GEAM6UL SW manual Yocto 2.0.5

Getting started manual

DATE	REVISION	CHANGE DESCRIPTION	Author
12/11/2015	1.0.0	Release	
16/04/2016	1.0.1	General review	
07/06/2016	2.0.0	General enhancement for Fido release	Mirko Ardinghi
20/07/2016	2.0.1	Gpio export with name	
05/09/2016	2.0.2	General enhancement	
11/05/2017	2.0.3	Modified variable in Boot from SD chapter and added brightness settings chapter	
07/12/2017	2.0.4	Added MicroGEA/ULL	
18/01/2018	2.0.5	Proofread & correct grammar	

Table of Contents

1. Introduction

1.1 How an embedded Linux boots

Three main software components participate in system startup: the bootloader, the kernel, and the init process in the filesystem.

The U-Boot is the system's boot loader used by Engicam SOM. It starts the machine and contains the basic informations to boot the operative system. By the U-Boot it's possible to program the module and run the basic configurations on the machine. E.g. indicate the locations in which to find the Kernel and file system. The type of media to use to boot: NAND memory, MMC card or NFS and related parameters such as the screen resolution, the console settings, and much more.

The Kernel is the actual operating system of the module. It contains all the drivers of the machine as well as a real Linux Kernel. It's possible to change the Kernel by adding and removing modules or drivers according to your needs. Please for further information refer to the Kernel development environment.

The dtb (Device Tree Blob) file or the device tree is a part of the kernel and includes the hardware description. Each board will have its own dtb file and features. This file is loaded by the bootloader and then passed to the kernel.

The filesystem is the "hard drive" of the module. It contains all the init script and the programs required by the embedded application. These programs specifically compiled for the platform ARM of the module can be managed using the configuration tool of the filesystem called *Yocto*. The filesystem also contains its own applications compiled as well as all the required files to your applications.

1.2 The Linux standard I/O device

The default standard I/O device is the serial console. Refer to chapter 3 for details on RS232 console connector on the Engicam starterkit and for a quick start guide.

2. Engicam BSP overview

2.1 Your Engicam website account

Your account on the Engicam website can be requested by the following form:

http://www.engicam.com/en/component/users/?view=registration

Once the account is confirmed, to enable the BSP download contact your sales person or send an email to support@engicam.com.

2.2 The BSP Virtual Machine

In order to develop your application and configure the U-Boot, the Kernel and the filesystem ENGICAM provides the customer a virtual machine preconfigured and ready to work. Our virtual machine BSP is compatible with both VMware Player and VirtualBox. To use VMware player you need to purchase a commercial license, but you can free download from Vmware an evaluation version. VirtualBox instead is usable with GPL licence.

ENGICAM recommend Vmware player as preferred player for the virtual machine. For both applications it is advisable to always use the latest versions and refer to the license conditions.

The filesystem distribution and configuration are based on *Yocto,* refer to *Yocto* documentation for details. The gcc cross compiler toolchain and the Qt environment are included in the BSP.

A valid internet connection is mandatory for a proper use of the BSP virtual machine. Using firewall network may generated several and serious errors during *Yocto* build process, these errors may corrupt the **built**. Please check and open all TCP port on the network and disable all network firewalls.

WARNING:

all the examples included in this manual are referred to GEA MX6UL, for the other modules refer to the corresponding machines included in the directory

/home/user/yocto/pyro/sources/meta-engicam/conf/machine

2.3 Virtual machine login data

Default user and password for virtual machine are the following:

user: ***user***
password: ***user***

root password (for the administrator privileges): ***user***

At the first login, to update the meta-engicam, run the following command:
cd ~/yocto/pyro/sources/meta-engicam
git pull

2.4 Licenses

Before you install and run the virtual machine, it is recommended a careful reading of the license and the conditions of use of the software.

