

Support du matériel

Christophe BLAESS

christophe.blaess@logilin.fr

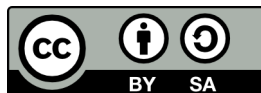
<https://www.blaess.fr/christophe/>



Ingénierie et formations sur Linux et les logiciels libres
<https://www.logilin.fr>

Support de la machine.....	3
Fichier de configuration.....	3
Travaux pratiques : image pour Raspberry Pi.....	4
Travaux pratiques : personnalisation de machine.....	5
Bootloader.....	8
Travaux pratiques : patch sur U-boot.....	10
Kernel.....	12
Version du noyau.....	13
Configuration du noyau.....	14
Travaux pratiques : configuration du kernel.....	15
Device Tree.....	19
Travaux pratiques : modification du device tree.....	20
Partitionnement amélioré.....	22
Travaux pratiques : ajout d'une partition de données.....	23

Ce support de formation est distribué sous licence **Creative Commons 4.0**



(Attribution - Partage dans les mêmes conditions).

Vous êtes libres de copier et partager ce document, en mentionnant son origine. Si vous l'intégrez dans un contenu plus vaste, ce dernier devra être distribué avec les mêmes droits.

Ce cours a été rédigé en utilisant des logiciels libres sur système d'exploitation Linux :

- *LibreOffice Writer* pour le support et la mise en page
- *Gimp* pour les images bitmap
- *Excalidraw* (en ligne) pour les schémas vectoriels.

Yocto Project avancé

YPA v. 1.2

<https://www.blaess.fr/christophe/>

<https://www.logilin.fr>

Support de la machine

Fichier de configuration

Au moment du *build*, le choix de la cible se fait dans ``local.conf`` à l'aide de la variable ``MACHINE``.

La configuration correspondante se trouve dans un fichier ``.conf`` dans le sous-répertoire ``conf/machine`` d'un layer. On y trouve entre autres :

- La configuration des ``MACHINE_FEATURES`` (``alsa``, ``bluetooth``, ``keyboard``, ``rtc``, ``screen``, ``touchscreen``, ``wifi``, etc.).
- La liste ``MACHINE_EXTRA_RRECOMMENDS`` des packages facultatifs spécifiques à cette machine.
- La liste ``MACHINEOVERRIDES`` (avec le séparateur deux-points ``:``) contenant des éléments de sélection dans les recettes ultérieures. Cette liste est ajoutée à la variable globale ``OVERRIDES``.
- La recette fournissant le kernel (``PREFERRED_PROVIDER_virtual/kernel``), son fichier de configuration (``KBUILD_DEFCONFIG``) et le nom du *device tree* à compiler dans ``KERNEL_DEVICETREE``.
- Les fichiers d'images désirés en sortie (``IMAGE_FSTYPES``), les plus courants étant ``tar.bz2``, ``wic``, ``wic.bz2``, ``ext2``, ``ext4`` etc.

- ...

Travaux pratiques : image pour Raspberry Pi

- ➔ Téléchargez le layer contenant les éléments de configuration du Raspberry Pi 4

```
$ cd layers/  
$ git clone git://git.yoctoproject.org/meta-raspberrypi -b kirkstone  
$ cd ..
```

- ➔ Préparez un répertoire de build spécifique

```
$ source layers/poky/oe-init-build-env builds/build-rpi4
```

