GOCONCEPT
a step ahead

INTRODUCTION

À

YOCTO

Écrit par

Anirudh THABJUL

Consultant Électronique

GOCONCEPT
a step ahead

TABLE DES MATIÈRES

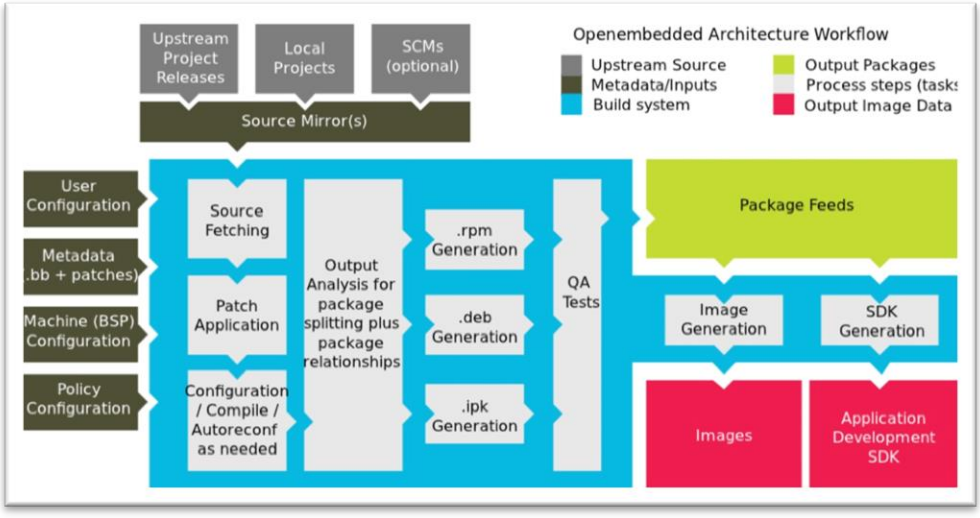
QU'EST-CE QUE YOCTO ?

Le Projet d'Yocto est un projet de collaboration open source qui aide des développeurs à créer un noyau personnalisé Linux pour des produits embarqués, indépendamment de l'architecture de Hardware (Par Exemple ARM, MIPS, PowerPC, and x86 etc).

POURQUOI L'YOCTO A ETE MIS EN PLACE ?

Expliquez en termes simples, Vous pouvez inclure ou exclure des composants créant un système qui répond à vos besoins précisément. Le système de fin peut démarrer juste votre application. Il peut prendre en charge vos composants hardware ou votre environnement de communication et puisque tout est open source, vous avez le contrôle total.

Expliquez en termes techniques, L'Yocto consiste en modèles, aussi appelés des « recettes », qui décrivent comment construire des composants, comme des bibliothèques et des utilités, qui sont incluses dans le système Linux embarqué. Ces recettes peuvent être groupées dans des « Couches » qui promeut la gestion de sortie dans des unités seules plutôt que maintenir qu'une série complexe des révisions de « build » pour chaque bibliothèque et outil. Les recettes et les couches sont regroupés en « images » qui sont des instantanés du noyau « Root File System » pour flasher sur un périphérique.



LES AVANTAGES DU YOCTO

Voici quelques points saillants pour le projet Yocto

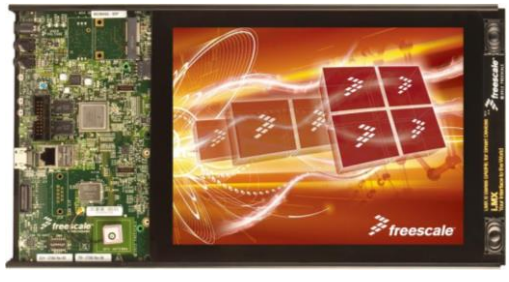
- Fournit un noyau Linux récent avec un ensemble de commandes de système et de bibliothèques appropriées pour l'environnement embarqué.
- Rend les composants du système tels que
  1. X11
  2. GTK +
  3. Qt
  4. Clutter etc ...

  pour vous permettre de créer une expérience utilisateur enrichie sur les appareils dotés d'affichage. Pour les périphériques qui n'ont pas d'affichage ou où vous voulez utiliser l'alternative « UI framework », ces composants n'a pas besoin d'être installée.
- Crée un noyau concentré et stable compatible avec le projet "OpenEmbedded" avec lequel vous pouvez créer et développer facilement et de manière fiable.
- Prend entièrement en charge une large gamme d'émulation de hardware et de périphérique via « Quick EMUlator (QEMU) ».
- Fournit un mécanisme de couche qui vous permet de facilement prolonger le système, faire des personnalisations et les garder organisés.

Consulter le lien suivant pour plus d'informations → Yocto Project

LE HARWARE UTILISÉ

Nous avez pris prendre le kit d'évaluation i.MX6dlsabresd qui utilise le processeur i.MX6 dual-lite comme la référence.



LA PROCEDURE POUR CONSTRUIRE ET FLASHER UNE IMAGE

Ceux-ci sont les étapes principaux pour construire l'image

1. Tout d'abord, configurez le système hôte
2. Installation de l'utilité de repo
3. Installation de git
4. Construire une image minimale par exemple « core-image-minimal »
5. Flasher l'image construite sur la carte SD

CONFIGURER LE SYSTÈME HÔTE

- On le recommande que fournissez au moins 120Gb la mémoire, qui est assez pour compiler n'importe quel Back-end (par exemple x11).
- Ceux-là sont les Distributions Linux supportées

    1. Ubuntu 14.10
    2. Ubuntu 15.04
    3. Ubuntu 15.10
    4. Ubuntu 16.04 (LTS)
    5. Fedora release 22
    6. Fedora release 23
    7. CentOS release 7.x
    8. Debian GNU/Linux 8.x (Jessie)
    9. Debian GNU/Linux 9.x (Stretch)
    10. OpenSUSE 13.2
    11. OpenSUSE 42.1
    12. Linux Mint (Ce n'est pas une distribution officielle pour yocto, mais marche bien)
- Ci-dessous sont les paquets hôtes essentiels d'yocto prévoient que vous devez installer avant que vous ne commenciez à construire les images

```
$ sudo apt-get install gawk wget git-core diffstat unzip texinfo gcc-multilib
build-essential chrpath socat libsdl1.2-dev xterm sed cvs subversion coreutils
texi2html docbook-utils python-pysqlite2 help2man make gcc g++ desktop-file-
utils libgl1-mesa-dev libglu1-mesa-dev mercurial autoconf automake groff curl
lzop asciidoc
```

Pour Ubuntu 12.04

```
$ sudo apt-get install uboot-mkimage
```

Pour Ubuntu 14.04

```
$ sudo apt-get install u-boot-tools
```

INSTALLATION DE L'UTILITÉ DE REPO

Repo est un outil construit sur « Git » qui fait plus facile de gérer les projets qui contiennent des dépôts multiples, qui ne doivent pas être sur le même serveur. Repo complète très bien la nature en couches du projet Yocto, facilitant ainsi l'ajout de ses propres couches au BSP.