3. The EDIMM Engicam starterkit first power on

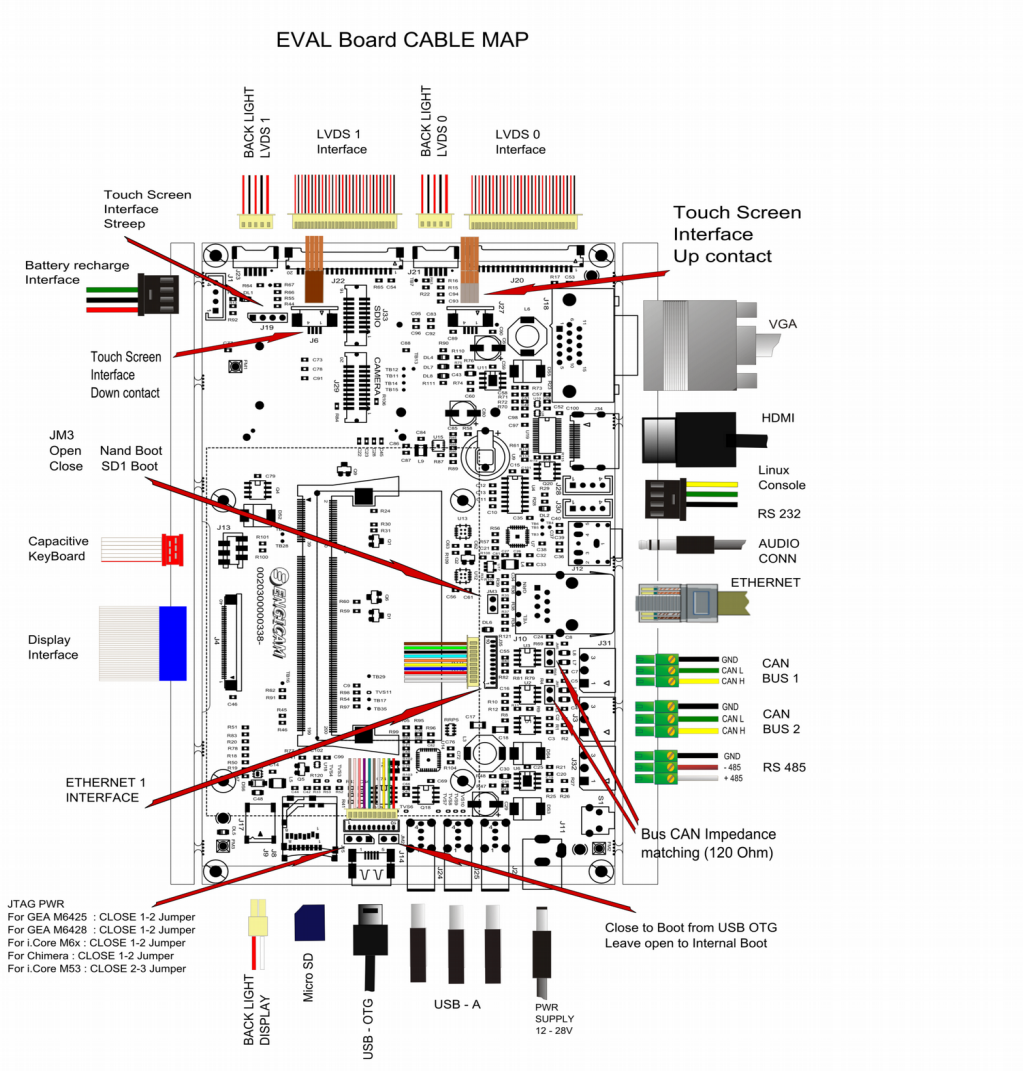
Engicam starterkit comes with CPU module already programmed with a standard image.

The AC power supply adapter and the serial cable are also provided.

Connect the provided serial cable J28 and a PC serial port.

AC adapter shall be connected to J11 Jack connector.

3.1 EDIMM starterkit connectors



Note: to use MicroGEA or GEA-L MX6ULL is necessary the EDIMM adapter board

3.2 The starterkit boot

When the power supply is applied the starter kit boots the default factory Linux image.

The u-boot and Linux standard output is redirected to serial console (J28 connector).

3.3 Login data on target

Default user and password for any Engicam CPU module on the Linux console are the following:

user: ***root***
password: ***(no password)***

3.4 Choose the boot device

The EDIMM starterkit supports the following boot devices, configurable by JM3 jumper:

JM3	Boot device
Opened	internal NAND FLASH
Closed	Micro SD card

4. First *Yocto* Build

Once started the machine, open a terminal window and enter into the directory:

/home/user/yocto/pyro

Then run the command of the configuration environment:

MACHINE=<machine target> source engicam-setup-environment build

Please verify the **/home/user/yocto/pyro/build** directory is entered, if not, retry again the previous command.

This command must be run each time a new shell is opened. Changing the default "build" target directory involves the rebuild of it. The target build directory must be named without the path; the script will provide to generate the folder in the right path (see the example below):

wrong syntax - source engicam-setup-environment /home/user/yocto/pyro/build

correct syntax - source engicam-setup-environment build

These commands will redirect the user in a dedicate *Yocto* command shell, in this shell there will be all the *Yocto* environment variables and path automatically exported. Please remember that from this shell will not be possible to executed all normal Linux command shell, use this shell only for working with *Yocto* system, for the normal shell please open a new Linux shell instance. Remember also to execute the environment script every time it's needed to work with *Yocto* system, when the shell is closed all the environment settings made by the environment script are lost.

It's possible to find the full list of the machines in the following path:

/home/user/yocto/pyro/sources/meta-engicam/conf/machine

Read and agree the licence, then the script sets the environment used for the image compiling.

Now we are ready to build the image, to perform the image's compiling, run command:

bitbake <image name>

where **<image name>** can be on of the following:

engicam-demo-qt	**// demo QT to write on NAND**
engicam-test-hw	**// engicam simple image for hardware test**

After building an image is possible to program a bootable SD card or write on the internal memory. Refer to proper chapter for details.

NOTE

To build the "engicam-demo-qt" image it's necessary to enable the following feature into the local.conf file, carried inside the "build" directory, to settings the environmental variables of the Qt application, used:

STARTUPDEMO = "resistive"	**→ resistive touch**
STARTUPDEMO = "capacitive"	**→ capacitive touch**

In the default mode STARTUPDEMO is setting as "resistive"

NOTE

For NAND programming of GEAM6UL please use only sdcard or linux tftp script.

