INTRODUCTION

The Yocto Project is an open source collaboration project that helps developers create custom linux-based systems for embedded products, regardless of the hardware architecture. For more information, check this link → yoctoproject

REQUIREMENTS OF THE HOST SYSTEM

1. It is recommended that at least provide 120Gb memory, which is enough to compile any backend.
2. This are the supported Linux Distributions
   - ✔ Ubuntu 14.10
   - ✔ Ubuntu 15.04
   - ✔ Ubuntu 15.10
   - ✔ Ubuntu 16.04 (LTS)
   - ✔ Fedora release 22
   - ✔ Fedora release 23
   - ✔ CentOS release 7.x
   - ✔ Debian GNU/Linux 8.x (Jessie)
   - ✔ Debian GNU/Linux 9.x (Stretch)
   - ✔ openSUSE 13.2
   - ✔ openSUSE 42.1

   Note

   The "Linux mint" which we are using at «STAUBLI» is not officially supported for Yocto but it works good as well.
3. The below are the essential Yocto Project host packages you need to install before you start building the images

```
$ sudo apt-get install gawk wget git-core diffstat unzip texinfo gcc-multilib build-essential chrpath socat
```

i.MX layers host packages for a Ubuntu 12.04 or 14.04 host setup

```
$ sudo apt-get install 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
```

i.MX layers host packages for a Ubuntu 12.04 host setup only

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

i.MX layers host packages for a Ubuntu 14.04 host setup only

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

SETTING-UP THE REPO UTILITY

Repo is a tool built on top of Git that makes it easier to manage projects that contain multiple repositories, which do not need to be on the same server. Repo complements very well the layered nature of the Yocto Project, making it easier for customers to add their own layers to the BSP.

To install the “repo” utility, perform these steps

1. Create a bin folder in your home directory

```
$ 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. Add the following line to the .bashrc file to ensure that the ~/bin folder is in your PATH variable

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

YOCTO PROJECT SETUP AND IMAGE BUILDING

1. After setting the repo utility, Create a directory named 'aurora' in the directory /users/stage3/mywork

```
$ mkdir aurora
$ cd aurora
$ repo init -u https://source.codeaurora.org/external/imx/fsl-arm-yocto-bsp -b imx-morty
$ repo sync
```

Note

If you get an error during this process, install libcur14-openssl-dev:i386 package

2. Setup the build folder by entering the following command

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

because we are building for imx6 dual-lite sabre SD board

Note

We need to be in /mywork/aurora directory each time we enter the above command and It will direct you automatically to the /mywork/aurora/build directory.

3. Edit the aurora/build/conf/local.conf file as follows which place the downloads, sstate-cache and tmp folders in the different location. This is done because this folders take lot of memory like 120 to 150 GB depending on the image you build.

```
MACHINE ??= 'imx6dlsabresd'
DISTRO ?= 'fsl-imx-x11'
PACKAGE_CLASSES ?= "package_rpm"
EXTRA_IMAGE_FEATURES ?= "debug-tweaks"
USER_CLASSES ?= "buildstats image-mklibs image-prelink"
PATCHRESOLVE = "noop"
BB_DISKMON_DIRS = "\
    STOPTASKS,${TMPDIR},1G,100K \
    STOPTASKS,${DL_DIR},1G,100K \
    STOPTASKS,${SSTATE_DIR},1G,100K \
    STOPTASKS,/tmp,100M,100K \
    ABORT,${TMPDIR},100M,1K \
    ABORT,${DL_DIR},100M,1K \
    ABORT,${SSTATE_DIR},100M,1K \
    ABORT,/tmp,10M,1K"
PACKAGECONFIG_append_pn-qemu-native = " sdl"
PACKAGECONFIG_append_pn-nativesdk-qemu = " sdl"
CONF_VERSION = "1"

# It is necessary to place in the /home/hdd500Gb to save the memory available on our users/stage3

DL_DIR = "/home/hdd500Gb/aurora/build/downloads/"
SSTATE_DIR = "/home/hdd500Gb/aurora/build/sstate-cache/"
TMPDIR = "/home/hdd500Gb/aurora/build/tmp/"

ACCEPT_FSL_EULA = "1"
```

4. Copy the local.conf to the local.conf.org

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

   This step is mandatory because the «local.conf.org» file replaces the «local.conf» contents each time you try to build an image

5. By default boot partition is 8MB but we can also increase it to 50MB for example. To do so, We need to edit the file in the following directory as follows mywork/aurora/sources/meta-freescale/conf/machine/imx6dlsabresd.conf

```
#@TYPE: Machine
#@NAME: NXP i.MX6DL SABRE Smart Device
#@SOC: i.MX6DL
#@DESCRIPTION: Machine configuration for NXP i.MX6DL SABRE Smart Device
#@MAINTAINER: Otavio Salvador <otavio@ossystems.com.br>

MACHINEOVERRIDES =. "mx6:mx6dl:"
require conf/machine/include/imx6sabresd-common.inc

KERNEL_DEVICETREE = " \
      imx6dl-sabresd.dtb \
      imx6dl-sabresd-ldo.dtb \
      imx6dl-sabresd-hdcp.dtb \
      imx6dl-sabresd-enetirq.dtb \
      imx6dl-sabresd-btwifi.dtb \
"

UBOOT_CONFIG ??= "sd"
UBOOT_CONFIG[sd] = "mx6dlsabresd_config,sdcard"
UBOOT_CONFIG[epdc] = "mx6dlsabresd_epdc_config"
UBOOT_CONFIG[mfgtool] = "mx6dlsabresd_config"
MACHINE_FIRMWARE += "firmware-imx-epdc"

# This parameter is used to set thz size of boot partition in KiloBytes
BOOT_SPACE += "51200"
```

6. After this we need to bitbake the image by following command

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

   We can find various types of bitbake command in this link → Bitbake commands

Notes

- Each time you want to change the «Boot partition» size, You have to enter the command below before bitbaking the image again which cleans the existing build

  ```
  $ bitbake -c cleanall core-image-minimal
  ```

- When you get errors during build, Check NXP forums for the solutions
- Make sure that you have "make" version 4.1 or above in the linux machine you are building

7. After the end of the building, We can find the images in the below directory

   /home/hdd500Gb/aurora/build/tmp/deploy/images/imx6dlsabresd

FLASHING THE BUILT IMAGE INTO THE SDCARD

1. Open the terminal in the directory /home/hdd500Gb/aurora/tmp/deploy/images/imx6dlsabresd
2. To know the address of the inserted sdcard, enter following command

   ```
   $ cat /proc/partitions
   ```

   You have inserted SDcard in /dev/sdc

   Note

   Please be careful at the inserted sdcard directory i.e. of=/dev/sdc

3. We can flash the built image into the sdcard by following command

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

   Enter the your password and it will take 5-7 minutes

   ```
   $ sync
   $ sync
   ```

   Note

   Afterwards takeout the sdcard (No need to eject it) and insert it into slot provided in the i.Mx6dlsabresd eval board and Power it up.

ADAPTING THE BUILT IMAGE TO THE UNICPU

In order to adapt to SDcard to your unicpu → «Customized STAUBLI board»

- ✔ Firstly, you have to flash the image created by yocto into SDcard
- ✔ You have to create our customized kernel image (uImage) and modules
- ✔ You have to create the «boot_unicpu.scr»
- ✔ You have to create and flash the «u-boot-with-spl.imx»

BUILDING THE KERNEL FROM STAUBLI SOURCE CODE

1. Download the source code in the directory /users/stage3/mywork by following command

   ```
   $ git clone git://fav10lin.staubli.pri/linux-2.6-imx.git
   ```

2. Change the cloned directory name to "linux-imx" and assign the remote name as follows

   ```
   $ git remote add aurora https://source.codeaurora.org/external/imx/linux-imx
   ```

3. Fetch and checkout to lastest aurora branch with following command

   ```
   $ git fetch aurora
   $ git branch -a
   $ git checkout aurora/imx_4.9.11_1.0.0_ga
   ```

4. The next task is to build the cross compiler tools, To do it first of all create an empty directory named as "Tools" in the following directory

   /home/hdd500Gb/Tools/meta-toolchain

5. Open the terminal in /users/stage3/mywork/aurora director and enter following commands

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

6. The above procedure will produce a directory "sdk" in the

   /home/hdd500Gb/aurora/build/tmp/deploy/sdk

7. Open the sdk directory and click on «fsl-imx-x11-glibc-x86_64-meta-toolchain-cortexa9hf-neon-toolchain-4.9.11-1.0.0.sh»

8. This leads to a terminal asking you a path to install the compiler tools give the path as «/home/hdd500Gb/Tools/meta-toolchain» and continue the installation. This procedure produces a toolchain which can be used to build kernel.

9. Now, You can start building the kernel. Firstly set the environment variables by enter the following commands