Pour installer l'utilité « repo », exécutez ces étapes

1. Créez un dossier « bin » dans votre dossier « home »

```
$ cd ~ (This takes you to home directory)
$ mkdir ~/bin (this step may not be needed if the bin folder already exists)
$ curl http://commondatastorage.googleapis.com/git-repo-downloads/repo >
~/bin/repo
$ chmod a+x ~/bin/repo
```

2. Ajoutez la ligne suivante au fichier « .bashrc » pour vous assurer que le dossier « ~/bin » se trouve dans votre variable PATH

```
$ export PATH=~/bin:$PATH
```

INSTALLATION DE GIT

Exécutez ces étapes pour l'installation

```
$ sudo apt-get install git
$ git config --global user.name "Your Name"
$ git config --global user.email "Your Email"
$ git config –list
```

Consulter le lien suivant pour plus d'informations → Git Installation

CONSTRUIRE UNE IMAGE MINIMALE

Tout d'abord, nous avons besoin de construire une image avec des paquets minimum installés pour démarrer c'est-à-dire l'image « core-image-minimal ». exécutez ces étapes

1. Les couches du « Yocto Project" sont téléchargées dans le répertoire des sources. Cela configure les recettes utilisées pour construire le projet.

```
$ cd ~ (Si vous voulez télécharger dans le dossier « home »)
$ mkdir imx-yocto-bsp (Le nom du dossier peut également être modifié)
$ cd imx-yocto-bsp
$ repo init -u https://source.codeaurora.org/external/imx/fsl-arm-yocto-bsp -b imx-morty (consulter le lien hypertexte pour les nouvelles versions)
$ repo sync
```

2. Configurez le dossier « build » en entrant la commande suivante

```
$ MACHINE=<machine name> DISTRO=<distro name> source fsl-setup-release.sh -b <build dir>
```

Pour notre hardware c'est-à-dire « imx6dlsabresd »

```
$ MACHINE=imx6dlsabresd DISTRO=fsl-imx-x11 source ./fsl-setup-release.sh -b build
```

DISTRO	LA DESCRIPTION
fsl-imx-fb	Linux Frame buffer - non x11 ou Wayland
fsl-imx-x11	Seulement x11 (X Window System)
fsl-imx-wayland	Wayland Weston graphics
fsl-imx-xwayland	Wayland graphics et x11. Les applications x11 utilisant EGL sont non supporté

3. Examinez le contenu du fichier « local.conf » en suivant la commande

```
$ pwd (assurez-vous d'être dans le répertoire « build »)
$ vi conf/local.conf
```

Le fichier « local.conf » contient les spécifications de la « machine » et de le « distro ». Apportez quelques modifications si vous le souhaitez, enregistrez et quittez l'éditeur « vi »

4. Copier le contenu sur le fichier « local.conf.org » en suivant la commande

```
$ cp conf/local.conf conf/ local.conf.org
```

Cette étape est obligatoire car le fichier «local.conf.org» remplace le contenu de «local.conf» chaque fois que vous essayez de créer une image

5. Entrez la commande suivante pour créer l'image

```
$ bitbake -v core-image-minimal
```

Nous pouvons trouver différents types de commande « bitbake » dans ce lien → Bitbake commands

LES REMARQUES IMPORTANTES

- Lorsque vous obtenez des erreurs lors de la construction de l'image, consultez le « NXP Forums » pour les solutions
- Assurez-vous que vous avez « make » version 4.1 ou supérieure dans la machine Linux que vous construisez l'image

7. Flash the built image in the SD card by following commands

```
$ cd imx-yocto-bsp/build/tmp/deploy/images/imx6dlsabresd
$ cat /proc/partitions (Pour savoir le chemin de la carte SD insérée)
$ sudo dd if=<image name>.sdcard of=/dev/sd<partition> bs=1M conv=fsync
```

Par exemple

```
$ sudo dd if=core-image-minimal-imx6dlsabresd.sdcard of=/dev/sdc bs=1M
conv=fsync
$ sync
$ sync
```

LE REMARQUE IMPORTANTE

Retirer la carte SD sans éjecter

Insérez la carte SD flashée dans la fente fournie et mettez le kit d'évaluation sous tension.

LES CONCEPTS SUPPLÉMENTAIRES POUR SAVOIR

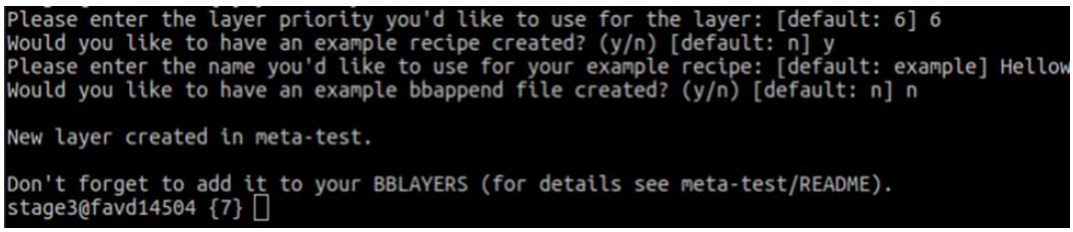
CRÉER ET AJOUTER UNE COUCHE DE YOCTO PERSONNALISÉE

Il est recommandé de créer une couche de yocto personnalisée avec des recettes personnalisées au lieu d'éditer des recettes existantes, sauf si elles sont absolument obligatoires

1. Créer une couche de « yocto » en utilisant les commandes suivantes

```
$ cd imx-yocto-bsp/

$ MACHINE=imx6dlsabresd DISTRO=fsl-imx-x11 source ./fsl-setup-release.sh -b
build

$ cd ..

$ cd sources/

$ yocto-layer create <Le nom de couche>
```

   Cela vous demandera quelques questions comme



2. Après cela, entrez les commandes suivantes pour ajouter une nouvelle couche à la construction d'image

```
$ cd imx-yocto-bsp/build/conf

$ vi bblayers.conf.org
```

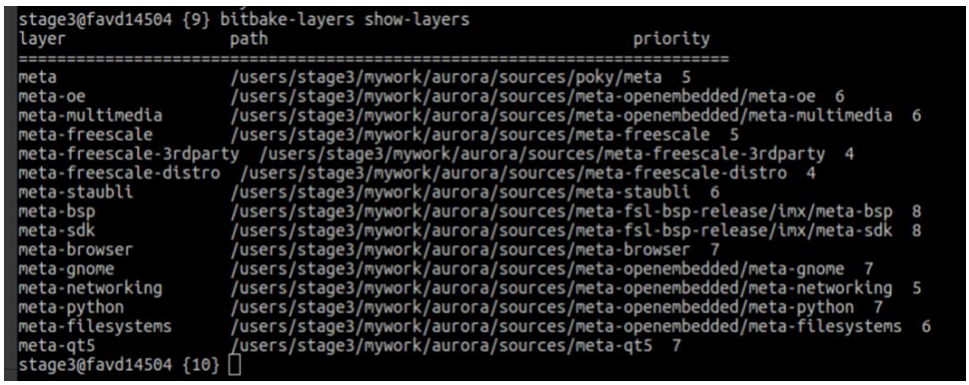
   ajouter le nouveau chemin de couche créé comme suit

```
LCONF_VERSION = "6"


BBPATH = "${TOPDIR}"

BSPDIR := "${@os.path.abspath(os.path.dirname(d.getVar('FILE', True)) +
'/../..')}"
```

```
BBFILES ?= ""

BBLAYERS = " \

  ${BSPDIR}/sources/poky/meta \

  ${BSPDIR}/sources/poky/meta-yocto \

  \

  ${BSPDIR}/sources/meta-openembedded/meta-oe \

  ${BSPDIR}/sources/meta-openembedded/meta-multimedia \

  \

  ${BSPDIR}/sources/meta-freescale \

  ${BSPDIR}/sources/meta-freescale-3rdparty \

  ${BSPDIR}/sources/meta-freescale-distro \

"

# added a custom yocto layer

BBLAYERS += "${BSPDIR}/sources/meta-<Le nom de couche>"
```