5. SD card Compiling

After the image generation, it's possible to write the image on an SD card and directly boot the system from the SD card by closing the jumper JM3, or to boot the system by SD using the *Yocto* boot loader parameter, see the bootloader chapter for bootcmd_setup.

For example, build the engicam-demo-qt:

bitbake engicam-demo-qt

After building, an .sdcard file will be generated. These files are SD card image pre-compiled with the bootloader, kernel, device tree and filesystem already included and ready to be written on an SD card. Copy the SD image on a SD card using the command:

sudo dd if=tmp/deploy/images/<machine target>/engicam-demo-qt-<machine target>.sdcard of=/dev/sd<X> bs=10M && sync

where <X> is the device index.

Close the jumper, insert the SD card and power the system. If the system boot from NAND, it is needed to stop the system into the boot mode and run the command:

run bootcmd_mmc

The image into the SD card includes the script required to program the NAND memory, as shown in the NAND programming by Linux tftp.

NOTE:
To settings the booting mode from SD card or NAND, edit the file local.conf

UBOOT_CONFIG="sd" **to boot from SD card saving the parameter on SD**
UBOOT_CONFIG="nand" **to boot from NAND saving the parameter on NAND**

Edit this value to choose how to boot the system and where to save the starting parameters.

6. NAND Programming (SODIMM)

It 's possible to program the NAND using the following ways:

1. NAND programming by u-boot tftp script
2. NAND programming by sdcard script
3. NAND programming by Linux tftp script

6.1 NAND programming by uboot-tftp

Into the directory:

/home/user/yocto/pyro/sources/meta-engicam/tools/uboot_script/mx6/scr-file

there are the followings programming scripts:

- **ker.scr** **(allows to write the kernel on NAND)**
- **dtb.scr** **(allows to write the file .dtb on NAND)**
- **fs.scr** **(allows to write the filesystem on NAND)**
- **ker_dtb.scr** **(allows to write the kernel and the file .dtb on NAND)**
- **ker_dtb_fs.scr** **(allows to write all the file before listed)**

Once an image was compiled, into the target folder choice following files will be available:

- **/tmp/deploy/images/<machine target>/engicam-demo-qt-<machine target>.ubifs**
- **/tmp/deploy/images/<machine target>/uImage-<machine target>.bin**
- **/tmp/deploy/images/<machine target>/uImage-<file>.dtb**

The files will be copied into the virtual machine in the directory "tftpboot".
copy the script (e.g. "ker_dtb_fs.scr") in the previously created folder and files to be saved in the directory NAND iMX6 and rename them as follows:

- **engicam-demo-qt-<machine target>.ubifs → rootfs.ubifs**
- **uImage → uImage**
- **uImage-<file>.dtb → uImage.dtb**

Let's start the module and stop it in **uboot** mode.

Set the IP address of the virtual machine to use to download the programming script. E.g. using the following commands to flash the kernel on NAND:

```
setenv serverip 192.168.xxx.xxx
tftp ker.scr
so
```

It's possible to apply this commands at all the scripts include into the scr-file folder. Then it's possible to run the system from NAND using the following command:

```
run bootcmd_ubi
```

6.2 NAND programming by Linux SD card

Build the first Qt demo image to use it for the NAND writing using the following command:

```
bitbake engicam-demo-qt
```

Once the build is finished, write the image .sdcard just generated as described in the previous chapter.
Copy to the SD the file u-boot, kernel, dtb and filesystem to program the NAND

```
sudo cp tmp/deploy/images/<machine target>/u-boot.imx /media/user/<6e5241db-53ac-4215-ba3e-de74e1e5db36>/u-boot.imx
sudo cp tmp/deploy/images/<machine target>/uImage /media/user/<6e5241db-53ac-4215-ba3e-de74e1e5db36>/uImage
sudo cp tmp/deploy/images/<machine target>/uImage-<file>.dtb /media/user/<6e5241db-53ac-4215-ba3e-de74e1e5db36>/uImage.dtb
sudo cp tmp/deploy/images/<machine target>/engicam-demo-qt-<machine target>.tar.bz2 /media/user/<6e5241db-53ac-4215-ba3e-
de74e1e5db36>/rootfs.tar.bz2
sync
```

Start the system from SD card as described previously.
Enter in the "/" root directory to sdcard and run the following scripts to programming NAND:

```
prboot.sh          // Program the boot
prkernel.sh        // Program the kernel and the dtb
prfs.sh            // Program the filesystem
```

Remove the jumper and reboot the system using NAND. If the system doesn't start automatically (boot from NAND not set) use the following command:

```
run bootcmd_ubi
```