- ➔ Ajoutez les layers nécessaires

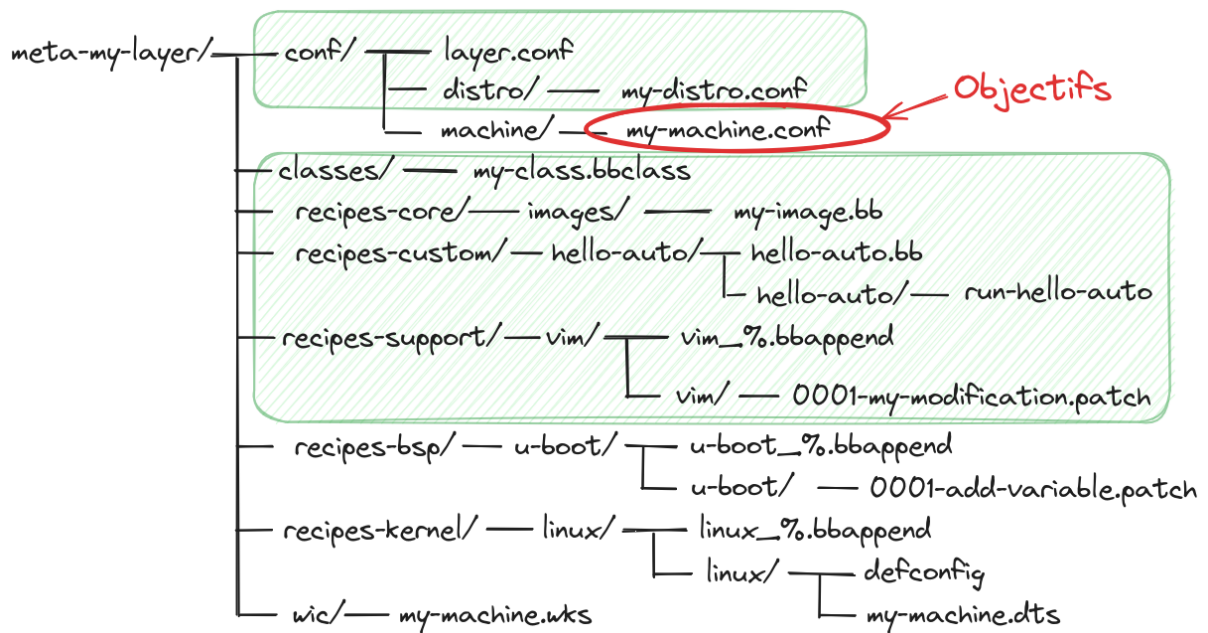
```
$ bitbake-layers add-layer ../../layers/meta-openembedded/meta-oe/  
$ bitbake-layers add-layer ../../layers/meta-raspberrypi/  
$ bitbake-layers add-layer ../../layers/meta-my-layer/
```

- ➔ Éditez le fichier `local.conf` et ajoutez :

```
MACHINE = "raspberrypi4"  
ENABLE_UART = "1"  
DL_DIR ?= "${TOPDIR}/../../downloads"  
SSTATE_DIR ?= "${TOPDIR}/../../sstate-cache"
```

- ➔ Compilez l'image, installez-la sur une carte micro-SD et vérifiez que le Raspberry Pi démarre correctement.

Travaux pratiques : personnalisation de machine



Exercice sur cible Raspberry Pi 4

- Copiez le fichier de configuration du Raspberry Pi 4 sous le nom ``my-machine.conf`` dans le layer ``meta-my-layer``.
- Éditez le fichier pour préciser le nom du fichier de configuration à utiliser pour la configuration du kernel : ``bcm2711_defconfig``
- Modifiez également le type d'image de sortie à ``wic`` uniquement.

Enfin, ajoutez la ligne

```
|| ENABLE_UART = "1"
```

- Mettez à jour dans ``conf/local.conf`` la variable ``MACHINE``

Solution

```
$ mkdir -p ../../layers/meta-my-layer/conf/machine/

$ cp ../../layers/meta-raspberrypi/conf/machine/raspberrypi4.conf
  ../../layers/meta-my-layer/conf/machine/my-machine.conf

$ nano ../../layers/meta-my-layer/conf/machine/my-machine.conf
[...]
KBUILD_DEFCONFIG = "bcm2711_defconfig"
IMAGE_FSTYPES = "wic"
ENABLE_UART = "1"

$ nano conf/local.conf
[...]
MACHINE = "my-machine"
[...]

$ bitbake my-image

$ ls tmp/deploy/images/my-machine/my-image
[...]
tmp/deploy/images/my-machine/my-image-my-machine.wic
```

Poky (Yocto Project Reference Distro) 4.0.17 my-machine /dev/ttyS0

Exercice sur cible Qemuarm

- ➔ Copiez le fichier de configuration de l'émulateur `qemuarm` sous le nom `my-machine.conf` dans le layer `meta-my-layer`.
- ➔ Éditez ce fichier pour préciser le nom de la machine à utiliser comme configuration du kernel : `qemuarm15`
Modifiez le paramètre `QB_SMP` de Qemu pour émuler 8 cœurs au lieu de 4.
Ajoutez une ligne de paramètre pour augmenter la RAM de la cible à 1Gb :
|| QB_MEM = "-m 1024"
- ➔ Créez dans votre layer une recette d'extension pour `linux-yocto` avec la ligne
|| COMPATIBLE_MACHINE = "my-machine"
- ➔ Mettez à jour dans `conf/local.conf` la variable `MACHINE`
Recompilez l'image, et vérifiez sur la cible le nombre de cœurs de CPU émuls et la mémoire disponible.