3. Ensuite, entrez les commandes suivantes pour vérifier si la nouvelle couche créé est ajoutée à la construction de l'image

```
$ cd imx-yocto-bsp/

$ MACHINE=imx6dlsabresd DISTRO=fsl-imx-x11 source ./fsl-setup-release.sh -b build

$ bitbake-layers show-layers
```



Cela montre toutes les couches qui sont utilisées pour construire l'image et leur priorité respective.

AJOUTER LES PAQUETS ESSENTIELS

1. Entrez les commandes suivantes

```
$ cd imx-yocto-bsp/sources/meta-<Le nom de couche>/recipes-<Le nom de couche>/
core-image-minimal/
```

```
$ touch core-image-minimal.bbappend
```

```
$ vi core-image-minimal.bbappend
```

2. Editez le fichier « core-image-minimal.bbappend »,

Par exemple, si vous souhaitez ajouter le support Qt5, Prenez le «fsl-image-qt5.bb», «fsl-image-qt5-validation-imx.bb» et «fsl-image-validation-imx.bb» comme référence pour ajouter le support Qt à notre image personnalisée. éditez comme suit

```
# Install Freescale QT demo applications

QT5_IMAGE_INSTALL_APPS = ""

QT5_IMAGE_INSTALL_APPS_imxgpu3d =
"${@bb.utils.contains("MACHINE_GSTREAMER_1_0_PLUGIN", "imx-gst1.0-plugin", "imx-
qtapplications", "", d)}"


# Install fonts

QT5_FONTS = "ttf-dejavu-common ttf-dejavu-sans ttf-dejavu-sans-mono ttf-dejavu-
serif"


# Install Freescale QT demo applications for X11 backend only

MACHINE_QT5_MULTIMEDIA_APPS = ""

QT5_IMAGE_INSTALL = ""

QT5_IMAGE_INSTALL_common = " \

     packagegroup-qt5-demos \

     packagegroup-qt5-toolchain-target \

    ${QT5_FONTS} \
```

```
    ${QT5_IMAGE_INSTALL_APPS} \
    ${@bb.utils.contains('DISTRO_FEATURES', 'x11', 'libxkbcommon', '', d)} \
    ${@bb.utils.contains('DISTRO_FEATURES', 'wayland', 'qtwayland qtwayland-plugins', '', d)}\
"
QT5_IMAGE_INSTALL_imxgpu2d = "${@bb.utils.contains('DISTRO_FEATURES', 'x11','${QT5_IMAGE_INSTALL_common}', \
    'qtbase qtbase-plugins', d)}"

QT5_IMAGE_INSTALL_imxpxp = "${@bb.utils.contains('DISTRO_FEATURES', 'x11','${QT5_IMAGE_INSTALL_common}', \
    'qtbase qtbase-examples qtbase-plugins', d)}"

QT5_IMAGE_INSTALL_imxgpu3d = " \
    ${QT5_IMAGE_INSTALL_common} \
    gstreamer1.0-plugins-bad-qt"

# Add packagegroup-qt5-webengine to QT5_IMAGE_INSTALL_mx6 and comment out the line below to install qtwebengine to the rootfs.
QT5_IMAGE_INSTALL_remove = " packagegroup-qt5-webengine"

IMAGE_INSTALL += " \
${QT5_IMAGE_INSTALL} \
"
## Select Image Features
IMAGE_FEATURES += " \
      debug-tweaks \
      tools-profile \
      nfs-server \
      tools-debug \
```

```
        hwcodecs \
        splash \
        tools-testapps \
        ${@bb.utils.contains('DISTRO_FEATURES', 'wayland', '', \
         bb.utils.contains('DISTRO_FEATURES',     'x11', 'x11-base x11-sato', \
                                                         '', d), d)} \
"


CORE_IMAGE_EXTRA_INSTALL += " \
        packagegroup-core-full-cmdline \
        packagegroup-fsl-tools-gpu \
        packagegroup-fsl-tools-testapps \
        packagegroup-fsl-tools-bluetooth \
        packagegroup-fsl-tools-audio \
        packagegroup-fsl-gstreamer1.0 \
        packagegroup-fsl-gstreamer1.0-full \
        packagegroup-fsl-tools-gpu-external \
${@bb.utils.contains('DISTRO_FEATURES', 'wayland', 'weston-init', '', d)} \
${@bb.utils.contains('DISTRO_FEATURES', 'x11 wayland', 'weston-xwayland xterm',
'', d)} \
"
```

3. Maintenant, reconstruisez l'image entière en suivant les commandes

```
$ cd imx-yocto-bsp/
$ MACHINE=imx6dlsabresd DISTRO=fsl-imx-x11 source ./fsl-setup-release.sh -b
build
$ bitbake -c cleanall core-image-minimal
$ bitbake -v core-image-minimal
```

Nous avons donc ajouté le support Qt5 à notre image personnalisée en éditant le fichier « core-image-minimal.bbappend » mais pas en éditant le fichier « core-image-minimal.bb »

C'est ainsi que vous pouvez ajouter des paquets essentiels à vos images personnalisées.

Il est toujours préférable de créer et éditer le fichier « .bbappend » mais pas le fichier « .bb » d'origine

ENLEVEZ LES PAQUETS INUTILES DE LA CONSTRUCTION D'IMAGE

Quand vous construisez l'image fournie par « Yocto », nous obtenons tant de paquets installés par défaut mais dans beaucoup d'applications, nous n'avons pas besoin de tous ces paquets. Ainsi dans ces cas, Nous pouvons enlever certains de paquets inutiles pour réduire la taille de « Rootfs ».