Following the code included inside the programming script

```
prboot.sh
  dd if=u-boot.imx of=/mnt/u-boot_my.imx bs=512 seek=2
  flash_erase /dev/mtd0 0 0
  kobs-ng init -v /mnt/u-boot_my.imx

prkernel.sh
  #kernel programming:
  flash_erase /dev/mtd1 0 0
  nandwrite /dev/mtd1 -p uImage
  #kernel device tree programming:
  flash_erase /dev/mtd2 0 0
  nandwrite /dev/mtd2 -p  /uImage.dtb

prfs.sh
  ubiformat /dev/mtd3
  ubiattach /dev/ubi_ctrl -m 3
  if cat /proc/mtd | grep -q 1f500000; then
    ubimkvol /dev/ubi0 -N rootfs -s495000000
  else
    ubimkvol /dev/ubi0 -N rootfs -s240000000
  fi
  mkdir /rootfs
  mount -t ubifs ubi0:rootfs /rootfs
  tar xvf /rootfs.tar.bz2 -C /rootfs
  sync
```

6.3 NAND programming by Linux tftp

There is the possibility to program the module using TFTP, passing by the SD card (it's not possible to program the NAND during its use). The virtual machine already includes a FTP server installed and ready to use. Copy the programming file inside the TFTP folder:

cp tmp/deploy/images/<machine target>/engicam-demo-qt-<machine target>.tar.bz2	**/tftpboot/rootfs.tar.bz2**
cp tmp/deploy/images/<machine target>/uImage	**/tftpboot/**
cp tmp/deploy/images/<machine target>/ uImage-<file>.dtb	**/tftpboot/uImage.dtb**
cp tmp/deploy/images/<machine target>/u-boot.imx	**/tftpboot/**

Now it's possible to use the following script to program the following elements:

export serverip=192.168.xxx.xxx

tftp_boot.sh
tftp_kernel.sh
tftp_fs.sh

7. eMMC

7.1 eMMC Compiling

To create images booting from eMMC you need to modify your local.conf and change the UBOOT_CONFIG as follows

UBOOT_CONFIG="emmc"

Remember to set UBOOT_CONFIG="emmc"

Choose the file with the emmc word in the name. ex. uImage-imx6ul-gea-emmc.dtb

7.2 eMMC Programming

It's possible to program the eMMC by SD card scripts

To enable these scripts in your sdcard image you need to include the recipe engicam-emmc-tools (available into the layer meta-engicam) to your image

To enable the EMMC from linux you need to load a specific device tree. Change the fdt_file in the u-boot console as follows

fdt_file=imx6ul-gea-emmc.dtb

Run the system from SD card.

You can choose to program your eMMC via tftp or via usb

7.3 Programming via tftp

Once an image was compiled, into the target folder choice following files will be available:

- **/tmp/deploy/images/<machine target>/engicam-demo-qt-<machine target>.tar.bz2**
- **/tmp/deploy/images/<machine target>/u-boot.imx**[1]
- **/tmp/deploy/images/<machine target>/uImage-<machine target>.bin**
- **/tmp/deploy/images/<machine target>/uImage-<file>.dtb**[2]

The files will be copied into the virtual machine in the directory "tftpboot".

Copy the files in the directory and rename them as follows:

- **engicam-demo-qt-<machine target>.tar.bz2 → rootfs.tar.bz2**
- **uImage-<machine target>.bin → uImage**
- **uImage-<file>.dtb → <file>.dtb**
- **u-boot.imx → u-boot.imx**

Now it's possible to use the followig scripts to program the eMMC:

a) **emmc_boot_tftp.sh** : write the u-boot. Require u-boot.imx in the tftpboot directory

b) **emmc_fs_tftp.sh** : write the root filesystem. Require rootfs.tar.bz2 in the tftpboot directory
c) **emmc_ker_tftp.sh** : write the kernel. Require uImage in the tftpboot directory
d) **emmc_dtb_tftp.sh** : write the device tree. Require a dtb file in the tftpboot directory
e) **emmc_ker_dtb_tftp.sh** : write the kernel and the device tree. The required files are the union of (c)(d)

f) **emmc_fs_ker_dtb_tftp.sh** : write the kernel, the device tree and the root filesystem. The required files are the union of (b)(c)(d)

For the usage and parameters run the script with the parameter **-h**
This scripts are available when you boot your system from sdcard. A subset of them (a)(c)(d) will be available too when you boot from emmc.

7.4 Programming via usb

Once an image was compiled, into the target folder choice following files will be available:

- **/tmp/deploy/images/<machine target>/engicam-demo-qt-<machine target>.tar.bz2**
- **/tmp/deploy/images/<machine target>/engicam-demo-qt-<machine target>.sdcard**
- **/tmp/deploy/images/<machine target>/u-boot.imx (1)**
- **/tmp/deploy/images/<machine target>/uImage-<machine target>.bin**
- **/tmp/deploy/images/<machine target>/uImage-<file>.dtb (2)**

The files will be copied and renamed into an usb drive as follows:

- **engicam-demo-qt-<machine target>.tar.bz2 → rootfs.tar.bz2**
- **engicam-demo-qt-<machine target>.sdcard → engicam-demo-qt.sdcard**
- **uImage-<machine target>.bin → uImage**
- **uImage-<file>.dtb → <file>.dtb**
- **u-boot.imx → u-boot.imx**