Solution

```
$ mkdir ../../layers/meta-my-layer/conf/machine

$ cp ../../layers/poky/meta/conf/machine/qemuarm.conf ../../layers/meta-my-layer/conf/machine/my-machine.conf

$ nano ../../layers/meta-my-layer/conf/machine/my-machine.conf
|| QB_SMP = "-smp 8"
|| KMACHINE:my-machine = "qemuarm15"

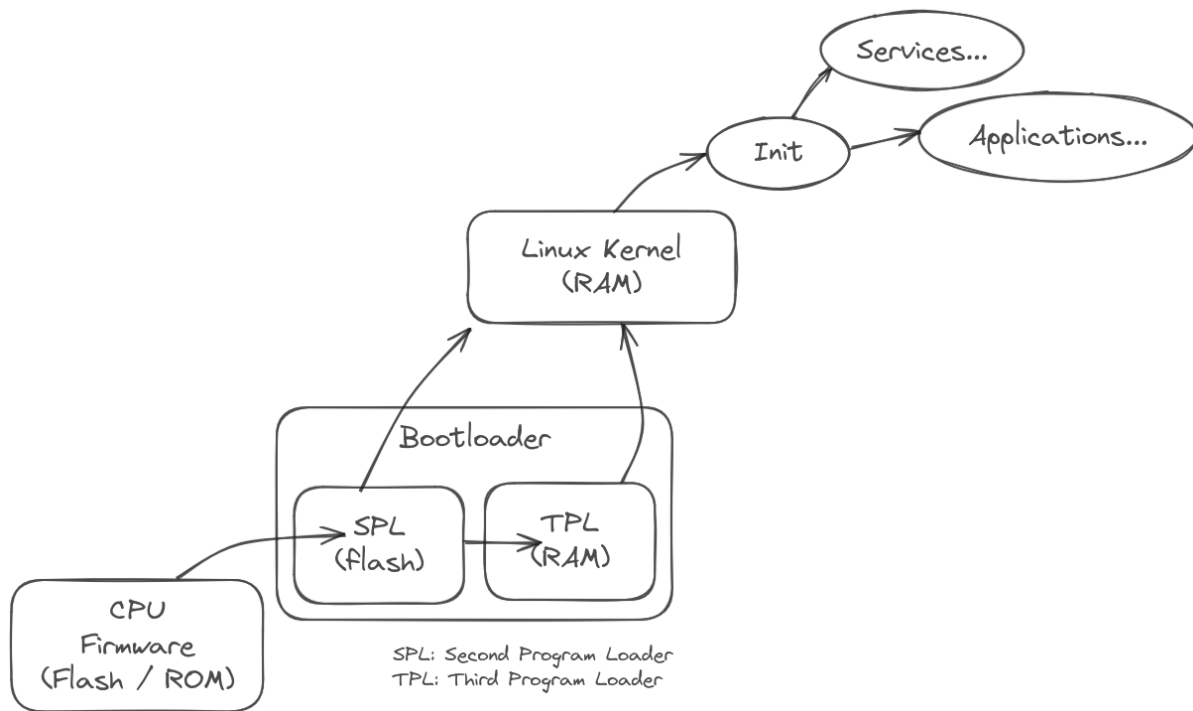
$ mkdir ../../layers/meta-my-layer/recipes-kernel/linux/ -p
$ nano ../../layers/meta-my-layer/recipes-kernel/linux/linux-yocto_%.bbappend
|| COMPATIBLE_MACHINE = "my-machine"

$ nano conf/local.conf
|| MACHINE = "my-machine"
||

$ bitbake core-image-base
$ runqemu nographic
Poky (Yocto Project Reference Distro) 4.0.17 my-machine /dev/ttyAMA0
[...]
root@my-machine:~# cat /proc/cpuinfo
processor      : 0
[...]
processor      : 7
```

Bootloader

Le choix du bootloader est généralement intégré dans le fichier de configuration de la machine.