Il existe plusieurs façons de supprimer les paquets. elles sont

1. Utilisez la variable « `DISTRO_FEATURES_remove` » dans le fichier « local.conf » lors de la construction d'une image

```
$ cd imx-yocto-bsp/
$ MACHINE=imx6dlsabresd DISTRO=fsl-imx-x11 source ./fsl-setup-release.sh -b
build
$ vi conf/local.conf
```

Par exemple, ajouter cette lignes

```
DISTRO_FEATURES_remove = "\
      x11 \
      wayland \
      alsa \
      bluetooth \
      ext2 \
“
$ cp conf/local.conf conf/local.conf.org
$ bitbake -v core-image-minimal
```

2. Utilisez la variable « PACKAGE_EXCLUDE » or dans le fichier « local.conf » lors de la construction d'une image

```
$ cd imx-yocto-bsp/

$ MACHINE=imx6dlsabresd DISTRO=fsl-imx-x11 source ./fsl-setup-release.sh -b
build

$ vi conf/local.conf
```

Par exemple vous voulez supprimer le paquet « alsa-tools », Ajouter cette lignes

```
PACKAGE_EXCLUDE_pn-core-image-minimal += " \
      alsa-tools \
"
```

Ensuite, entrez les commandes suivantes

```
$ cd imx-yocto-bsp/sources/
$ grep -rnw 'imx-yocto-bsp/sources' -e 'alsa-tools'
```

Par la façon suivante, Nous viendrons pour savoir ce qui est les autres paquets ont besoin de ce paquet. Donc, Nous devons enlevez la dépendance inverse le paquet « alsa-tools » sur le paquet « packagegroup-fsl-tools-testapps »

```
$ cd imx-yocto-bsp/sources/meta-<Le nom de couche>/recipes-<Le nom de
couche>/packagegroup-fsl-tools-testapps
$ nano packagegroup-fsl-tools-testapps.bbappend
```

Editez-le comme suit

```
RDEPENDS_${PN}_remove = " \
alsa-tools \
"

$ cd imx-yocto-bsp
$ MACHINE=imx6dlsabresd DISTRO=fsl-imx-x11 source ./fsl-setup-release.sh -b build
$ bitbake -c cleanall core-image-minimal
$ bitbake core-image-minimal
```

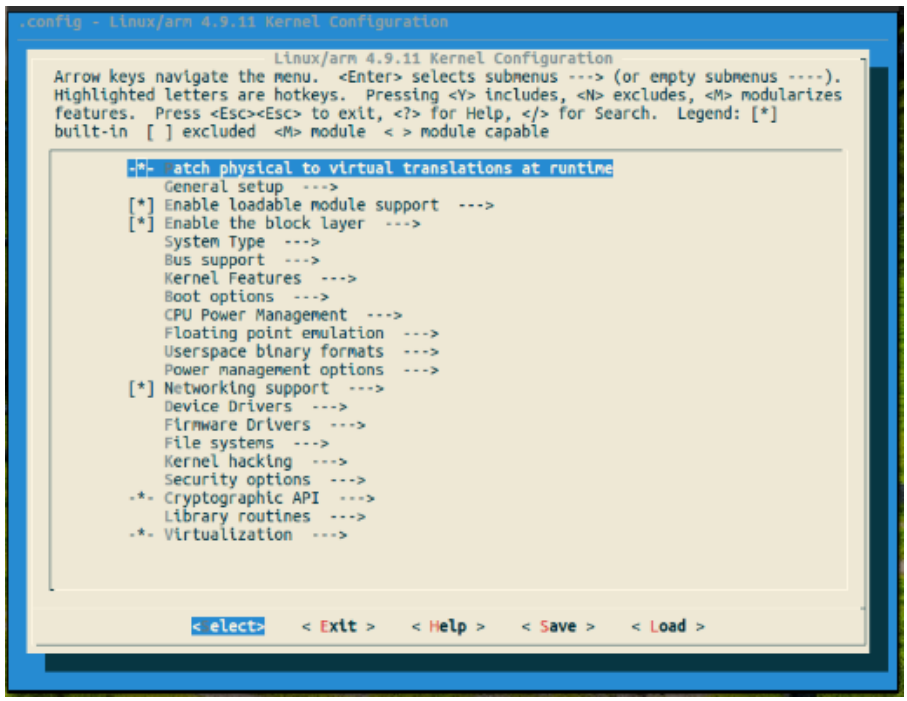
Par conséquent nous avons avec succès enlevé le paquet « alsa-tools »

OPTIMISER LE NOYAU

Entrez les commandes suivantes

```
$ cd imx-yocto-bsp
$ MACHINE=imx6dlsabresd DISTRO=fsl-imx-x11 source ./fsl-setup-release.sh -b build
$ bitbake -c menuconfig linux-imx
```

Supprimer les fonctionnalités inutiles et créer des pilotes de périphérique inutilisés et incertains comme « module »



Consulter le lien suivant pour plus d'informations → Building the Kernel

Utilisez les liens suivants pour optimiser le noyau

Bootlin - Boot time optimization

NXP - Reducing Boot Time:Techniques for Fast Booting

Yocto Project - Tuning Embedded Linux

Mentor Graphics - Achieving Faster Boot Time with Linux

La remarque Importante

Consulter le lien suivant pour un exemple → « .config »

AJOUTER NOTRE PAQUET PERSONNALISE

Nous pouvons ajouter nos paquets personnalisés en utilisant « `IMAGE_INSTALL_append` ». Par exemple, Prenez une application basée sur Qt5 que l'affichage « Hello World »

1. Tout d'abord, nous devons construire "meta-toolchain-qt5" pour créer et compiler une application Qt, Entrez les commandes suivantes

```
$ cd imx-yocto-bsp
$ MACHINE=imx6dlsabresd DISTRO=fsl-imx-x11 source ./fsl-setup-release.sh -b build
$ bitbake meta-toolchain-qt5
```

2. Après que le toolchain est construit, Entrez les commandes suivantes

```
$ cd imx-yocto-bsp/build/tmp/deploy/sdk
$ ./fsl-imx-x11-glibc-x86_64-meta-toolchain-qt5-cortexa9hf-neon-toolchain-4.9.11-1.0.0.sh
```

   Entrez le chemin suivant pour installer /imx-yocto-bsp/tools/meta-toolchain-qt5/ et et continuer l'installation. Nous avons donc construit et installé le « meta-toolchain-qt5 ».