Insert the usb drive in your board and mount it:

mount /dev/sda1 /mnt

Now it's possible to use the followig scripts to program the eMMC:

- **emmc_boot.sh** : write the u-boot. Require u-boot.imx in the usb drive
- **emmc_fs.sh** : write the root filesystem. Require rootfs.tar.bz2 in the usb drive
- **emmc_ker.sh** : write the kernel. Require uImage in the usb drive
- **emmc_dtb.sh** : write the device tree. Require a dtb file in the usb drive
- **emmc_ker_dtb.sh** : write the kernel and the device tree. The required files are the union of (i)(j)
- **emmc_fs_ker_dtb.sh** : write the kernel, the device tree and the root filesystem. The required files are the union of (h)(i)(j)
- **emmc_sdcard.sh** : write the entire emmc. Require an sdcard image in the usb drive

For the usage and parameters run the script with the parameter **-h**

7.5 NAND programming by Linux tftp

There is the possibility to program the module using TFTP, passing by the SD card (it's not possible to program the NAND during its use). The virtual machine already includes a FTP server installed and ready to use. Copy the programming file inside the TFTP folder:

cp tmp/deploy/images/<machine target>/engicam-demo-qt-<machine target>.tar.bz2	**/tftpboot/rootfs.tar.bz2**
cp tmp/deploy/images/<machine target>/uImage	**/tftpboot/**
cp tmp/deploy/images/<machine target>/ uImage-<file>.dtb	**/tftpboot/uImage.dtb**
cp tmp/deploy/images/<machine target>/u-boot.imx	**/tftpboot**

Now it's possible to use the following script to program the following elements:

export serverip=192.168.xxx.xxx

tftp_boot.sh
tftp_kernel.sh
tftp_fs.sh

8. SDK Installation for C development

To have the access to C compilers is required to install and compile the toolchain and the appropriate SDK. The execution of the following steps is needed to customize and to edit the kernel and the bootloader. Enter in your work folder name:

yocto/pyro/FOLDER_NAME

and build the toolchain by the command:

bitbake meta-toolchain

Install it using the command:

cd tmp/deploy/sdk/
./fslc-framebuffer-glibc-x86_64-engicam-demo-qt-armv7at2hf-neon-toolchain-2.3.1.sh
Enter target directory for SDK (default: /opt/fslc-framebuffer/2.3):

At the end of this procedure the C compiler are installed

9. Packet make clean

To force the compiling of a packet after a modify or after some settings changes, is needed to run the following command to execute the clean and force the compiling through bitbake. This command run the config packet again

bitbake -c cleansstate <packet-name>

E.g. for kernel and u-boot

bitbake -c cleansstate u-boot-eng
bitbake -c cleansstate engicam-linux-fslc

10. Development environment for U-boot

As already mentioned, you can change and configure U-Boot according to your own needs. It is not recommended to make changes and even delete the first sectors of flash U-Boot where it resides, this to avoid "Brik " the module. The official boot application is the "Das-U-Boot" (http://www.denx.de/wiki/U-Boot).
For further information about U-Boot refer to the official manual page:

http://www.denx.de/wiki/DULG/Manual

10.1 U-Boot compiling

Run the command to open the U-boot development shell:

```
bitbake -c devshell u-boot-eng
```

By this shell it's possible to set the U-Boot by running the following export:

```
./opt/fslc-framebuffer/2.3/environment-setup-armv7at2hf-neon-fslc-linux-gnueabi
export ARCH=arm
export CROSS_COMPILE=$TARGET_PREFIX
unset LDFLAGS
```

To compile the bootloader run the config and then the make command:

```
make geamx6ul_nand_defconfig
make
```

This is the possible config:

SODIMM		
NAND	EMMC	MMC
geamx6ul_nand_defconfig	geamx6ul_emmc_config	geamx6ul_sd_config

Write the bootloader on SD card using the command:

```
sudo dd if=u-boot.imx of=/dev/sd<X> bs=512 seek=2
```

10.2 U-Boot Environment Variables

U-Boot booting the machine and is responsible for passing parameters to boot the machine using the bootargs. These include the type of filesystem to be used, the partitions, the physical way by which you can boot, the mac address. Using the "**printenv**" command you can view the list of environment variables. Using the "**setenv**" command you are able to set variable values.

```
setenv serverip 192.168.1.2        set an environment variable
setenv serverip                    delete an environment variable
```

Once user has set the environment variable he must save the setup using "**save"** command.

If you reprogram the U-Boot, the environment variables are reset to the default value that should already be pre-set to boot the module from NAND. In the case of loss of environment variables, see Deleting of the U-

Boot parameters.