```
$ cd /users/stage3/mywork/linux-imx
$ source /home/hdd500Gb/Tools/environment-setup-cortexa9hf-neon-poky-linux-gnueabi
```

this will automatically set all the required environment variables

Note

This above step should be repeated when ever you want to perform some changes to kernel

10. We have to enable the touch on UNICPU, The procedure for that is as follows

- Acquire and add the "egalax_i2c.c" file to directory /users/stage3/mywork/linux-imx/drivers/input/touchscreen
- Add the following lines to the "Kconfig" file in the directory /users/stage3/mywork /linux-imx/drivers/input/touchscreen

```
# Added Egalax i2c driver
   config TOUCHSCREEN_EGALAX_I2C
    tristate "EETI eGalax i2c multi-touch panel support"
    depends on I2C && OF
    help
    Say Y here to enable support for I2C connected EETI
    eGalax multi-touch panels.
    To compile this driver as a module, choose M here: the
    module will be called egalax_i2c.
```

- Add the following line to the "Makefile" file in the following directory /home/hdd500Gb/linux-imx/drivers/input/touchscreen

```
#Added Egalax i2c Driver
obj-$(CONFIG_TOUCHSCREEN_EGALAX_I2C)+= egalax_i2c.o
```

11. Now you can configure the kernel by using menuconfig by which you can choose what are the configurations needed to build your uImage

```
$ make menuconfig
```

Note

- sometimes if you don't find the added driver option, use the below command instead of above command

```
$ make ARCH=arm menuconfig
```

- Before entering the above command please make sure to install the libncurses5-dev and libncursesw5-dev packages in our host system by following command

```
$ sudo apt-get install libncurses5-dev libncursesw5-dev
```

12. In the menuconfig window,

    Press «m» for

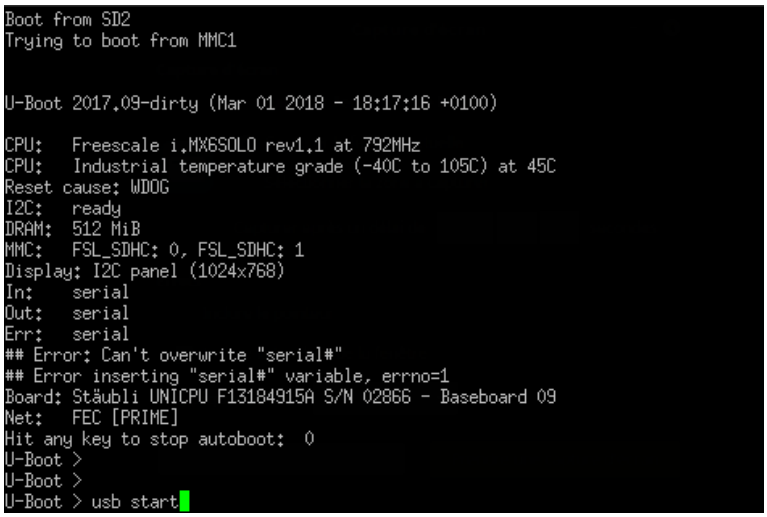
    Device Drivers → Input device support → Touchscreens → EETI eGalax i2c multi-touch panel support

    This enables the touch screen drivers but as a «Module»

    Press «y» for

    Device Drivers → HID support → Generic HID driver

    This enables the USB mouse and you may need to type «usb start» command in uboot after starting the kernel



13. To build the kernel, Enter the command below

```
$ make uImage LOADADDR=0x10008000
```

    this might take some time to build and after finishing, the output is located in the following directory /users/stage3/mywork/linux-imx/arch/arm/boot

14. The above step results only uImage file but to build the kernel modules, use following command

```
$ make modules
```

it will take some time to build, after that install them by using following command

```
$ make modules_install INSTALL_MOD_PATH=/users/stage3/mywork/Installed_modules
```

15. In order to build the .dtb file (Device tree)

```
$ cd /users/stage3/mywork/linux-imx
$ source /home/hdd500Gb/Tools/environment-setup-cortexa9hf-neon-poky-linux-gnueabi
```

Note

- Please make sure that device-tree-complier package is installed in your linux machine before running the below commands.
- To enable the Touch, make sure to add the following lines to linux-imx/arch/arm/boot/dts/imx6qdl-unicpu-faverges.dtsi

```
&i2c1 {
    clock-frequency = <400000>;
    pinctrl-names = "default";
    pinctrl-0 = <&pinctrl_i2c1>;
    status = "okay";
    egalax_i2c@2a {
        compatible = "eeti,egalax_i2c";
        reg = <0x2a>;
        interrupt-parent = <&gpio6>;
        interrupts = <8 8>;
        int-gpios = <&gpio6 8 0>;
     };
};
```

- To enable Ethernet, make sure to add the following block into the linux-imx/arch/arm/boot/dts/imx6qdl-unicpu.dtsi/

```
&clks {
    assigned-clocks = <&clks IMX6QDL_CLK_ENET_REF>;
    assigned-clock-rates = <50000000>;
};
&fec {
    pinctrl-names = "default";
    pinctrl-0 = <&pinctrl_enet>;
//  phy-connection-type = "rmii";
    phy-mode = "rmii";
    phy-reset-gpios = <&gpio7 7 0>;
    phy-reset-duration = <2>;
//   clocks = <&clks IMX6QDL_CLK_ENET>, <&clks IMX6QDL_CLK_ENET>, <&rmii_clk>;
    status = "okay";
};
```

Enter the following command to generate the dtb file

```
$ make imx6dl-unicpu-fav1024768.dtb
```

16. Remove the existing zImage, copy the created uImage and dtb file in the «Boot partition» and created modules to «Rootfs» partition of the SDcard

```
uImage from /users/stage3/mywork/linux-imx/arch/arm/boot

dtb files from /users/stage3/mywork/linux-imx/arch/arm/boot/dts

Modules from /users/stage3/mywork/Installed_modules
```

17. Remove the existing modules folder and copy the created modules in the Rootfs partition by following commands

```
$ cd /<Rootfs partition>/lib/modules

$ sudo rm -rf 4.9.11-1.0.0+gc27010d (This folder name is dynamic)

$ sudo cp -R /users/stage3/mywork/Installed_modules/lib/modules/4.9.11-02205-
gc27010d-dirty/ /media/stage3/<Rootfs Partition>/lib/modules/
```

18. Create a text file «boot_unicpu_fdt_sd_nodhcp»

```
# U-boot script sur carte SD - avec dtb
#
# Boot a Linux kernel from the SD card. The scipt and the uImage are located
# in the first partition of the SD card. It is a DOS partition.
# Screens: 800x600 FAV-LDB-SVGA, 1024x768 SAR-LDB-XVGA

setenv loadaddr 10800000
setenv dtaddr 12000000
setenv mmcdev 0

# Root pour le noyau: mmcblk1p2 pour SD
setenv rootdev /dev/mmcblk1p2

if test $baseboard_id -eq 20 ; then
  setenv dtfile imx6dl-unicpu-sar800600.dtb
else
  if test $x -eq 800 ; then
    setenv dtfile imx6dl-unicpu-fav800600.dtb
  else
    setenv dtfile imx6dl-unicpu-fav1024768.dtb
  fi
fi

# Argument du noyau
setenv bootargs "console=ttymxc0,115200 root=$rootdev rootwait=1"
```

```
if load mmc $mmcdev:1 $loadaddr uImage ; then
   if load mmc $mmcdev:1 $dtaddr $dtfile ; then
       bootm $loadaddr - $dtaddr
   fi
fi
```

19. Build «boot_unicpu.scr» file by following command