3. L'étape suivante consiste à télécharger et installer le qtcreator du lien suivant → Qtcreator

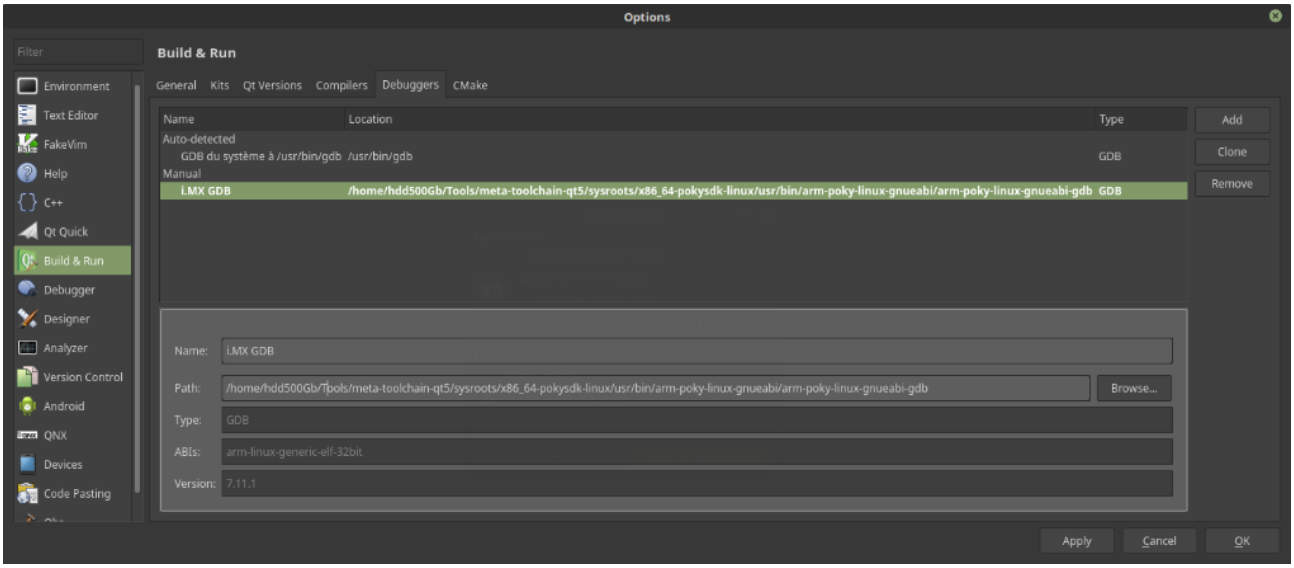
   La remarque importante

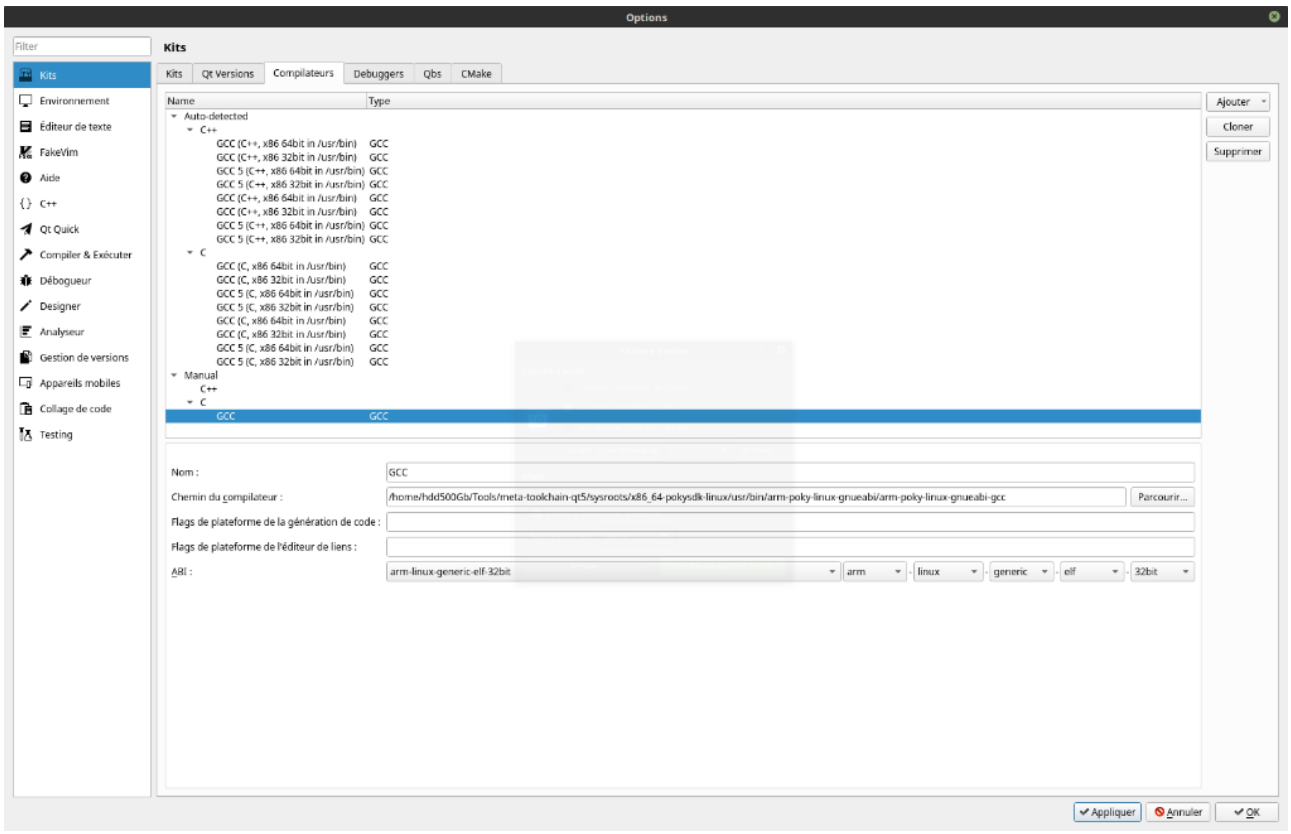
   - ✓ Spécifiez le chemin pour l'installation de qt en tant que « /imx-yocto-bsp/Qt/ »
   - ✓ Veillez à sélectionner «QT 5,8» dans l'installation dans la liste « Select components » et terminez l'installation.

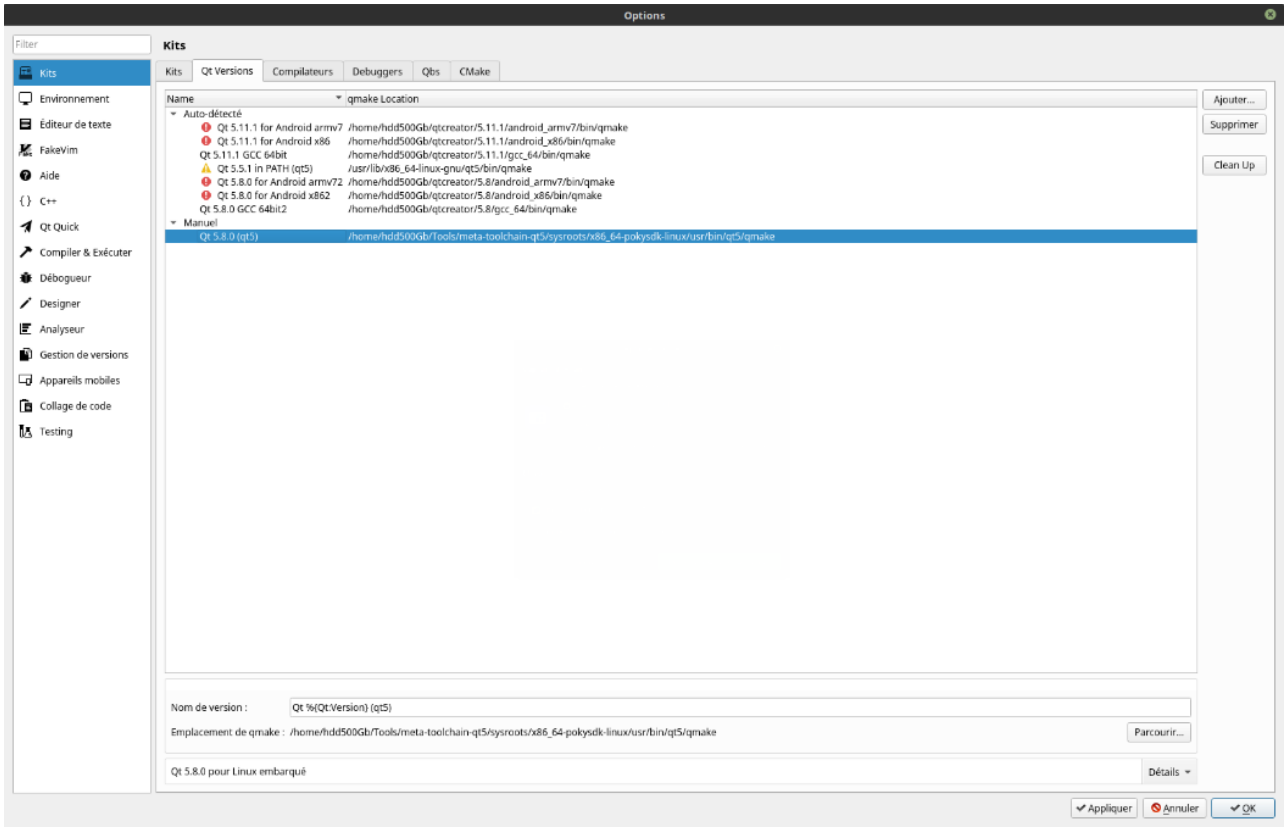
   Consultantez le lien suivante → Installation de Qtcreator