> ***WARNING***
> DO NOT SAVE the environment parameters after a kernel loading failure, this may generate a **Kernel Panic** error. If this occur restore the default parameters see Deleting of the U-Boot parameters

10.2.1 Bootcmd setup

Once stopped the boot into the U-Boot, it's possible to boot from the various devices by running the various bootcmd via the following commands:

```
run bootcmd_mmc          // boot from SD card
run bootcmd_emmc         // boot from EMMC
run bootcmd_ubi          // boot from NAND with UBI filesystem
run bootcmd_net          // boot from network with nfs filesystem
```

If you wish to associate one of these commands to the default boot, without having to stop in U-boot mode, will be enough to save the parameters of the bootcmd associating it with the desired bootcmd:

```
setenv bootcmd   'run bootcmd_mmc'    //boot by default from MMC
setenv bootcmd   'run bootcmd_emmc'   //boot by default from EMMC
setenv bootcmd   'run bootcmd_ubi'    //boot by default from NAND with ubi filesystem
setenv bootcmd   'run bootcmd_net'    //boot by default from NETWORK with nfs filesystem
```

10.2.2 Deleting of the U-Boot parameters

To delete the parameter individually is possible to use the following command:

```
setenv <parameter_name>
```

You can restore the default parameters of the U-Boot deleting existing ones. From uboot version 1:09 is implemented a command to perform the erase of the parameters on both the modules

```
env default -a
save
```

> **WARNING**
>
> This operation implies the loss of the factory parameters such as the mac address. After the erase remember to reset the parameters of the ethaddr for the MAC managing with the one assigned to the module, the MAC address is indicated on the label on top of the CPU (the field starts to 9C53CD)

11. Kernel development environment

Run the following command to open the kernel development shell:

```
bitbake engicam-linux-fslc -c devshell
```

Using this shell to modify the kernel running in sequence the following export:

```
./opt/fslc-framebuffer/2.3/environment-setup-armv7at2hf-neon-fslc-linux-gnueabi
export ARCH=arm
export CROSS_COMPILE=$TARGET_PREFIX
unset LDFLAGS
export LOADADDR=0x80008000
```

To compile the kernel and make the uImage to load also via tftp use command:

```
make -j 8 uImage
```

Then generate the device tree file .dtb using the command:

```
make dtbs
```

The result will be found in the folder $KBUILD_OUTPUT. To go to the folder use the command:

```
cd $KBUILD_OUTPUT
```

Follow the indications how to write the kernel and dtb file on the Nand or SD Card as described in Edit this value to choose how to boot the system and where to save the starting parameters.

11.1 Kernel config

Access to the Kernel's configurer using the following command:

```
bitbake -c devshell engicam-linux-fslc
```

and:

```
make menuconfig
```

To save the changes permanently, copy the hidden file .config from the main kernel directory (this file contains all the menuconfig settings) and rename it as defconfig, do that with command as below:

```
cd $KBUILD_OUTPUT
cp .config /home/user/Desktop/defconfig
```

Now replace this file in each machine config folder in the kernel recipes meta-layer

```
recipes-kernel/linux/engicam-linux-fslc
```

Replacing this file will overwrite the new kernel configuration; during kernel building this file will be used (depending for the machine selected) as the default .config file.

12. Device Tree

From this kernel version the platform file is not present any more for ARM architecture, and the hardware description is made by the device tree file, which may be dynamically load from the bootloader to allow to have only one monolithic, hardware free, kernel.

This chapter describes the hierarchical structure about Engicam board of the device tree file. For further information on how to use the device tree, refer to the official documentation and the site "devicetree.org".

The PAD pin to be used is inside the file

linux/arch/arm/boot/dts/imx6ul-pinfunc.h
....

```
/*
 * The pin function ID is a tuple of
 * <mux_reg conf_reg input_reg mux_mode input_val>
 */
        #define   MX6UL_PAD_BOOT_MODE0__GPIO5_IO10                    0x0014 0x02A0 0x0000 5 0
        #define   MX6UL_PAD_BOOT_MODE1__GPIO5_IO11                    0x0018 0x02A4 0x0000 5 0
        ....
```

These "define" must be inserted inside the device tree to initialize the pins.

All the source files of the device tree are included inside arch/arm/boot/dts and have the following extension:

.dtb	**Output file from compile, readable from uboot**
.dts	**high-level device tree file which will be compiled**
.dtsi	**file included in .dts which contain a higher level of abstraction**

The device tree files are

`geamx6ul-starterkit.dts` **high level file, one file for each board and custom board for UL property**

These .dtsi file are included by .dts and contain generic specification of icore modules and imx6 modules

imx6ul.dtsi generic file .dtsi contain all generic configuration regarding icore modules

Any changes in the device tree can be made at the different file's levels considering the differentiation of the level. Note that all .dts will be maintained by Engicam development team.

12.1 How to select a device node

To use a resource, the corresponding device node must be located in the kernel sources and declared in the device tree.

Below is showing the location by name and the location by path. The first way (by name) is recommended.

