b/)**, il suffit de lancer :

```
make raspberry4_defconfig
make
```

Maintenant vous attendez environ 30 minutes (cela prend plus de temps la première fois), Buildroot va télécharger et compiler tous les logiciels nécessaires. Vous pouvez maintenant passer à l'étape suivante.

Attendez ! Je n'arrive pas à trouver ma carte dans le répertoire « board »! Ok, ce n'est pas de chance. Cela signifie qu'elle n'est pas (encore ?) officiellement supportée dans **Buildroot**. S'il s'agit d'une carte entièrement personnalisée ou d'une carte très rare, ce n'est pas surprenant. Dans ce cas, je vous suggère de trouver une carte avec le même microprocesseur :

```
cd configs
grep -inr "imx6" .
```

Cela vous donnera des cartes avec le même CPU, trouvez-en une qui est similaire à la vôtre. Utilisez-le comme base pour créer votre propre dossier dans board/ et configuration dans config/ modifié pour vos besoins propres.

Démarrer votre système Linux embarqué

À la fin du processus de compilation, vous trouverez les fichiers les plus utiles dans le répertoire output/images :

```
ls output/images
am335x-boneblack.dtb  am335x-evm.dtb    MLO          rootfs.tar
uEnv.txt
am335x-bone.dtb       am335x-evmsk.dtb  rootfs.ext2  sdcard.img
zImage
am335x-bonegreen.dtb  boot.vfat         u-boot.img
```

Quels sont tous les fichiers dans output/images de Buildroot ? Eh bien, il y a beaucoup à dire. Bien souvent, vous trouverez :

- dtb: (device tree binary) files: ils sont générés à partir de fichiers dts : ils décrivent essentiellement le matériel à Linux, lui indiquent où trouver les différents composants présents sur un PCB et quels registres et quels pilotes utiliser.
- zImage : l'image binaire Linux, zImage est un format d'image binaire mais il y en a d'autres tel que uImage.
- u-boot.img : est une image binaire de bootloader, habituellement nous utilisons U-Boot. Il initialise certains composants (généralement DDR, contrôleur de sdcard, et Ethernet).
- rootfs.ext2 : le système de fichiers racine : la partition où Linux s'attend à trouver tous les fichiers nécessaires.
- une image sdcard.img prête à l'emploi qui peut être flashée sur une carte SD.

Dans le cas du Beaglebone Black, vous trouverez également un fichier MLO. Le MLO est le premier programme de démarrage, il effectue l'initialisation de base et démarre ensuite U-Boot.

Comment démarrer la carte :

Insérez une microSD sur votre PC, vérifiez son nom avec dmesg et flashez l'image générée avec dd:

```
sudo dd if=output/images/sdcard.img of=/dev/mmcblk0 bs=2M
sync
```

Attention ! Vérifiez deux fois le périphérique de destination, en particulier assurez-vous qu'il n'est pas sda, sdb : vous pourriez tout à fait effacer le disque dur de votre PC.

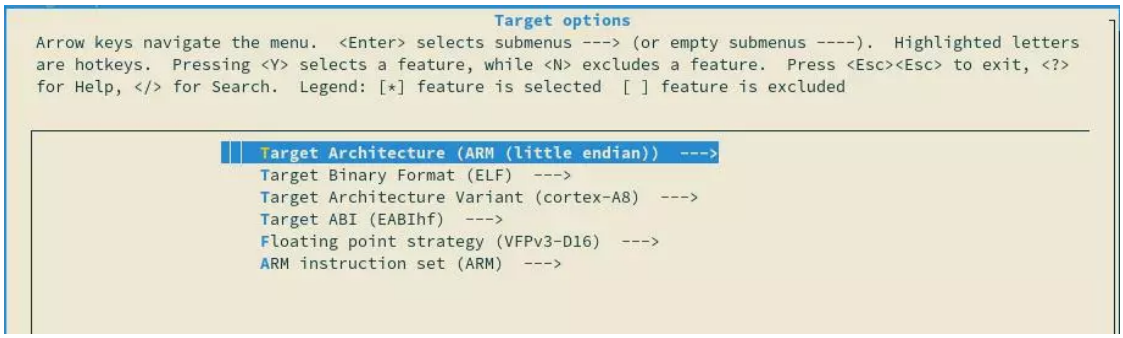
Insérez maintenant la carte SD sur le Beaglebone Black et branchez le bloc d'alimentation. Vous le verrez démarrer sur votre console via l'UART. Vous pouvez maintenant vous connecter à Buildroot. L'utilisateur par défaut est root.