```
$ mkimage -T script -A ARM -d boot_unicpu_fdt_sd_nodhcp boot_unicpu.scr
```

20. Copy the created boot_unicpu.scr file to the «Boot partition» of SDcard

21. Acquire and Flash the "u-boot-with-spl.imx" file by using following command

```
$ sudo dd if=u-boot-with-spl.imx of=/dev/sdc bs=1k seek=1 conv=fsync
$ sync
$ sync
```

Note

Remove the SDcard without ejecting it

There you go your SDcard is ready which can boot-up the «UNICPU»

22. For enabling the Touch, enter the following commands after the login prompt which loads the egalax driver manually

```
$ root
$ cd /lib/modules/4.9.11-03209-gc27010d-dirty/kernel/drivers/input/touchscreen
$ insmod egalax_i2c.ko
```

this will install the egalax i2c driver.

23. You can verify the ethernet is working by using the following command

```
$ ifconfig
```

Note

We have enabled the connman.service in the yocto recipe itself but in case if it doesn’t start, Enter the following commands

```
$ systemctl unmask connman.service
$ systemctl enable connman.service
$ systemctl start connman.service
$ ifconfig
```

These are essential steps to adapt the SDcard to our UNICPU board.

TRANSFERING FILES BETWEEN HOST AND TARGET

We can also transfer files between host and target by using SSH

1. After enabling the Ethernet, Use the following command to get into connected UNICPU board through ethernet

   ```
   $ ssh root@imx6dlsabresd
   ```

   After,Press "yes" so that the unicpu board is permanently added to your list of known hosts.

2. Use the following command to see the available folders in the UNICPU board

   ```
   $ ls /dev
   ```

3. Open another terminal and use this following command to copy any files from your PC to host

   ```
   $ scp  <source path> <host user name ex:root@imx6dlsabresd>: <Destination path>
   ```

   Example

   ```
   $ scp /pdf/permanent/composants/freescale/IMX6/ root@imx6dlsabresd:/dev/null
   ```

4. Use the following command to copy from host to your PC

   ```
   $ scp <host user name ex:root@imx6dlsabresd>:<source path> <Destination path>
   ```

CREATE AND ADD A CUSTOMIZED YOCTO LAYER

It is recommended to create a customized yocto layer with customized recipes (.bb and .bbapend) instead of editing existing recipes.

1. Create a yocto layer using following commands

   ```
   $ cd /users/stage3/mywork/aurora
   $ MACHINE=imx6dlsabresd DISTRO=fsl-imx-x11 source ./fsl-setup-release.sh -b build
   $ cd ..
   $ cd sources/
   $ yocto-layer create staubli
   ```

This will ask you for few questions like

```
Please enter the layer priority you'd like to use for the layer: [default: 6] 6
Would you like to have an example recipe created? (y/n) [default: n] y
Please enter the name you'd like to use for your example recipe: [default: example] Hellow
Would you like to have an example bbappend file created? (y/n) [default: n] n

New layer created in meta-test.

Don't forget to add it to your BBLAYERS (for details see meta-test/README).
stage3@favd14504 {7}
```

Thus you have sucessfully created a yocto layer

2. After that enter the following commands to add newly created layer to the image building

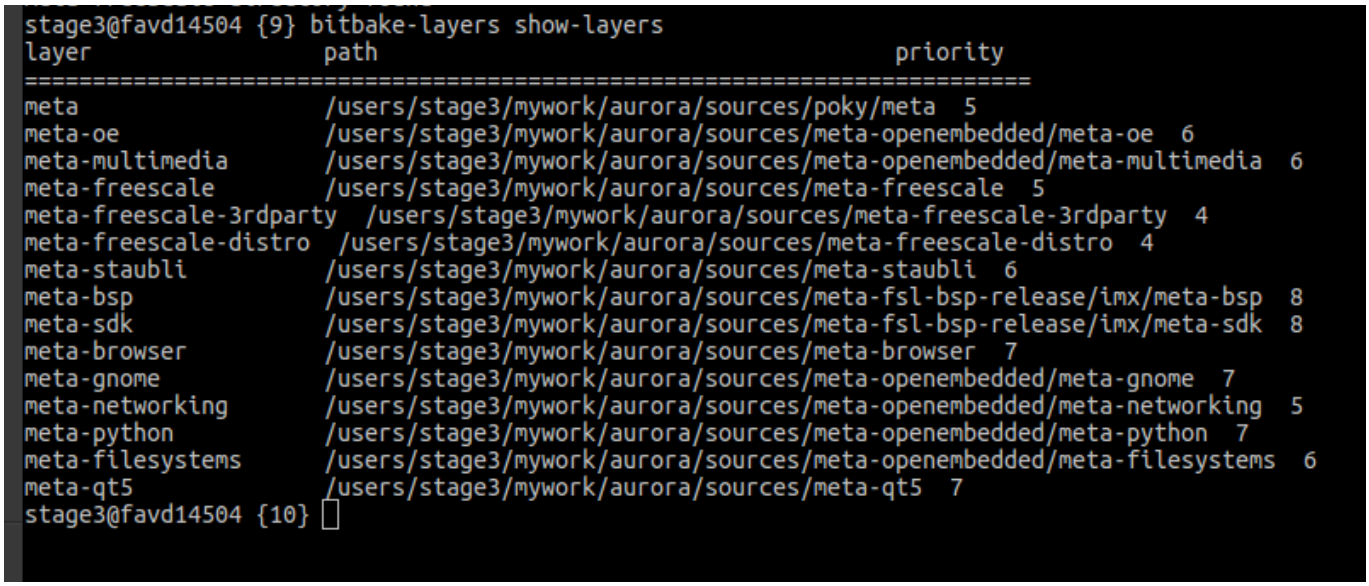
```
$ cd /users/stage3/mywork/aurora/build/conf

$ vi bblayers.conf.org
```

add the newly created layer path as follows

```
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-staubli"
```

3. Next enter the following commands to verify whether newly created layer is added to image building

```
$ cd /users/stage3/mywork/aurora

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

$ bitbake-layers show-layers
```



This displays all the layers which are used for building the image and their respective priority

ADDING THE QT SUPPORT IN THE IMAGE BUILD

1. Enter the following commands

```
$ cd /users/stage3/mywork/aurora/sources/meta-fsl-bsp-release/imx/meta-sdk/recipes-fsl/images
```

2. Take the «fsl-image-qt5.bb», «fsl-image-qt5-validation-imx.bb» and «fsl-image-validation-imx.bb» as a reference to add the Qt support to our custom image
3. Enter the following commands

```
$ cd /users/stage3/mywork/aurora/sources/meta-staubli/recipes-staubli/core-image-minimal/
$ nano core-image-minimal.bbappend
```

4. Edit the «core-image-minimal.bbappend» as follows

Note

You can add or remove the unnecessary packages according to your your need. I have just added the minimum Qt supports and debug tools

```
# 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-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 \
    tools-debug \
    hwcodecs \
    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-gstreamer1.0 \
    packagegroup-fsl-gstreamer1.0-full \
    ${@bb.utils.contains('DISTRO_FEATURES', 'wayland', 'weston-init', '', d)} \
```

```
    ${@bb.utils.contains('DISTRO_FEATURES', 'x11 wayland', 'weston-xwayland xterm',
'', d)} \
"

#IMAGE_FEATURES += "ssh-server-dropbear nfs-server "

#CORE_IMAGE_EXTRA_INSTALL += "packagegroup-tools-bluetooth packagegroup-fsl-tools-
audio packagegroup-fsl-tools-gpu-external ${ERPC_COMPS}"

#QT5_IMAGE_INSTALL_common = "packagegroup-qt5-demos"

#uncommment the below line if you want to integrate the splash screen
#IMAGE_FEATURES += "splash"

#ERPC_COMPS ?= ""
#ERPC_COMPS_append_mx7ulp = "packagegroup-imx-erpc"
```

5. Now rebuild the entire image by following commands

```
$ cd /users/stage3/mywork/aurora/
$ MACHINE=imx6dlsabresd DISTRO=fsl-imx-x11 source ./fsl-setup-release.sh -b build
$ bitbake -c cleanall core-image-minimal
$ bitbake core-image-minimal
```

Hence we have added the Qt support to our custom image

QT APPLICATION

Qt is much more than just a cross-platform SDK - it's a technology strategy that lets you quickly and cost-effectively design, develop, deploy, and maintain software while delivering a seamless user experience across all devices. For more information click the following link → Qt

SETTING-UP THE QTCREATOR

For building a Qt application on the target, We need two things

1. Meta-toolchain-qt5

2. Qtcreator

PROCEDURE TO BUILD META-TOOLCHAIN FOR QT FROM YOCTO

1. Enter the following commands