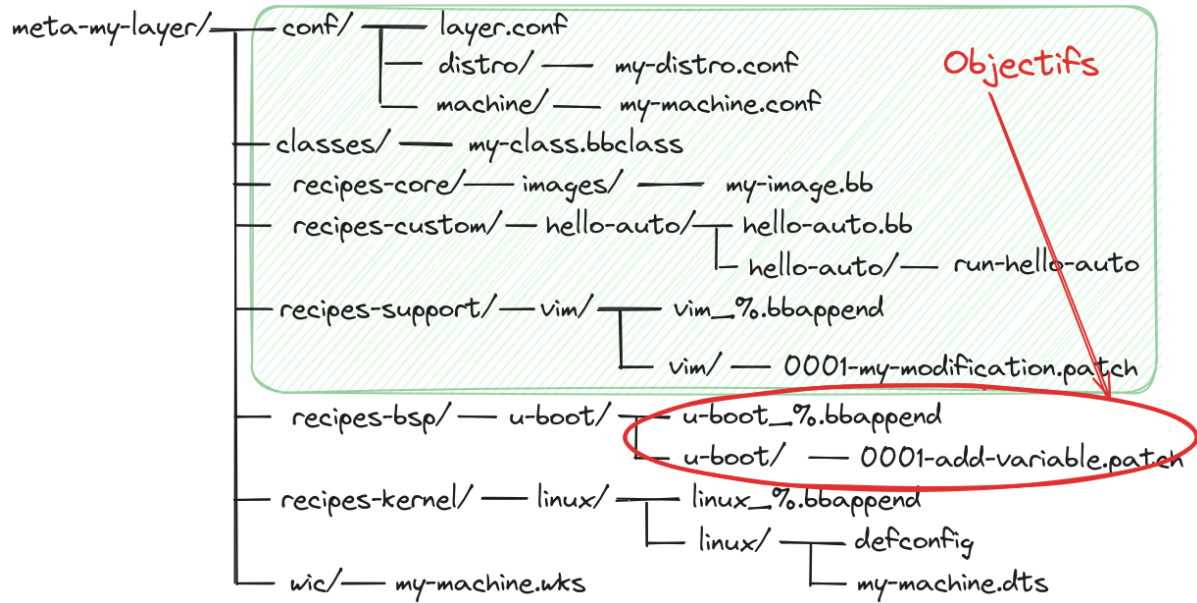
On est souvent amené à patcher U-boot pour modifier l'environnement par défaut.

Par exemple en ajoutant ``logo.nologo`` et ``loglevel=0`` sur la ligne d'arguments du kernel, on masque les messages de boot.

Pour le Raspberry Pi, si la variable ``RPI_USE_U_BOOT`` est égale à 1, l'intégration de U-boot est automatique :

```
RPI_USE_U_BOOT = "1"
```

Travaux pratiques : patch sur U-boot



Exercice sur Raspberry Pi

→ Utilisez ``devtool`` pour créer un environnement de *patch* pour U-boot.

→ Éditez le fichier ``include/configs/rpi.h`` et ajoutez la ligne

```
|| "myvar=myvalue\0" \
```

dans la variable ``CONFIG_EXTRA_ENV_SETTINGS`` en fin de fichier.

→ *Commitez* la modification, enregistrez le patch et recompilez l'image.

→ Vérifiez au prompt de U-boot que la variable a bien été définie.

Kernel

Pour la configuration du noyau Linux, plusieurs points sont à prendre en compte :

- choix de la version du kernel,
- ajout éventuel de patches (par exemple adaptation de driver pour un périphérique custom),
- configuration du noyau (choix des drivers à intégrer),
- configuration et modification du device tree.

Version du noyau

Le choix de la branche du kernel est généralement fait dans le fichier `.conf` de description de la machine.

Quelques possibilités :

- ``linux-yocto``, (<https://git.yoctoproject.org/linux-yocto>), layer ``poky/meta``) : noyau standard
- ``linux-yocto-rt`` (<https://git.yoctoproject.org/linux-yocto>, branche ``preempt-rt-base``, layer ``poky/meta``) : noyau avec patch PREEMPT_RT appliqué.
- ``linux-raspberrypi`` (<https://github.com/raspberrypi/linux>, layer ``meta-raspberrypi``) : noyau avec patches spécifiques pour les Raspberry Pi.
- ``linux-imx`` (<https://github.com/nxp-imx/linux-imx>, layer ``meta-freescale``) : noyau avec patches pour les *Systems-On-Chips* i.MX6/i.MX8.
- ``linux-toradex-mainline`` (<https://git.kernel.org/pub/scm/linux>, layer ``meta-toradex``) noyau standard et patches pour le support des *Systems-On-Module* Toradex.
- ``linux-mainline`` (<https://git.phytec.de/linux-mainline>, layer ``meta-phytec``) : noyau standard avec patches pour *Systems-On-Module* Phytec.
- ``linux-stm32mp`` (<https://github.com/STMicroelectronics/linux>, layer ``meta-st-stm32mp``) : noyau avec patches pour le *System-On-Chip* STM32mp15x.

Exemple pour le Raspberry Pi :

```
$ cat ../../layers/meta-raspberrypi/conf/machine/raspberrypi4.conf
[...]
include conf/machine/include/rpi-base.inc
[...]

$ cat ../../layers/meta-raspberrypi/conf/machine/include/rpi-base.inc
include conf/machine/include/rpi-default-settings.inc
include conf/machine/include/rpi-default-versions.inc
include conf/machine/include/rpi-default-providers.inc
[...]

$ cat ../../layers/meta-raspberrypi/conf/machine/include/rpi-default-providers.inc
[...]
PREFERRED_PROVIDER_virtual/kernel ??= "linux-raspberrypi"
```

Configuration du noyau

La configuration du noyau consiste surtout à choisir les drivers à compiler.