12.2 Locate by name

It's possible to locate a node in the device tree by name. In the following example is showing how to locate a node by name, using a function as a recursive method. In the example the secondary node is included inside the primary node.

```
struct device_node *main_node = NULL;
struct device_node *secondary_node = NULL;
bool value_found_in_dtb=false
```

```
main_node = of_find_node_by_name(NULL, "main node");
 secondary_node = of_find_node_by_name(main_node, "secondary node");
if (of_get_property(main_node, "value_to_look_for", NULL))
{
        value_found_in_dtb=true;
}
```

12.3 Locate by address path

This chapter describes how to derive the path argument to pass.
Identify the name of the node to find into IOMUXC

```
&uart1 {
        pinctrl-names = "default";
        pinctrl-0 = <&pinctrl_uart1>;
        status = "okay";
};

pinctrl_uart1: uart1grp {
        fsl,pins = <
                MX6UL_PAD_UART1_TX_DATA__UART1_DCE_TX 0x1b0b1
                MX6UL_PAD_UART1_RX_DATA__UART1_DCE_RX 0x1b0b1
        >;
};
```

In the example above the node searched is called uart1.
Now find inside the file .dts the mapping in the memory of the node that corresponds to the serial@02020000

```
uart1: serial@02020000 {
        compatible = "fsl,imx6ul-uart",
        "fsl,imx6q-uart", "fsl,imx21-uart";
        reg = <0x02020000 0x4000>;
        interrupts = <GIC_SPI 26 IRQ_TYPE_LEVEL_HIGH>;
        clocks = <&clks IMX6UL_CLK_UART1_IPG>,
        <&clks IMX6UL_CLK_UART1_SERIAL>;
        clock-names = "ipg", "per";
        status = "disabled";
};
```

It's necessary to find out in which bus node is the device, this can be understood in finding in which node is inserted uart1: serial@02020000

```
aips1: aips-bus@02000000 {
        compatible = "fsl,aips-bus", "simple-bus";
        #address-cells = <1>;
        #size-cells = <1>;
        reg = <0x02000000 0x100000>;
ranges;
```

12.4 Edit a new dts for a custom board

When you create a new board it is recommended to maintain the structure currently present in the DTS.
As already mentioned in the chapter structure of the Device Tree there is an inclusive hierarchy for the generation of the **geamx6ul- <board_name>.dts.**
Starting from scratch a new dts file is deprecated, start from an existing one from dts folder. Copy it and then edit with your modifications.

12.5 Build a new dts file

Each time a new dts file is created, it must be included in the "makefile" and it must be built.

arch/arm/boot/dts/Makefile

```
imx6sx-sdb-lcdif1.dtb \
imx6ul-14x14-lpddr2-arm2.dtb \
imx6ul-14x14-ddr3-arm2.dtb \
imx6ul-14x14-ddr3-arm2-adc.dtb \
imx6ul-14x14-ddr3-arm2-emmc.dtb     \
imx6ul-14x14-ddr3-arm2-flexcan2.dtb \
imx6ul-14x14-ddr3-arm2-gpmi-weim.dtb \
imx6ul-14x14-ddr3-arm2-lcdif.dtb \
imx6ul-14x14-ddr3-arm2-mqs.dtb \
imx6ul-14x14-ddr3-arm2-spdif.dtb \
imx6ul-14x14-ddr3-arm2-wm8958.dtb \
imx6ul-14x14-evk.dtb \
imx6ul-14x14-evk-csi.dtb \
+       imx6ul-gea.dtb \
        vf610-cosmic.dtb \
```

For each kernel compile, a new dts file will be generated.

12.6 Device tree setup

Now it is required to load the right device tree at the bootloader. In the current state all the compiling, performed by *Yocto*, produce a different dtb for each Engicam board. The dtb labelled "**gea-loco7 and gea-loco43**" are referred to the open-frame resistive board, the dtb labelled **"gea-lococap-7"** is referred to the capacitive open-frame board and the dtb labelled "**gea**" is referred to the starterkit.

<module_type>-gea-loco7.dtb
<module_type>-gea-loco43.dtb
<module_type>-gea-lococap-7.dtb

12.7 Boot from SD

If the start is done from the SD card, into the boot partition containing the boot loader are entered both the dtb file. Stop in the boot and load the correct file dtp running the command:

setenv fdt_file geamx6ul-starterkit.dtb	**for starterkit**
setenv fdt_file imx6ul-gea-loco7.dtb	**for resistive openframe 7"**
setenv fdt_file imx6ul-gea-loco43 .dtb	**for resistive openframe 4.3"**
setenv fdt_file imx6ul-gea-lococap-7.dtb.dtb	**for capacitive openframe 7"**

12.8 Settings a recipe for a new board

Once generated, the dts must be included into a new machine dedicated to the own board. The machine's .conf file are located in the folder:

/home/user/yocto/pyro/sources/meta-engicam/conf/machine

Editing the KERNEL_DEVICETREE, is created the list of the device tree, which will be copied in the bootimage of the SD partition or prepared for the build on the NAND.

KERNEL_DEVICETREE = "geamx6ul-starterkit.dtb "

Remember to select the machine in the build settings, once that a new machine is generated or a pre-existing one is edited (refer to The EDIMM Engicam starterkit first power on).

12.9 How to select the device tree during the boot

Now select the appropriate device tree during the boot mode; there are two different way to select the device tree depending on the boot's device.