```
$ cd /users/stage3/mywork/aurora
$ MACHINE=imx6dlsabresd DISTRO=fsl-imx-x11 source ./fsl-setup-release.sh -b build
```

2. Bitbake the toolchain by following command

```
$ bitbake meta-toolchain-qt5
```

   This will take some time to build

3. Open the /home/hdd500Gb/aurora/build/tmp/deploy/sdk folder and click on the «fsl-imx-x11-glibc-x86_64-meta-toolchain-qt5-cortexa9hf-neon-toolchain-4.9.11-1.0.0.sh»
4. Enter the following path to install /home/hdd500Gb/Tools/meta-toolchain-qt5/ and continue the installation process

Hence we have build the toolchain for QT5.

PROCEDURE TO DOWNLOAD AND SETUP QTCREATOR

1. Download and Install the lastest Qtcreator from the link → Qtcreator Download
2. Make sure to select «QT 5,8» in the installation process in the «Select components» list and finish the installation
3. Open the terminal and enter the following command

```
$ source /home/hdd500Gb/Tools/meta-toolchain-qt5/environment-setup-cortexa9hf-neon-poky-linux-gnueabi
$ cd  /home/hdd500Gb/qtcreator/Tools/QtCreator/bin
$ ./qtcreator.sh
```

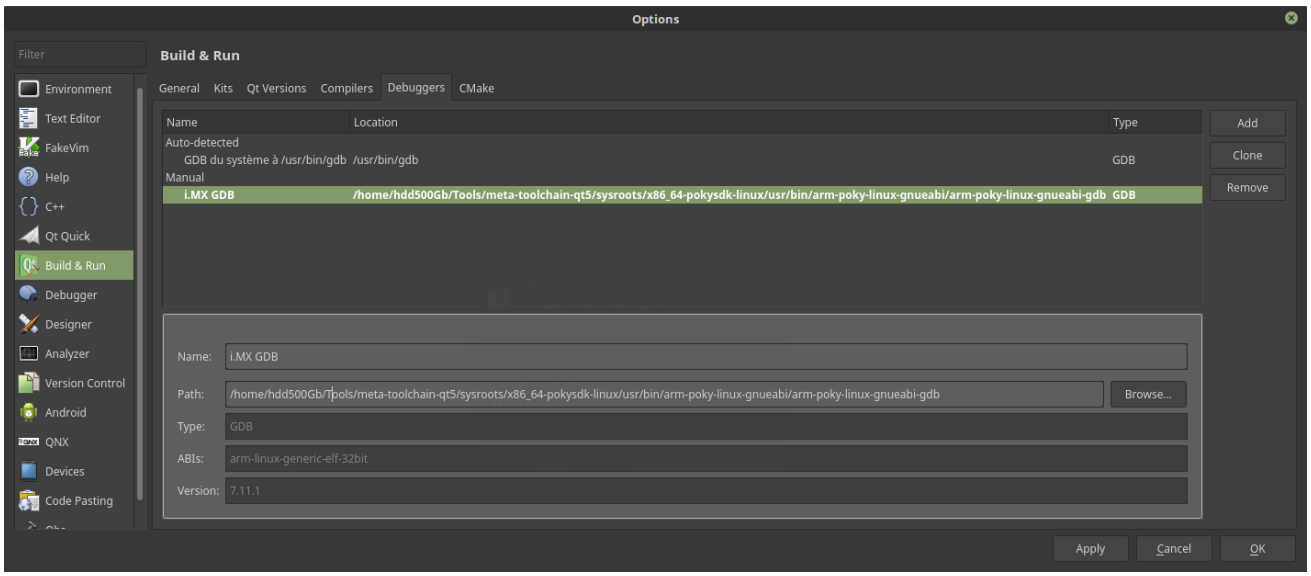
   This opens the Qtcreator.

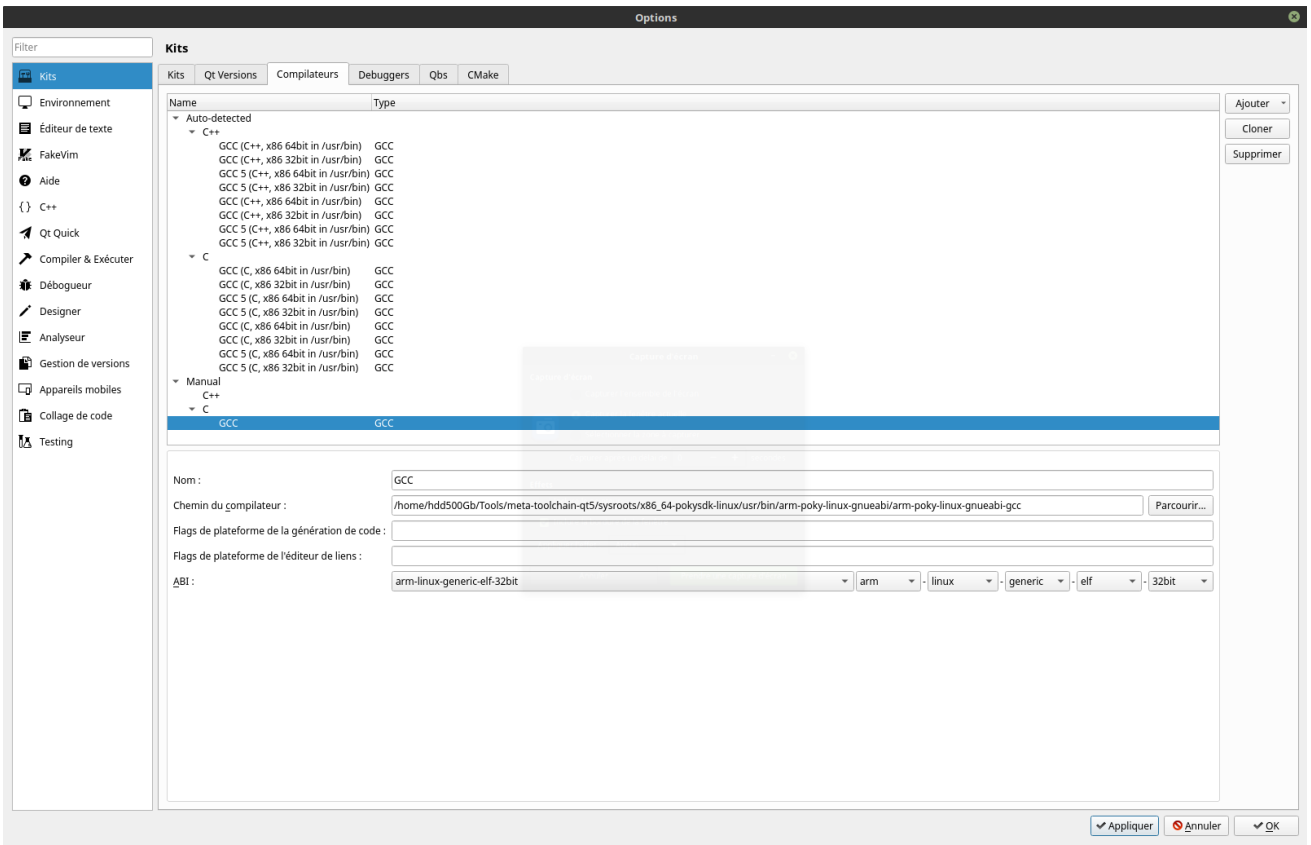
4. Edit the file in /home/hdd500Gb/Tools/meta-toolchain-qt5/sysroots/cortexa7hf-neon-poky-linux-gnueabi/usr/lib/qt5/mkspecs/linux-arm-gnueabi-g++/ qmake.conf

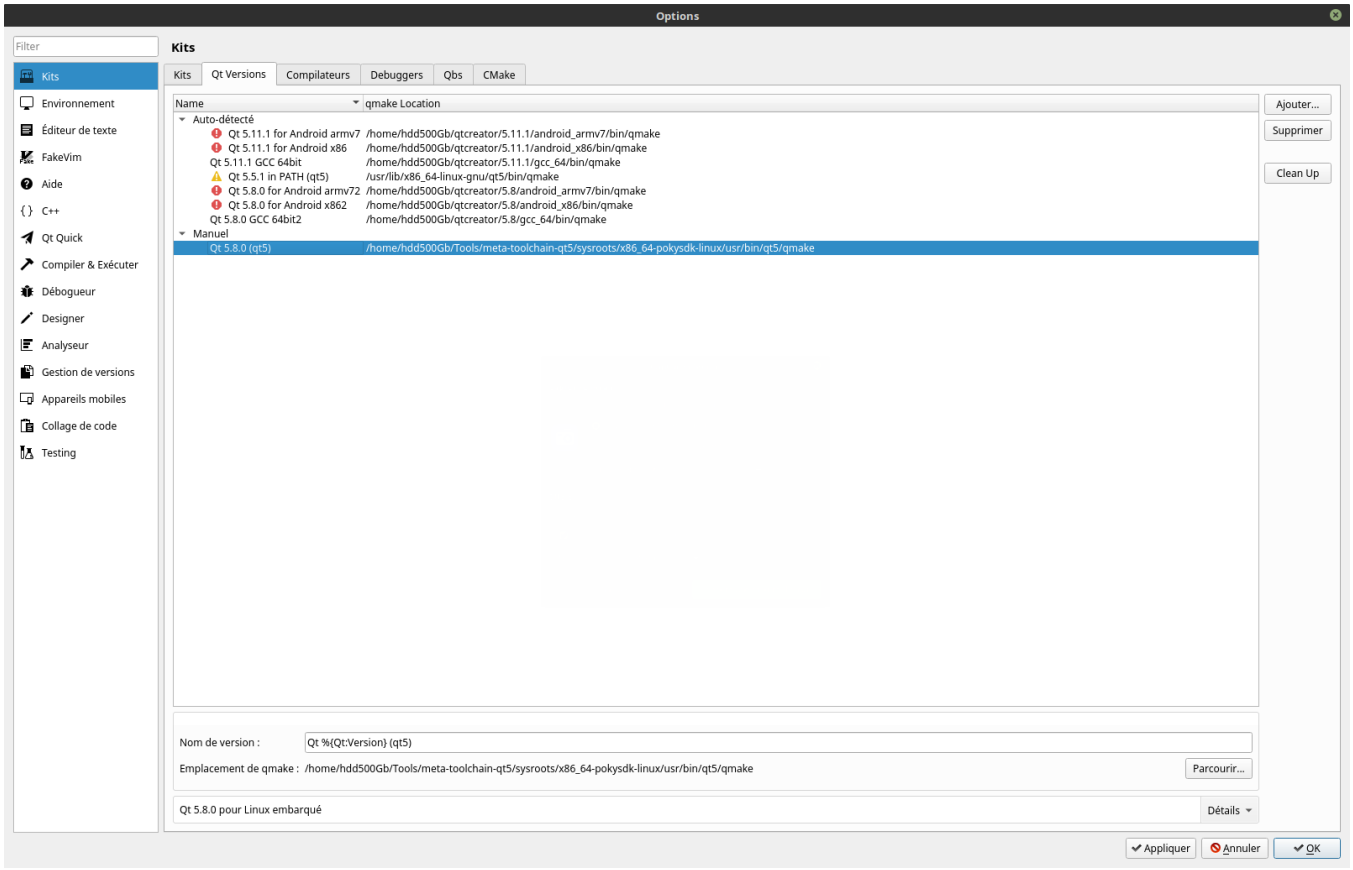
as follows

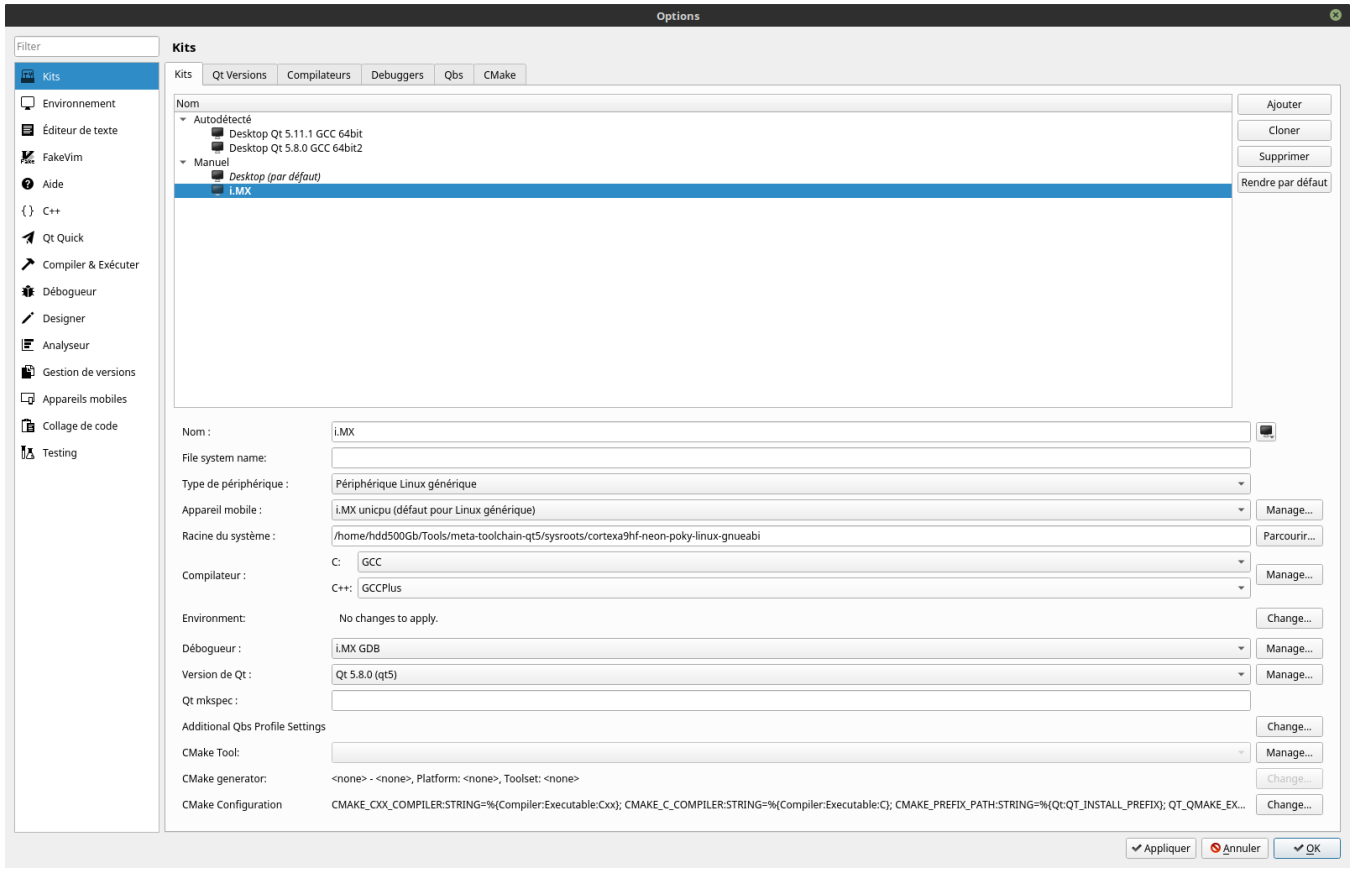
```
#
# qmake configuration for building with arm-linux-gnueabi-g++
#