Installer des logiciels supplémentaires sur votre rootfs

Buildroot ne supporte pas seulement des dizaines de cartes matérielles. Il contient également plus de 2300 packages. Le terme package n'est pas tout à fait exact, en fait nous devrions parler de recettes de compilation. De toute façon, vous pouvez ajouter de nouveaux paquets à votre rootfs, sur votre PC retournez dans le répertoire principal de **Buildroot (https://buildroot.org)** :

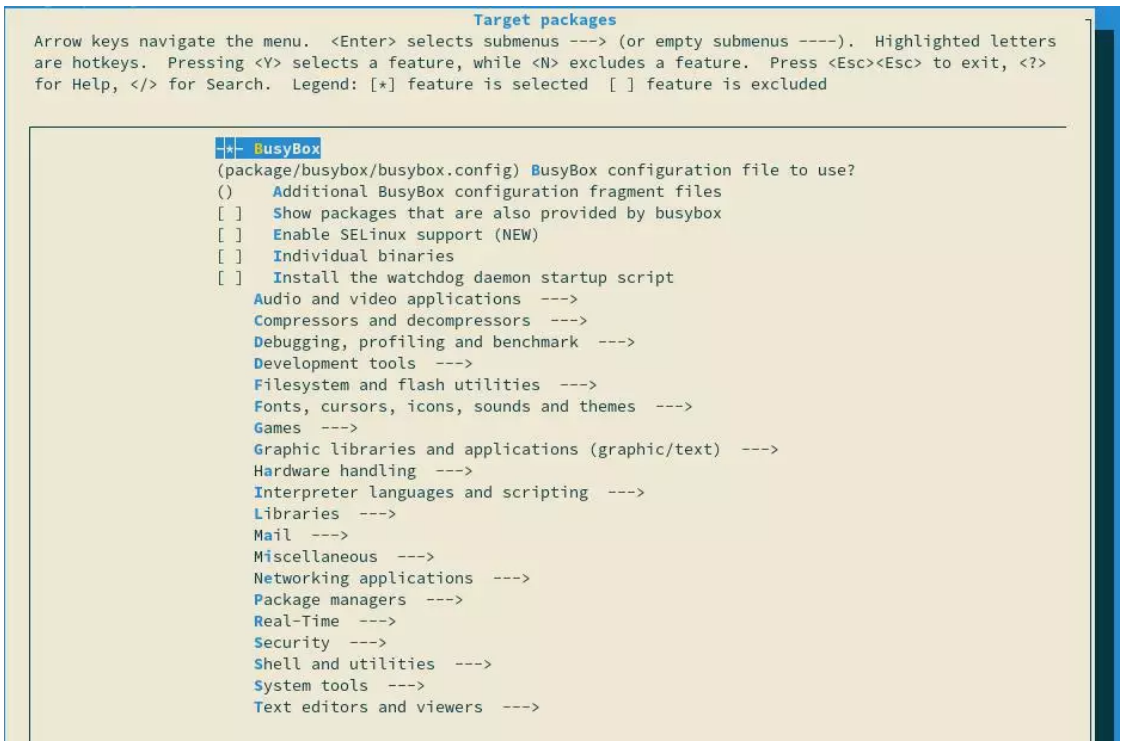
```
make menuconfig
```

Cela ouvrira une interface graphique :



make menuconfig sur Buildroot

A partir du menu principal, vous pouvez explorer les **logiciels embarqués** disponibles, allez dans la section « Target Packages » :



Si vous savez ce que vous voulez rechercher, tapez simplement « / » et entrez le nom de l'application ou de la bibliothèque que vous souhaitez ajouter.

Une fois que vous avez terminé, appuyez sur escape, sauvegardez la nouvelle configuration et réexécutez make une fois que vous êtes de retour sur votre console.

Développer votre propre application embarquée

La meilleure chose avec le développement Linux embarqué est que vous pouvez utiliser n'importe quel langage, de **Java (https://en.wikipedia.org/wiki/Java_(programming_language))** à **Go (https://golang.org/)** et même **javascript (https://en.wikipedia.org/wiki/JavaScript)**. Bien sûr, vous devez prendre en compte les ressources disponibles sur votre carte, mais de nos jours, nous voyons beaucoup de logiciels **nodeJS (https://nodejs.org/en/)** ou

Python (https://www.python.org/) fonctionner sur des systèmes embarqués. Vous pouvez donc choisir votre language préféré ou porter des logiciel existants sur une **plate-forme embarquée**.

Le processus de développement peut varier en fonction de votre projet, mais vous pouvez facilement commencer à développer sur n'importe quel PC Linux et prendre en charge les aspects spécifiques au matériel ultérieurement.

Pour les langues **non compilées**, installez simplement l'outil nécessaire avec l'outil *make menuconfig*. Passez ensuite à la section suivante.

Pour les **langages compilés**, tels que C ou **C++ (https://en.wikipedia.org/wiki/C%2B%2B)**, vous pouvez installer directement un compilateur sur votre système embarqué mais ce ne serait pas efficace. Au lieu de cela, vous pouvez utiliser une Cross toolchain. Une cross toolchain compile des logiciels pour une architecture différente. Dans notre cas, nous allons compiler du code ARM sur un PC.

Où se trouve la chaîne de cross compilation de Buildroot ? Lorsque nous avons généré notre système de fichiers, Buildroot a également compilé sa propre toolchain. Depuis le répertoire principal de Buildroot, vous pouvez la trouver dans output/host :

```
ls output/host/
arm-buildroot-linux-uclibcgnueabihf  bin  etc  include  lib  lib64
libexec  sbin  share  usr
```

Le répertoire output/host est l'endroit où Buildroot place tous les fichiers nécessaires au système **hôte** (votre PC) pour générer le système **cible** (votre carte). Vous trouverez **gcc** et tous ses outils associés dans le répertoire bin.