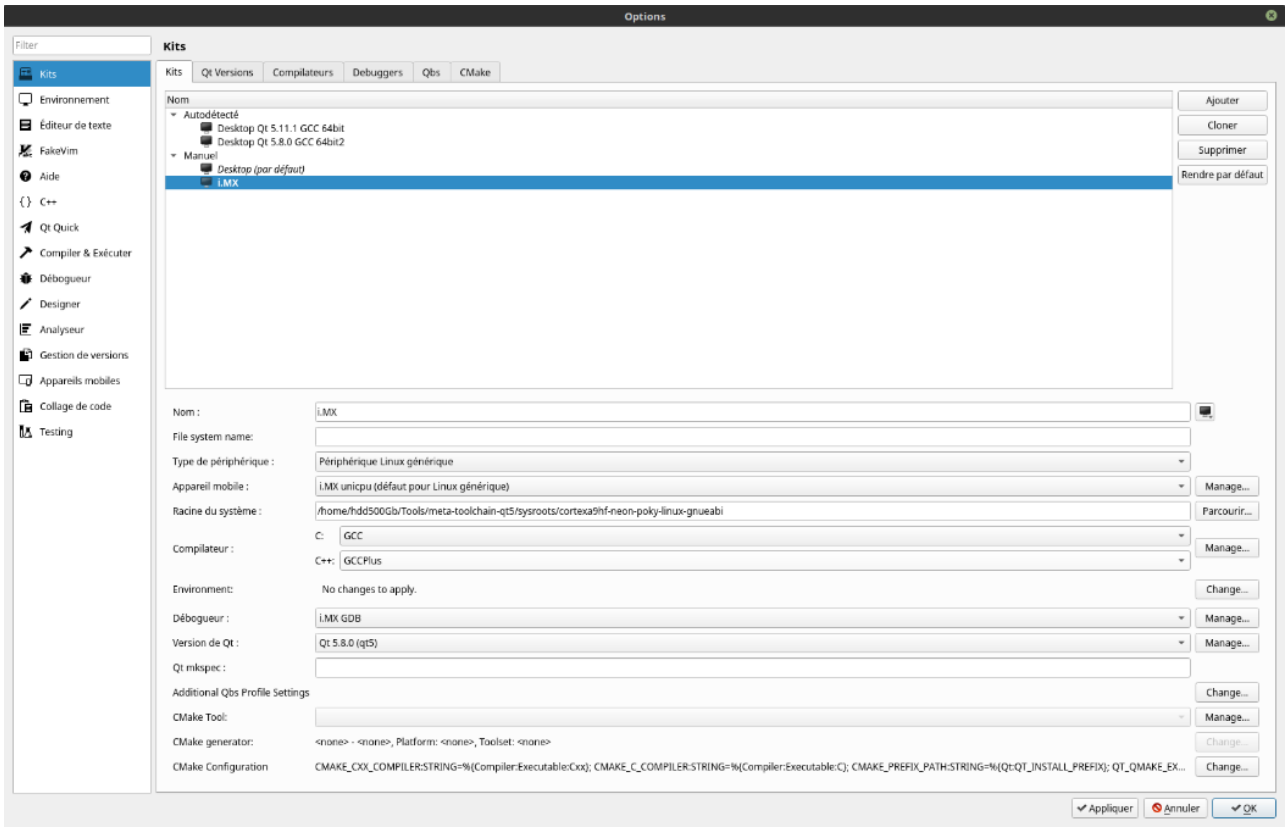
4. Ensuite configurez le Qtcreator, Entrez les commandes suivantes