#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)
```

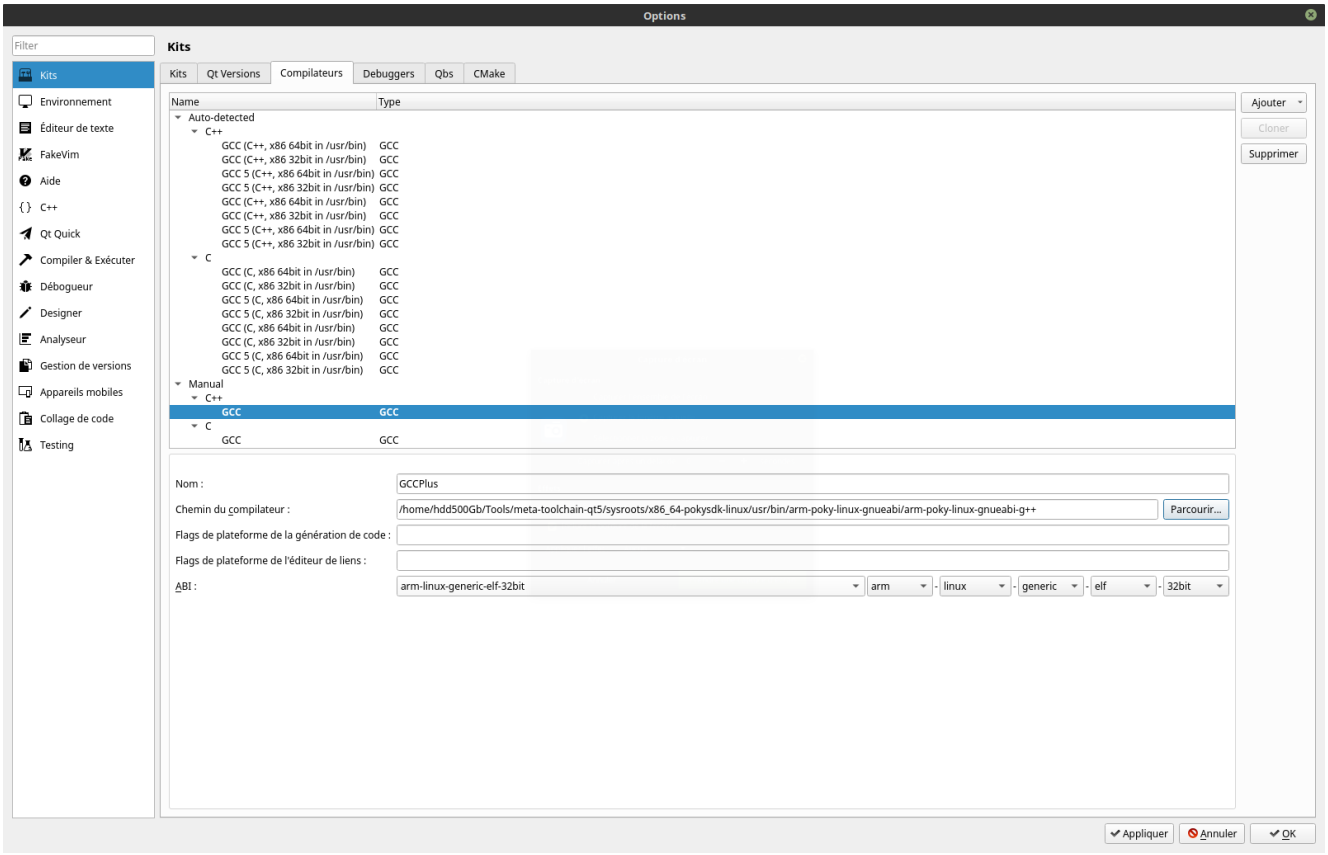
5. In the Qtcreator, follow the procedure for configuring the kits for the compilation and debugging






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



Apply all the changes and press ok. You are ready to use Qtcreator.

HELLOWORLD QT5 APPLICATION COMPILIATION AND INTEGRATION

COMPILATION OF QT5 HELLOWORLD APPLICATION

You can follow this youtube link → Tutorial

INTEGRATING THE QT5 HELLOWORLD APPLICATION INTO YOCTO

1. Enter the following commands

```
$ cd mywork/aurora/sources/meta-staubli/recipes-staubli/helloworldqt5service
$ cp helloworld-0.1 files
$ cd files
```

2. Add «helloworld.cpp» created from the Qtcreator to /files folder and edit it as

```
#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();
}
```

3. Add the «helloworld.pro» created from the Qtcreator to /files folder and edit it as

```
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
```

4. Create, add and edit the «helloworldqt5.service» as

```
[Unit]
Description=Helloworld service
DefaultDependencies=no

[Service]
ExecStart=/usr/bin/helloworldqt5.sh

[Install]
WantedBy=default.target
```

5. Create, add and edit the «helloworldqt5.sh» as

```
#!/bin/sh
export DISPLAY=:0
# We can set the below variable to change the orientation of display
export QT_QPA_EGLFS_ROTATION=90
export QT_EGLFS_IMX6_NO_FB_MULTI_BUFFER=1
export QT_QPA_EGLFS_PHYSICAL_HEIGHT=100
export QT_QPA_EGLFS_PHYSICAL_WIDTH=30
helloworldqt5 -platform eglfs
```

6. Enter the following command

```
$ cd ..
$ touch helloworldqt5service_0.1.bb
```

7. Edit the «helloworldqt5service_0.1.bb» as

```
#
# 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.sh \
            file://helloworldqt5.service \
            file://helloworldqt5.pro \
            file://helloworldqt5.cpp \
"
DEPENDS +="qtbase fontconfig"

S = "${WORKDIR}"

inherit systemd
inherit qmake5

DEPENDS = "qtdeclarative qtgraphicaleffects"
RDEPENDS_${PN} = "qtdeclarative-qmlplugins qtgraphicaleffects-qmlplugins"

require recipes-qt/qt5/qt5.inc

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

```
        install -d ${D}${bindir}
        install -m 0755 ${WORKDIR}/helloworldqt5.sh ${D}${bindir}
        install -d ${D}${systemd_unitdir}/system/
        install -m 0644 ${WORKDIR}/helloworldqt5.service
${D}{systemd_unitdir}/system
}

NATIVE_SYSTEMD_SUPPORT = "1"
SYSTEMD_PACKAGES = "${PN}"
SYSTEMD_SERVICE_${PN} = "helloworldqt5.service"
SYSTEMD_AUTO_ENABLE = "enable"
FILES_${PN} += "${systemd_unitdir}/system/helloworldqt5.service"
```

8. Enter the following commands to add the application package to image build

```
$ cd /users/stage3/mywork/aurora
$ MACHINE=imx6dlsabresd DISTRO=fsl-imx-x11 source ./fsl-setup-release.sh -b build
$ vi conf/local.conf
$ cp conf/local.conf conf/local.conf.org
$ bitbake -c cleanall core-image-minimal
$ bitbake core-image-minimal
```

Hence we have sucessfully integrated our customized QT5 application in form of a systemd service via yocto which launches automatically during the start-up.

OPTIMIZATION OF BOOT TIME

Initially it was taking around 22 seconds for the start-up i.e. Power-on to Qt application launching. We can optimize the boot time by

1. Optimizing the kernel image i.e. uImage and Modules
2. Removing the unwanted systemd service
3. Optimizing the Rootfs
4. Passing some special boot parameters

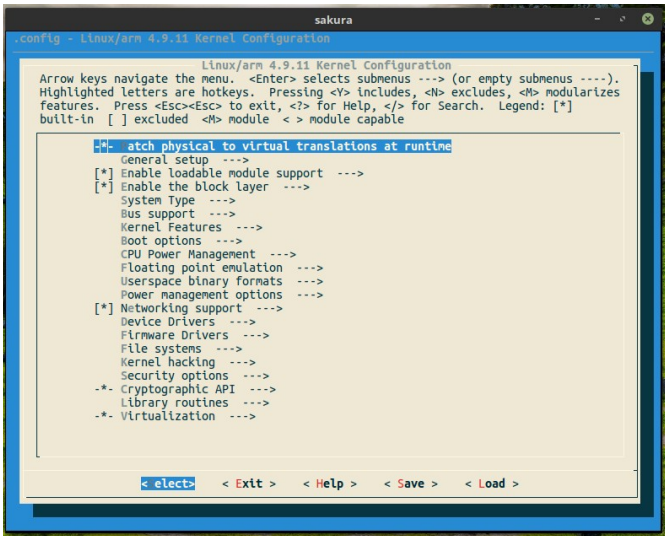
OPTIMIZING THE KERNEL IMAGE

We can remove some unwanted features, compile many unwanted drivers as modules to optimize the kernel image.