Le compilateur n'est pas tout. Vous voulez probablement lier votre programme avec les bibliothèques des bibliothèques de votre système embarqué. Ils sont situés dans la output/target » :

```
ls output/target/
bin  etc  lib32    media  opt   root  sbin
THIS_IS_NOT_YOUR_ROOT_FILESYSTEM  usr
dev  lib  linuxrc  mnt    proc  run   sys   tmp
var
```

Il contient tous les fichiers installés sur votre système cible, y compris les binaires de vos bibliothèques et programmes. Vous pouvez ajouter ces chemins à l'option « -L » de votre ligne de commande gcc pour vous assurer que vous pouvez linker avec ces binaires.

Les fichiers d'en-têtes ne sont pas installés dans le répertoire cible, mais vous les trouverez dans « output/build/ »

Déployer votre logiciel sur votre système embarqué

Lorsque vous développez un programme, vous voulez le tester très souvent. Vous avez donc besoin d'un workflow efficace qui vous permette de transférer rapidement votre programme et vos données.

Il y a plusieurs options sur la table, vous pouvez essayer **NFS (https://fr.wikipedia.org/wiki/Network_File_System)**, ou **SSHFS (https://fr.wikipedia.org/wiki/Secure_shell_file_system)** mais ils sont un peu plus difficiles à configurer. Au lieu de cela, nous utiliserons simplement la commande scp.

Comment installer un serveur ssh sur votre système embarqué ? Encore une fois, utilisez l'outil de configuration de Buildroot : make menuconfig. Maintenant vous pouvez chercher **dropbear**, un petit serveur ssh conçu pour les systèmes embarqués. Vous le trouverez dans « paquetages cibles/applications/réseautage/ours-poussoir » :

Dropbear est situé dans « target packages/Networking applications/dropbear », il suffit de le sélectionner.

Pour utiliser ssh, nous devons faire en sorte que le réseau soit fonctionnel. La façon la plus simple d'y parvenir est d'utiliser Ethernet. Si vous n'avez pas de port Ethernet sur votre carte, vous pouvez utiliser Wi-Fi avec wpa-supplicant à la place. Nous pouvons dire à Buildroot de s'assurer que l'interface réseau par défaut reçoit une adresse IP par DHCP dans le menu « Configuration système/interfaces réseau à configurer par DHCP » :

Saisissez ici le nom de l'interface réseau par défaut. Souvent, c'est eth0.

Maintenant vous pouvez lancer make pour construire dropbear et re-générer l'ensemble de vos rootfs. Ensuite, flashez la carte SD et démarrez le système. Il obtiendra une adresse IP de votre serveur DHCP et le dropbear sera démarré. Notez l'adresse IP de votre interface :

```
ip address show eth0
```

Maintenant, tout ce que nous avons à faire est de transférer votre fichier exécutable vers le système embarqué. Sur votre PC, exécutez :

```
scp my_program root@IP_ADDRESS:/root
```

De retour sur le système embarqué, vous pouvez maintenant exécuter votre programme :

```
cd /root
./myprogram
```

Aller plus loin dans le développement Linux embarqué

Le développement Linux embarqué est un vaste domaine d'expertise. Il faut des années aux experts en Linux embarqué pour le maîtriser.

Veuillez noter que ce tutoriel ne mentionne pas la sécurité, bien sûr, ce ne serait pas la meilleure idée de tout exécuter en tant qu'utilisateur root ou d'installer dropbear sur le produit final ! Par contre c'est assurément un bon moyen pour débuter.

Néanmoins, nous espérons que cet article vous a donné quelques-unes des clés dont vous aviez besoin. Selon vos besoins, nous vous suggérons de lire nos autres articles de blog, sur **les mises à jour des systèmes embarqués**

(https://openest.io/2018/09/18/comment-mettre-a-jour-un-systeme-embarque/) ou sur la façon dont vous devriez **choisir le matériel (https://openest.io/2019/06/07/choisir-un-som-sbc-en-fonction-du-support-logiciel/)** pour assurer une bonne **intégration matériel/logiciel (https://openest.io/conseil-en-conception-electronique/)**.

Faites-nous savoir ce que vous en pensez dans la section commentaires. Nous cherchons à améliorer nos tutoriels et toute remarque est la bienvenue !

Partager cet article

Articles Connexes

Linux Embarqué

Utilisez le MPU6050 sur Raspberry-Pi (https://openest.io/2020/01/21/utiliser-mpu6050-raspberry-pi/)

Utiliser le MPU6050 sur Raspberry-Pi Voici comment utiliser le MPU6050 sur Raspberry-Pi ou sur tout système Linux embarqué. Les MPU-6050 & MPU-6000 sont des composants

Julien Grossholtz · 21 janvier 2020