On peut également activer ou désactiver certaines fonctionnalités propres du noyau.

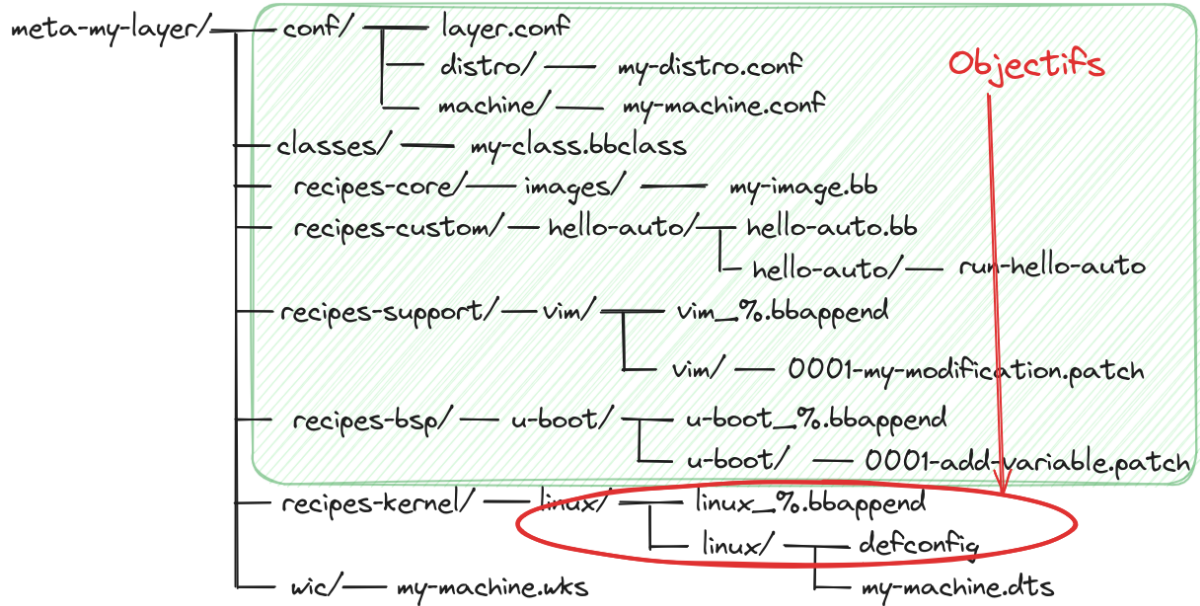
Les drivers peuvent être :

- ``[]`` : ignorés totalement
- ``[*]`` : compilés statiquement dans l'image du kernel chargée en mémoire au démarrage
- ``<M>`` : compilés sous formes de fichiers externes (des modules du noyau) qui seront chargés dynamiquement selon les besoins de fonctionnement.

La configuration sert à produire un fichier qui servira lors de la compilation du kernel.

On peut éventuellement fournir à ``bitbake`` des fichiers fragments de configuration pour modifier celle-ci.

Travaux pratiques : configuration du kernel



Nous allons activer quelques options de configuration du *kernel*.

→ Ouvrez le menu de configuration :

```
$ bitbake -c menuconfig virtual/kernel
```

(Le menu contient des milliers d'options.)

→ Configurez les options suivantes comme suit :

General setup --->

Preemption Model --->

(X) Preemptible Kernel (Low-Latency Desktop)

Networking support --->

Networking options --->

< > The IPv6 protocol --->

Devices Drivers --->

Network device support --->

Ethernet driver support --->

[*] Amazon devices

<M> Elastic Network Adapter (ENA) support

Kernel hacking --->

printk and dmesg options --->

(1) Default console loglevel (1-15)

→ Quittez le menu en validant la sauvegarde.

Méthode d'installation pour Raspberry Pi

- Créez un fichier `defconfig` et placez-le dans un répertoire d'extension pour le kernel de la machine cible :

```
$ bitbake -c savedefconfig virtual/kernel
[...]  
Saving defconfig to:  
/home/trainings/builds/build-[...]/defconfig
```

```
$ mkdir -p ../../layers/meta-my-layer/recipes-kernel/linux/files
```

```
$ cp /home/trainings/builds/build-[...]/defconfig ../../layers/meta-my-layer/recipes-kernel/linux/files/
```