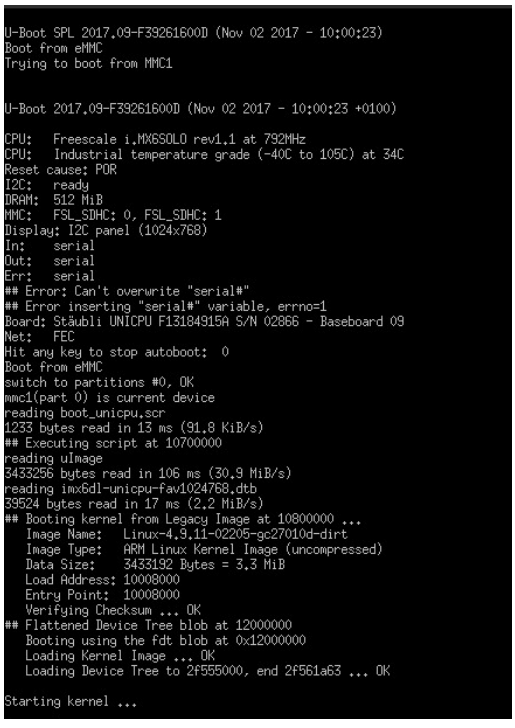
1. Enter the following commands to edit the kernel image

   ```
   $ cd /users/stage3/mywork/linux-imx
   $ source /home/hdd500Gb/Tools/meta-toolchain/environment-setup-cortexa9hf-neon-poky-linux-gnueabi
   $ make menuconfig
   ```

   Here we get an window where we can delete some unnecessary features and make unnecessary device drivers as «modules»



2. When i added the PCI support for the kernel image, It works fine but when i remove the PCI hardware, try to the boot-up, It crashes as follows



3. So, to solve this issue we decided to use two .config files one with PCI support and other without PCI support
4. The .config file with and without PCI support are found in /users/stage3/mywork/.config_files

REMOVING THE UNWANTED SYSTEMD SERVICE

1. Enter the following commands to remove unwanted systemd services

```
$ cd /users/stage3/mywork/aurora/sources/meta-staubli/recipes-staubli/systemd
$ nano systemd_%.bbappend
```

and edit the file as follows which removes the symlinks surgically

```
do_install_append() {
        rm -rf
${D}${sysconfdir}/systemd/system/bluetooth.target.wants/bluetooth.service
        rm -rf
${D}${sysconfdir}/systemd/system/getty.target.wants/getty@tty1.service
        rm -rf ${D}${sysconfdir}/systemd/system/multi-user.target.wants/avahi-
daemon.service
        rm -rf ${D}${sysconfdir}/systemd/system/sockets.target.wants/avahi-
daemon.socket
        rm -rf ${D}${base_libdir}/systemd/system/sysinit.target.wants/systemd-
modules-load.service
        rm -rf ${D}${base_libdir}/systemd/system/sysinit.target.wants/systemd-
journald.service
        rm -rf ${D}${base_libdir}/systemd/system/sysinit.target.wants/systemd-
journal-catalog-update.service
       rm -rf ${D}${base_libdir}/systemd/system/sysinit.target.wants/systemd-
journal-flush.service
       rm -rf ${D}${base_libdir}/systemd/system/sockets.target.wants/systemd-
journald-dev-log.socket
       rm -rf ${D}${base_libdir}/systemd/system/sockets.target.wants/systemd-
journald.socket
       rm -rf ${D}${base_libdir}/systemd/system/sysinit.target.wants/systemd-
vconsole-setup.service
```

```
	rm -rf ${D}${sysconfdir}/systemd/system/sysinit.target.wants/systemd-
timesyncd.service
	rm -rf ${D}/run/systemd/generator/multi-user.target.wants/rc-local.service
	rm -rf ${D}${base_libdir}/systemd/system/multi-user.target.wants/systemd-
logind.service
	rm -rf ${D}${base_libdir}/systemd/system/sockets.target.wants/systemd-
journald-audit.socket
	rm -rf ${D}${sysconfdir}/systemd/system/multi-user.target.wants/remote-
fs.target
	rm -rf ${D}${base_libdir}/systemd/system/sysinit.target.wants/systemd-
random-seed.service
	rm -rf ${D}/sys/fs/cgroup/systemd/system.slice/system-systemd\
x2dbacklight.slice
	rm -rf ${D}{base_libdir}/systemd/system/systemd-backlight@.service
	rm -rf ${D}${sysconfdir}/systemd/system/dbus-org.bluez.service
	rm -rf
${D}${base_libdir}/systemd/system/multi-user.target.wants/dbus.service
	rm -rf ${D}${base_libdir}/systemd/system/sockets.target.wants/dbus.socket
	rm -rf ${D}${base_libdir}/systemd/system/dbus.target.wants/dbus.socket
	rm -rf ${D}${base_prefix}/sys/fs/cgroup/systemd/system.slice/dbus.service
	rm -rf ${D}${sysconfdir}/systemd/system/dbus-org.freedesktop.Avahi.service
}
```

OPTIMIZING THE ROOTFS

In our case, we don’t need sound packages, some debug tools and some default Qt applications and demos. So in order to remove them, I will write an example of «alsa-tools» package which we need for sound

1. Enter the following command

```
$ cd /users/stage3/mywork/aurora
$ MACHINE=imx6dlsabresd DISTRO=fsl-imx-x11 source ./fsl-setup-release.sh -b build
$ nano conf/local.conf
```

Edit the local.conf as follows

```
MACHINE ??= 'imx6dlsabresd'
```

```
DISTRO ?= 'fsl-imx-x11'
PACKAGE_CLASSES ?= "package_rpm"
EXTRA_IMAGE_FEATURES ?= "debug-tweaks ssh-server-openssh"
USER_CLASSES ?= "buildstats image-mklibs image-prelink"
PATCHRESOLVE = "noop"
BB_DISKMON_DIRS = "\
    STOPTASKS,${TMPDIR},1G,100K \
    STOPTASKS,${DL_DIR},1G,100K \
    STOPTASKS,${SSTATE_DIR},1G,100K \
    STOPTASKS,/tmp,100M,100K \
    ABORT,${TMPDIR},100M,1K \
    ABORT,${DL_DIR},100M,1K \
    ABORT,${SSTATE_DIR},100M,1K \
    ABORT,/tmp,10M,1K"
PACKAGECONFIG_append_pn-qemu-native = " sdl"
PACKAGECONFIG_append_pn-nativesdk-qemu = " sdl"

#This includes linuxfb plugins which are useful in displaying our application
PACKAGECONFIG_append_pn-qtbase =  "linuxfb"

"This is used to start "SYSTEMD" as init manager instead of "SYSVINIT" to reduce
the boot time
IMX_DEFAULT_DISTRO_FEATURES_append = " systemd"
DISTRO_FEATURES_append = " systemd"
DISTRO_FEATURES_BACKFILL_CONSIDERED = "sysvinit"
VIRTUAL-RUNTIME_init_manager = "systemd"
VIRTUAL-RUNTIME_initscripts = ""

#This is used to exclude certain default distro features which we don't want in our
application
DISTRO_FEATURES_remove = "x11 wayland alsa bluetooth 3g nfs ext2 pulseaudio bluez5
gstreamer1.0-plugins-base-alsa bluez4"

#This used to enable removal and adding of the packages on the target i.e. after
the image building
IMAGE_FEATURES += "package-management"

#This is used to add our helloworld Qt application as a systemd service
IMAGE_INSTALL_append = "helloworldqt5service"


# This are the packages excluded from the build to optimize the Rootfs
PACKAGE_EXCLUDE_pn-core-image-minimal += " \
alsa-tools \
"
```

```
#This produces a build history folder which is very useful in debugging
INHERIT += "buildhistory"
BUILDHISTORY_COMMIT = "1"

CONF_VERSION = "1"

# It is necessary to place this folders in the /home/hdd500Gb directory to save the
memory available on our users/stage3
DL_DIR = "/home/hdd500Gb/auroraa/downloads"
SSTATE_DIR = "/home/hdd500Gb/auroraa/sstate-cache/"
TMPDIR = "/home/hdd500Gb/auroraa/tmp"

ACCEPT_FSL_EULA = "1"
```

Don’t forget to do the following command after editing the local.conf

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

2. Enter the following command

```
$ cd /users/stage3/mywork/aurora/sources/
$ grep -rnw '/users/stage3/mywork/aurora/sources' -e 'alsa-tools'
```