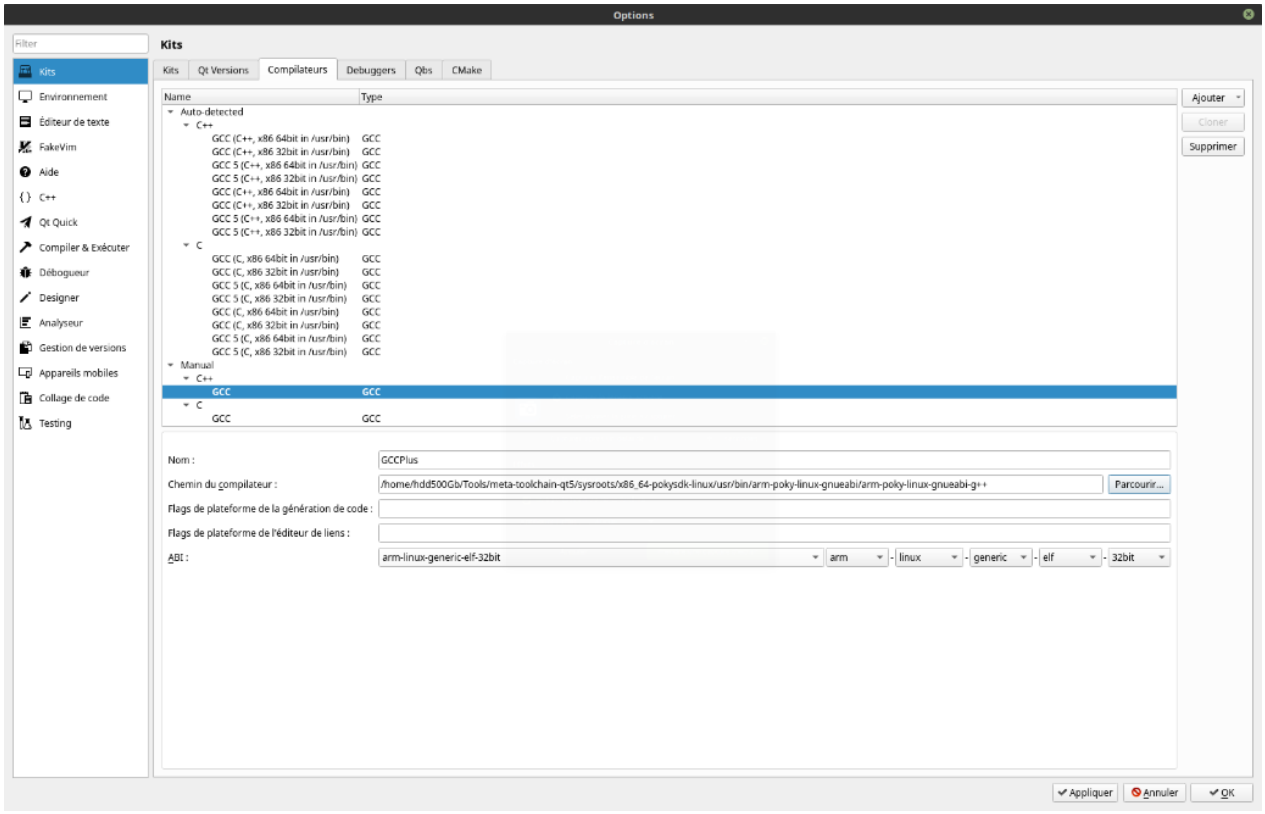
```
$ source imx-yocto-bsp/tools/meta-toolchain-qt5/environment-setup-cortexa9hf-neon-poky-linux-gnueabi
$ cd /imx-yocto-bsp/Qt/qtcreator/Tools/QtCreator/bin
$ ./qtcreator.sh
```


Options
Filter
Build & Run
Environment
Text Editor
FakeVim
Help
C++
Qt Quick
Build & Run
Debugger
Designer
Analyzer
Version Control
Android
QNX
Devices
Code Pasting
General
Kits
Qt Versions
Compilers
Debuggers
CMake
Name
Location
Type
Add
Clone
Remove
Auto-detected
GDB du système à /usr/bin/gdb
/usr/bin/gdb
GDB
Manual
i.MX GDB
/home/hdd500Gb/Tools/meta-toolchain-qt5/sysroots/x86_64-pokysdk-linux/usr/bin/arm-poky-linux-gnueabi/arm-poky-linux-gnueabi-gdb
GDB
Name:
i.MX GDB
Path:
/home/hdd500Gb/Tools/meta-toolchain-qt5/sysroots/x86_64-pokysdk-linux/usr/bin/arm-poky-linux-gnueabi/arm-poky-linux-gnueabi-gdb
Browse...
Type:
GDB
ABIs:
arm-linux-generic-elf-32bit
Version:
7.11.1
Apply
Cancel
OK


Options
Filter
Kits
Environnement
Éditeur de texte
FakeVim
Aide
C++
Qt Quick
Compiler & Exécuter
Débogueur
Designer
Analyseur
Gestion de versions
Appareils mobiles
Collage de code
Testing
Kits
Kits
Qt Versions
Compilateurs
Debuggers
Qbs
CMake
Name
Type
Ajouter
Cloner
Supprimer
Auto-detected
C++
GCC (C++, x86 64bit in /usr/bin) GCC
GCC (C++, x86 32bit in /usr/bin) GCC
GCC 5 (C++, x86 64bit in /usr/bin) GCC
GCC 5 (C++, x86 32bit in /usr/bin) GCC
GCC (C++, x86 64bit in /usr/bin) GCC
GCC (C++, x86 32bit in /usr/bin) GCC
GCC 5 (C++, x86 64bit in /usr/bin) GCC
GCC 5 (C++, x86 32bit in /usr/bin) GCC
C
GCC (C, x86 64bit in /usr/bin) GCC
GCC (C, x86 32bit in /usr/bin) GCC
GCC 5 (C, x86 64bit in /usr/bin) GCC
GCC 5 (C, x86 32bit in /usr/bin) GCC
GCC (C, x86 64bit in /usr/bin) GCC
GCC (C, x86 32bit in /usr/bin) GCC
GCC 5 (C, x86 64bit in /usr/bin) GCC
GCC 5 (C, x86 32bit in /usr/bin) GCC
Manual
C++
C
GCC
GCC
Nom :
GCC
Chemin du compilateur :
/home/hdd500Gb/Tools/meta-toolchain-qt5/sysroots/x86_64-pokysdk-linux/usr/bin/arm-poky-linux-gnueabi/arm-poky-linux-gnueabi-gcc
Parcourir...
Flags de plateforme de la génération de code :
Flags de plateforme de l'éditeur de liens :
ABI :
arm-linux-generic-elf-32bit
arm
linux
generic
elf
32bit
Appliquer
Annuler
OK


Options
Filter
Kits
Environnement
Éditeur de texte
FakeVim
Aide
C++
Qt Quick
Compiler & Exécuter
Débogueur
Designer
Analyseur
Gestion de versions
Appareils mobiles
Collage de code
Testing
Kits
Kits
Qt Versions
Compilateurs
Debuggers
Qbs
CMake
Name
qmake Location
Auto-détecté
Qt 5.11.1 for Android armv7 /home/hdd500Gb/qtcreator/5.11.1/android_armv7/bin/qmake
Qt 5.11.1 for Android x86 /home/hdd500Gb/qtcreator/5.11.1/android_x86/bin/qmake
Qt 5.11.1 GCC 64bit /home/hdd500Gb/qtcreator/5.11.1/gcc_64/bin/qmake
Qt 5.5.1 in PATH (qt5) /usr/lib/x86_64-linux-gnu/qt5/bin/qmake
Qt 5.8.0 for Android armv72 /home/hdd500Gb/qtcreator/5.8/android_armv7/bin/qmake
Qt 5.8.0 for Android x862 /home/hdd500Gb/qtcreator/5.8/android_x86/bin/qmake
Qt 5.8.0 GCC 64bit2 /home/hdd500Gb/qtcreator/5.8/gcc_64/bin/qmake
Manuel
Qt 5.8.0 (qt5) /home/hdd500Gb/Tools/meta-toolchain-qt5/sysroots/x86_64-pokysdk-linux/usr/bin/qt5/qmake
Ajouter...
Supprimer
Clean Up
Nom de version : Qt %{Qt:Version} (qt5)
Emplacement de qmake : /home/hdd500Gb/Tools/meta-toolchain-qt5/sysroots/x86_64-pokysdk-linux/usr/bin/qt5/qmake
Parcourir...
Qt 5.8.0 pour Linux embarqué
Détails
Appliquer
Annuler
OK