- Ajoutez une extension de recette pour prendre en considération le `defconfig` :

```
$ nano ../../layers/meta-my-layer/recipes-kernel/linux/linux-raspberrypi_%.bbappend  
|| FILESEXTRAPATHS:prepend := "${THISDIR}/files:"  
|| SRC_URI += "file://defconfig"  
|| KBUILD_DEFCONFIG:forcevariable = ""
```

- Lancez la compilation après avoir nettoyé le répertoire de *build* du kernel :

```
$ bitbake -c clean virtual/kernel  
$ bitbake my-image
```

Méthode d'installation pour Qemu

→ Extrayez un fichier de différences par rapport à la configuration précédente :

```
$ bitbake -c diffconfig virtual/kernel  
[...]  
Config fragment has been dumped into:  
/home/training/builds/build-qemuarm/[...]/fragment.cfg
```

→ Intégrez le fragment dans votre layer :

```
$ recipetool appendsrcfile ../../layers/meta-my-layer/ linux-yocto-  
rt /home/training/builds/build-qemuarm/[...]/fragment.cfg
```

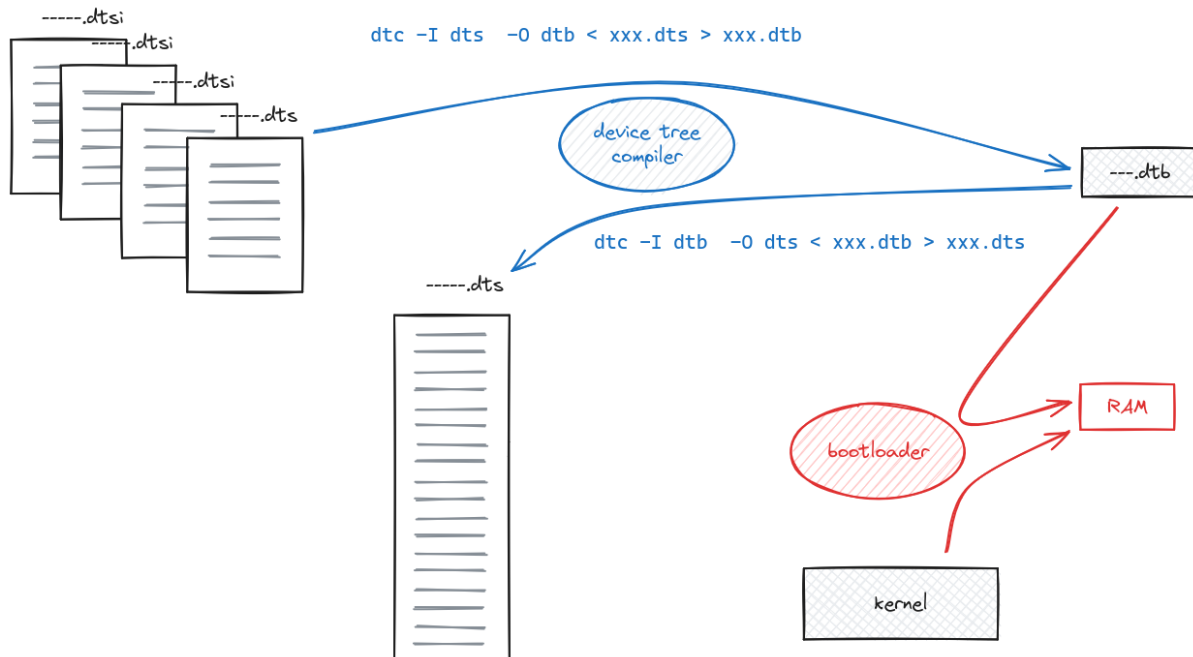
→ Nettoyez le répertoire de build du noyau :

```
$ bitbake -c clean virtual/kernel
```