By this we will come to know what are the other packages need this, So in our search we come to know that «packagegroup-fsl-tools-testapps.bb» has a reverse depended on it. So i need to remove this reverse dependency in order to remove «alsa-tools» package

3. Enter the following commands

```
$ cd /users/stage3/mywork/aurora/sources/meta-staubli/recipes-staubli/packagegroup-
fsl-tools-testapps
$ nano packagegroup-fsl-tools-testapps.bbappend
```

Edit it as follows

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

4. Now rebuild entire image with following commands

```
$ cd /users/stage3/mywork/aurora
$ MACHINE=imx6dlsabresd DISTRO=fsl-imx-x11 source ./fsl-setup-release.sh -b build
$ bitbake -c cleanall core-image-minimal
```

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

5. Verify the buildhistory to know whether the package is removed or not /users/stage3/mywork/aurora/build/buildhistory/images/imx6dlsabresd/glibc/core-image-minimal/installed-package-sizes.txt
6. At present our local.conf looks as follows after the optimization

```
MACHINE ??= 'imx6dlsabresd'
DISTRO ?= 'fsl-imx-x11'
PACKAGE_CLASSES ?= "package_rpm"
EXTRA_IMAGE_FEATURES ?= "debug-tweaks ssh-server-openssh"
USER_CLASSES ?= "buildstats image-mklibs image-prelink"
PATCHRESOLVE = "noop"
BB_DISKMON_DIRS = "\
    STOPTASKS,${TMPDIR},1G,100K \
    STOPTASKS,${DL_DIR},1G,100K \
    STOPTASKS,${SSTATE_DIR},1G,100K \
    STOPTASKS,/tmp,100M,100K \
    ABORT,${TMPDIR},100M,1K \
    ABORT,${DL_DIR},100M,1K \
    ABORT,${SSTATE_DIR},100M,1K \
    ABORT,/tmp,10M,1K"
PACKAGECONFIG_append_pn-qemu-native = " sdl"
PACKAGECONFIG_append_pn-nativesdk-qemu = " sdl"

#This includes linuxfb plugins which are useful in displaying our application
PACKAGECONFIG_append_pn-qtbase =  "linuxfb"

"This is used to start "SYSTEMD" as init manager instead of "SYSVINIT" to reduce the boot time
IMX_DEFAULT_DISTRO_FEATURES_append = " systemd"
DISTRO_FEATURES_append = " systemd"
DISTRO_FEATURES_BACKFILL_CONSIDERED = "sysvinit"
VIRTUAL-RUNTIME_init_manager = "systemd"
VIRTUAL-RUNTIME_initscripts = ""

#This is used to exclude certain default distro features which we don't want in our application
DISTRO_FEATURES_remove = "x11 wayland alsa bluetooth 3g nfs ext2 pulseaudio bluez5 gstreamer1.0-plugins-base-alsa bluez4"

#This used to enable removal and adding of the packages on the target i.e. after the image building
IMAGE_FEATURES += "package-management"

#This is used to add our helloworld Qt application as a systemd service
```

```
IMAGE_INSTALL_append = "helloworldqt5service"

# This are the packages excluded from the build to optimize the Rootfs
PACKAGE_EXCLUDE_pn-core-image-minimal += " \
alsa-conf-base \
alsa-conf \
alsa-lib-dev \
alsa-lib \
alsa-states \
alsa-tools \
alsa-utils-aconnect \
alsa-utils-alsactl \
alsa-utils-alsaloop \
alsa-utils-alsamixer \
alsa-utils-alsatplg \
alsa-utils-alsaucm \
alsa-utils-amixer \
alsa-utils-aplay \
alsa-utils-aseqdump \
alsa-utils-aseqnet \
alsa-utils \
alsa-utils-iecset \
alsa-utils-midi \
alsa-utils-speakertest \
pulseaudio \
fsl-rc-local \
vlan \
cryptodev-module \
cryptodev-tests \
gstreamer1.0-plugins-base-audioconvert \
gstreamer1.0-plugins-base-audiorate \
gstreamer1.0-plugins-base-audioresample \
gstreamer1.0-plugins-base-audiotestsrc \
gstreamer1.0-plugins-base-volume \
gstreamer1.0-plugins-good-audioparsers \
imx-qtapplications \
imx-qtapplications-dev \
gstreamer1.0-plugins-good-flac \
gstreamer1.0-plugins-good-icydemux \
gstreamer1.0-plugins-good-id3demux \
gstreamer1.0-plugins-good-speex \
gstreamer1.0-plugins-good-wavparse \
gstreamer1.0-plugins-base-ogg \
gstreamer1.0-plugins-base-ximagesink \
gstreamer1.0-plugins-base-xvimagesink \
gstreamer1.0-plugins-bad-waylandsink \
```

```
minicom \
memtester \
logrotate \
fbset \
fbset-modes \
can-utils \
cracklib \
latencytop \
powertop \
ttf-dejavu-sans \
strace \
babeltrace \
nano \
blktrace \
trace-cmd \
lrzsz \
nfs-utils \
"
#Increase rootfs size by 500MB
IMAGE_ROOTFS_EXTRA_SPACE += "512000"

#This produces a build history folder which is very useful in debugging
INHERIT += "buildhistory"
BUILDHISTORY_COMMIT = "1"

CONF_VERSION = "1"

# It is necessary to place this folders in the /home/hdd500Gb directory to save the
memory available on our users/stage3
DL_DIR = "/home/hdd500Gb/auroraa/downloads"
SSTATE_DIR = "/home/hdd500Gb/auroraa/sstate-cache/"
TMPDIR = "/home/hdd500Gb/auroraa/tmp"

ACCEPT_FSL_EULA = "1"
```

This is the way we optimize the Rootfs

PASSING SOME SPECIAL BOOT PARAMETERS

I haved passed some boot parameters in «boot_unicpu_fdt_sd_nodhcp» for example

```
# U-boot script sur carte SD - avec dtb
#
# Boot a Linux kernel from the SD card. The scipt and the uImage are located
# in the first partition of the SD card. It is a DOS partition.
# Screens: 800x600 FAV-LDB-SVGA, 1024x768 SAR-LDB-XVGA
```

```
setenv loadaddr 10800000
setenv dtaddr 12000000

setenv mmcdev 1
# Root pour le noyau: mmcblk1p2 pour SD
setenv rootdev /dev/mmcblk3p2

if test $baseboard_id -eq 20 ; then
  setenv dtfile imx6dl-unicpu-sar800600.dtb
else
  if test $x -eq 800 ; then
    setenv dtfile imx6dl-unicpu-fav800600.dtb
  else
    setenv dtfile imx6dl-unicpu-fav1024768.dtb
  fi
fi


# Argument du noyau
setenv bootargs "console=ttymxc0,115200 root=$rootdev rootwait=1 quiet loglevel=3
vga=current rd.systemd.show_status=auto rd.udev.log_priority=3 silent=y verify=n
lpj=30000 vt.global_cursor_default=0"

if load mmc $mmcdev:1 $loadaddr uImage ; then
   if load mmc $mmcdev:1 $dtaddr $dtfile ; then
       bootm $loadaddr - $dtaddr
   fi
fi
```

1. **«Quiet»** is used to disable the console output. Please check the link → Silent boot
2. **«silent»** is used to disable the U-Boot console output
3. **«verify»** is used to disable the CRC checking with a U-boot environment variable
4. **«lpj»** is set because, At each boot, the Linux kernel calibrates a delay loop (for the udelay() function). This measures a number of loops per jiffy (lpj) value on each boot. So if you preset this value after looking for the first time in boot messages, we can save optimize few mseconds

5. **«vt.global_cursor_default»** is used to remove the console cursor at boot which keeps blinking during every start-up

This are the various ways to optimize the boot time. By using this we can bring boot time from 22 seconds to 4,5 Seconds. Please verfiy the following links for your further reference

https://bootlin.com/doc/training/boot-time/boot-time-slides.pdf
https://www.nxp.com/files-static/training_pdf/VFTF09_MONTAVISTA_LINUXBOOT.pdf?&lang_cd=en
https://elinux.org/images/2/2b/Elce11_hart.pdf
http://s3.mentor.com/embedded/linux-fast-boot-iesf2013.pdf