Options
Filter
Kits
Environnement
Éditeur de texte
FakeVim
Aide
C++
Qt Quick
Compiler & Exécuter
Débogueur
Designer
Analyseur
Gestion de versions
Appareils mobiles
Collage de code
Testing
Kits
Kits
Qt Versions
Compilateurs
Debuggers
Qbs
CMake
Nom
Autodétecté
Desktop Qt 5.11.1 GCC 64bit
Desktop Qt 5.8.0 GCC 64bit2
Manuel
Desktop (par défaut)
i.MX
Ajouter
Cloner
Supprimer
Rendre par défaut
Nom : i.MX
File system name:
Type de périphérique : Périphérique Linux générique
Appareil mobile : i.MX unicpu (défaut pour Linux générique)
Manage...
Racine du système : /home/hdd500Gb/Tools/meta-toolchain-qt5/sysroots/cortexa9hf-neon-poky-linux-gnueabi
Parcourir...
Compilateur : C: GCC
C++: GCCPlus
Manage...
Environment: No changes to apply.
Change...
Débogueur : i.MX GDB
Manage...
Version de Qt : Qt 5.8.0 (qt5)
Manage...
Qt mkspec :
Additional Qbs Profile Settings
Change...
CMake Tool:
Manage...
CMake generator: <none> - <none>, Platform: <none>, Toolset: <none>
Change...
CMake Configuration CMAKE_CXX_COMPILER:STRING=%{Compiler:Executable:Cxx}; CMAKE_C_COMPILER:STRING=%{Compiler:Executable:C}; CMAKE_PREFIX_PATH:STRING=%{Qt:QT_INSTALL_PREFIX}; QT_QMAKE_EX...
Change...
Appliquer
Annuler
OK



Ensuite, Editez le fichier « qmake.conf » qui est dans le chemin « imx-yocto-bsp/tools/meta-toolchain-qt5/sysroots/cortexa7hf-neon-poky-linux-gnueabi/usr/lib/qt5/mkspecs/linux-arm-gnueabi-g++/qmake.conf comme (Spécifiez votre chemin personnalisé si vous voulez)

```
#
# qmake configuration for building with arm-linux-gnueabi-g++
#

#MAKEFILE_GENERATOR      = UNIX
#CONFIG                 += incremental
#QMAKE_INCREMENTAL_STYLE = sublib

include(../common/linux.conf)
include(../common/gcc-base-unix.conf)
include(../common/g++-unix.conf)

# modifications to g++.conf

QMAKE_CC                = arm-poky-linux-gnueabi-gcc
QMAKE_CXX               = arm-poky-linux-gnueabi-g++
QMAKE_LINK              = arm-poky-linux-gnueabi-g++
QMAKE_LINK_SHLIB        = arm-poky-linux-gnueabi-g++
QMAKE_LFLAGS += --sysroot=/home/hdd500Gb/Tools/meta-toolchain-qt5/sysroots/cortexa7hf-neon-poky-linux-gnueabi -mfloat-abi=hard -mfpu=neon-vfpv4
QMAKE_CXXFLAGS +=  -mfloat-abi=hard -mfpu=neon-vfpv4

# modifications to linux.conf
QMAKE_AR                = arm-poky-linux-gnueabi-ar cqs
QMAKE_OBJCOPY           = arm-poky-linux-gnueabi-objcopy
QMAKE_NM                = arm-poky-linux-gnueabi-nm -P
QMAKE_STRIP             = arm-poky-linux-gnueabi-strip
load(qt_config)
```

Appliquez toutes les modifications et appuyez sur ok. Vous êtes prêt à utiliser « Qtcreator »

5. Dans le « Qtcreator », Consultez le lien suivante → Hello World
6. Copiez les fichiers « helloworld.cpp » et « helloworld.pro » créés par Qtcreator sur votre couche yocto sur le chemin suivant

« imx-yocto-bsp/sources/meta-<Le nom de couche>/recipes-<Le nom de couche>/ helloworldqt5/files »

7. Editez le fichier « helloworld.cpp » comme

```
#include <QtWidgets/QApplication>
#include <QtWidgets/QPushButton>

int main( int argc, char **argv ) {
  QApplication a( argc, argv );

  QPushButton hello( "Hello world!", 0 );
  hello.resize( 100, 30 );

  hello.show();
  return a.exec();
}
```

8. Editez le fichier « helloworld.pro » comme

```
QT += core gui
QT += widgets
QT += quick
CONFIG += c++11
# The following define makes your compiler emit warnings if you use
# any feature of Qt which as been marked deprecated (the exact warnings
# depend on your compiler). Please consult the documentation of the
# deprecated API in order to know how to port your code away from it.
DEFINES += QT_DEPRECATED_WARNINGS

# You can also make your code fail to compile if you use deprecated APIs.
# In order to do so, uncomment the following line.
# You can also select to disable deprecated APIs only up to a certain version of
Qt.
#DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000    # disables all the APIs
deprecated before Qt 6.0.0

SOURCES += \
    helloworldqt5.cpp
```

```
#RESOURCES +=

# Default rules for deployment.
#qnx: target.path = /tmp/$${TARGET}/bin
#else: unix:!android: target.path = /opt/$${TARGET}/bin
#!isEmpty(target.path): INSTALLS += target
```

9. Ensuite, Entrez les commandes suivantes

```
$ cd imx-yocto-bsp/sources/meta-<Le nom de couche>/recipes-<Le nom de couche>/
helloworldqt5
$ nano helloworldqt5_0.1.bb
```

Editez le fichier comme

```
#
# This file was derived from the 'Hello World!' example recipe in the
# Yocto Project Development Manual.
#

SUMMARY = "Simple helloworld application"
SECTION = "examples"
LICENSE = "MIT"
LIC_FILES_CHKSUM                                                    =
"file://${COMMON_LICENSE_DIR}/MIT;md5=0835ade698e0bcf8506ecda2f7b4f302"



SRC_URI = "file://helloworldqt5.pro \
           file://helloworldqt5.cpp \
"
DEPENDS +="qtbase fontconfig"

S = "${WORKDIR}"

inherit qmake5

do_install() {
           install -d ${D}${bindir}
           install -m 0755 helloworldqt5 ${D}${bindir}
}
```

Consultez le lien suivante → Hello World Recipe

10. Entrez les commandes suivantes

```
$ cd imx-yocto-bsp
$ MACHINE=imx6dlsabresd DISTRO=fsl-imx-x11 source ./fsl-setup-release.sh -b build
$ vi conf/local.conf
```

Ajouter la ligne suivante

```
IMAGE_INSTALL_append = "helloworldqt5"
```

```
$ cp conf/local.conf conf/local.conf.org
$ bitbake -c cleanall core-image-minimal
$ bitbake -v core-image-minimal
```

Ainsi nous avons avec succès ajouté votre paquet personnalisé.

GOCONCEPT
a step ahead

LES REFERENCES IMPORTANTES

Consultez les liens suivants pour plus d'informations

- ✓ Yocto Project Mega-Manual
- ✓ i.MX Yocto Project User's Guide
- ✓ NXP Community
- ✓ OpenEmbeddedMeta Layers Yocto
- ✓ i.MX Linux® User's Guide
- ✓ Creating a custom distribution image with Yocto