→ Re-générez l'image:

```
$ bitbake my-image
```

Device Tree

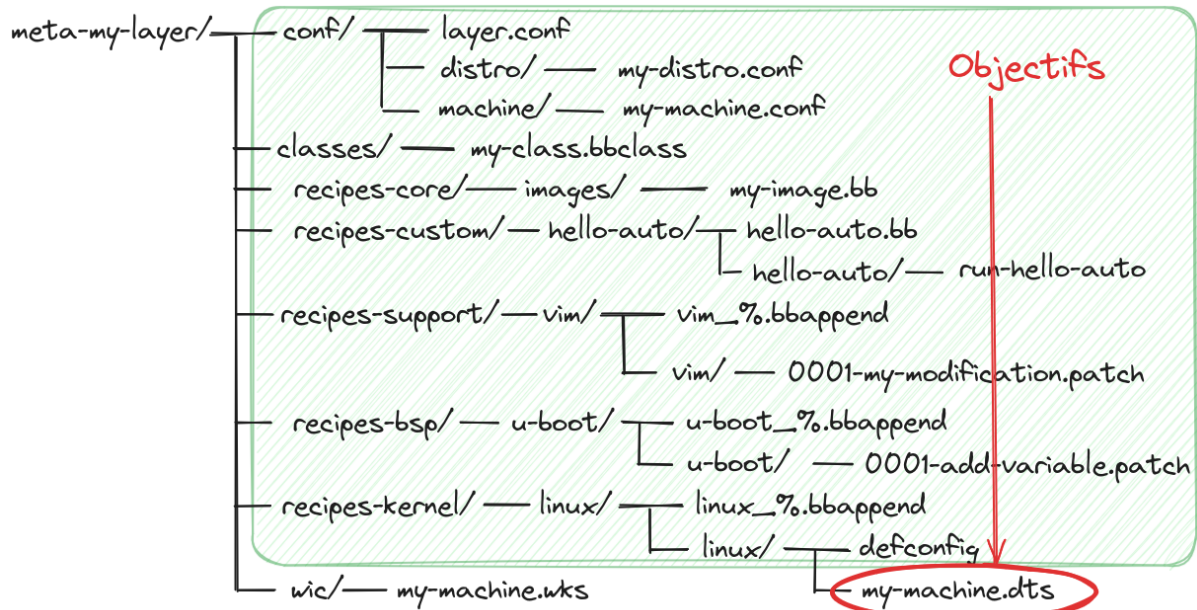


Le *device tree* contient la description du matériel présent, sur les architectures ne permettant pas d'obtenir ces informations directement (BIOS).

Conférence « *Device Tree : hardware description for everybody!* » Thomas Petazzoni – Live Embedded Event 2020 - <https://www.youtube.com/watch?v=Nz6aBffv-Ek>

Travaux pratiques : modification du device tree

Exercice sur Raspberry Pi



De nombreux *device trees* sont présents dans les sources du kernel, dans le répertoire ``arch/<architecture>/boot/dts/``, c'est le cas pour le Raspberry Pi.

- Copiez le *device tree* d'origine du Raspberry Pi 4 dans votre layer en modifiant son nom :

```
$ cp tmp/work-shared/my-machine/kernel-source/arch/arm/boot/dts/bcm2711-rpi-4-b.dts ../../layers/meta-my-layer/recipes-kernel/linux/files/my-machine.dts
```

```
$ nano ../../layers/meta-my-layer/recipes-kernel/linux/linux-raspberrypi%.bbappend
```

```
[...]
SRC_URI += "file://my-machine.dts"

do_compile:prepend() {
    cp ${WORKDIR}/my-machine.dts ${S}/arch/arm/boot/dts/
}
```

- Éditez le fichier `my-machine.dts` et ajoutez les lignes suivantes à la fin du fichier :

```
$ nano ../../layers/meta-my-layer/recipes-kernel/linux/files/my-machine.dts
```

```
[...]
/ {
    model = "My Own RPI4-based Machine";
    leds {
        led-pwr {
            linux,default-trigger = "heartbeat";
        };
    };
};
```

- Indiquez dans la configuration machine le nom du device tree souhaité :

```
$ nano ../../layers/meta-my-layer/conf/machine/my-machine.conf
```

```
[...]
KERNEL_DEVICETREE = "my-machine.dtb"
```

- Après compilation et installation sur la carte SD, éditez le fichier `config.txt` se trouvant sur la carte SD et ajoutez :

```
device_tree=my-machine.dtb
```

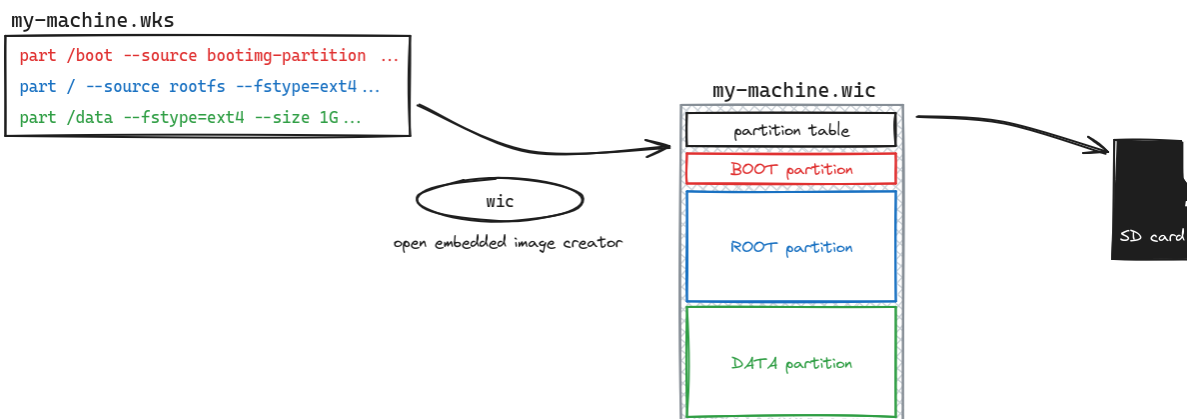
Si vous avez installé U-boot, interrompez son démarrage et saisissez :

```
U-Boot> setenv fdtfile my-machine.dtb
U-Boot> saveenv
Saving Environment to FAT... OK
U-Boot> reset
```

Partitionnement amélioré

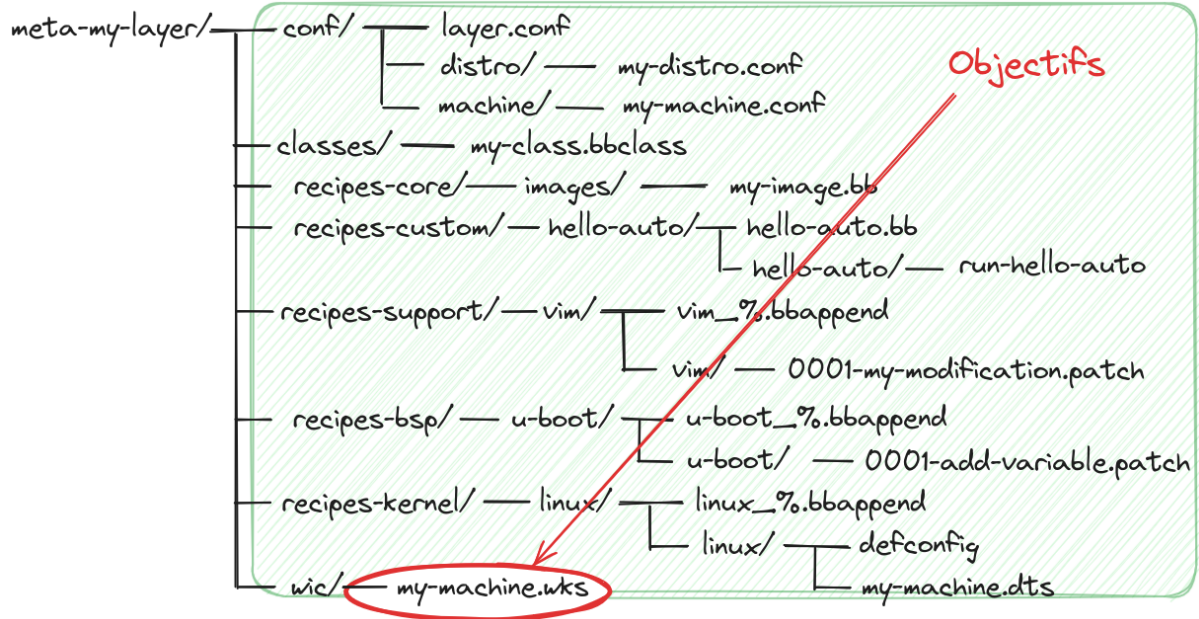
Le fichier `.wks` (*Wic Kickstart Source*) doit se trouver dans un répertoire `wic/` au sommet du layer.

Ce fichier permet d'obtenir un fichier `.wic` qui est une image complète du périphérique de stockage de la cible, comprenant une table de partitions et une ou plusieurs partitions.



Travaux pratiques : ajout d'une partition de données

Exercice sur Raspberry Pi



- Créez un répertoire de stockage pour votre fichier `.wks` et copiez-y le fichier source du layer `meta-raspberrypi` :

```
$ mkdir ../../layers/meta-my-layer/wic
$ cp ../../layers/meta-raspberrypi/wic/sdimage-raspberrypi.wks
  ../../layers/meta-my-layer/wic/my-sdcard.wks
```

- Éditez le fichier pour ajouter une ligne de description de partition.

```
$ nano ../../layers/meta-my-layer/wic/my-sdcard.wks
part /boot --source bootimg-partition --ondisk mmcblk0 --fstype=vfat --
label boot --active --align 4096 --size 20

part / --source rootfs --ondisk mmcblk0 --fstype=ext4 --label root --align
4096

part /data --ondisk mmcblk0 --fstype=ext4 --label data --align 4096 --size
128M
```

- Indiquez le nom du fichier `.wks` dans la configuration de la machine :

```
$ nano ../../layers/meta-my-layer/conf/machine/my-machine.conf
[... ]
WKS_FILE = "my-sdcard.wks"
```