12.9.1 Booting from SD

If the bootloader starts from SD card, the bootloader partition must be included both the file dtb. To load the right dtb file, stop in the boot mode and edit the following command:

```
setenv fdt_file geamx6ul-starterkit.dtb
```

The boot from SD card allows to modify the sources of the bootloader to set the new default value for the fdt_file parameter

12.9.2 Boot from NAND

When the system is started from NAND the file uImage.dtb must be copied with the correct dtb source with respect to the open-frame. See details Edit this value to choose how to boot the system and where to save the starting parameters.

13. Filesystem

To add or remove packages from the file system is required to customize an own custom recipe. It's necessary to maintain the inheritance of several other recipes that contain packages for the core and for the module itself including the files .dts. Via web and the official *Yocto* page, it's possible to find all the information and documentations needed for the customization.

13.1 Busybox Setup

To configure the busybox make the first compile and run the configurer using the command:

```
bitbake -c menuconfig busybox
```

To make the changes permanent, copy the modified .config file from the local busybox devshell build folder into the *Yocto* recipes folder and rename it defconfig.

```
bitbake -c devshell  busybox
cp <busybox folder>/.config /home/user/yocto/pyro/sources/meta-engicam/recipes-core/busybox/busybox/defconfig
```

13.2 Compiling external packages

To compile the external packages it's necessary to export all the cross compilers and the toolchain inside the own used environment. Enter in the source folder, where the make file is storage and run the following export:

```
./fslc-framebuffer-glibc-x86_64-engicam-demo-qt-armv7at2hf-neon-toolchain-2.3.1.sh
export ARCH=arm
export CROSS_COMPILE=$TARGET_PREFIX
unset LDFLAGS
```

the reference in the CC compiler shell, will be referred to the target of the arm-gcc.
Remember to check inside the own makefile there are not any static reference to CC x86 compiler and replace it with arm-linux-gcc compiler**:**

```
CC= arm-linux-gcc
```

13.3 Compiling additional packages

Sometimes additional packages could be necessary to build the own application.

13.3.1 Package compiled by toolchain

If the package is present into a layer add it to the recipe, then recompile it. Once recompiled insert the recipe in the toolchain and recompile the toochain again.

```
bitbake engicam-test-hw
bitbake meta-toolchain
bitbake engicam-test-hw -c populate_sdk
```

Installing the Toolchain see How to Install the toolchain. Confirm the overwrite and the install path.

Now all the package lib will be available and exported into the toolchain. Referred to the Package compiled by toolchain to application compiling.

13.4 Network setup

The network configuration is done by a *Yocto* recipes. It's possible to customize the settings by editing the file:

```
gedit meta-engicam/recipes-core/init-ifupdown/init-ifupdown/interfaces
```

The loopback interface

```
auto lo
iface lo inet loopback

auto eth1
iface eth1 inet static
        address 192.168.2.67
        netmask 255.255.255.0
        network 192.168.2.0
        gateway 192.168.2.1

# Wireless interfaces
iface wlan0 inet dhcp
        wireless_mode managed
        wireless_essid any
        wpa-driver wext
        wpa-conf /etc/wpa_supplicant.conf
```

Editing this file, the default values to start the network from image will be changed. If you would like to change the network address run time to the target you should editing the file

```
etc/network/interfaces
```

After editing, sync the filesystem with the "sync" command and reboot the device and you will change the network settings permanently on target.

14. How to build a C application

Built a "Hello World" application in C is easy and fast. For example, try to compile the following code:

HelloWorld.c

```
#include<stdio.h>

main()
{
   printf("Hello World");
   return 0;
}
```

```
/opt/fslc-framebuffer/2.3/sysroots/x86_64-fslcsdk-linux/usr/bin/arm-fslc-linux-gnueabi/arm-fslc-linux-gnueabi-gcc  HelloWorld.c
–sysroot=/opt/fslc-framebuffer/2.3/sysroots/armv7at2hf-neon-fslc-linux-gnueabi/ -mfloat-abi=hard -o HelloWorld
```

Now you can copy and run the application on the end target.

15. Qt Application Development

The virtual machine has already pre-configured in order to connect the QT and to run the deploy directly on the target. If a version of the Qt, different from that included on the virtual machine, is recompiled it will be necessary to reconfigure manually the Qt creator. The version change may take place by updating the meta-layer qt. Remember that the Qt revision on the Linux machine will be match with the one included on the target. In the following chapter will be shown all the operations to execute to realize an own Qt application, from the toolchain generation to the deploy of the application.

15.1 How to Install the toolchain

The toolchain includes all the library, the packets and the executables files needed to compile the own application. Re-build the toolchain and re-install after every update of the layer Qt. The default Qt toolchain is already installed inside the virtual machine for the t 5.8 Compile the toolchain via bitbake using command:

bitbake engicam-demo-qt

run the script to install the toolchain:

./tmp/deploy/sdk/fslc-framebuffer-glibc-x86_64-engicam-demo-qt-armv7at2hf-neon-toolchain-2.3.1.sh

It's possible to install the toolchain using the default directory or another directory.

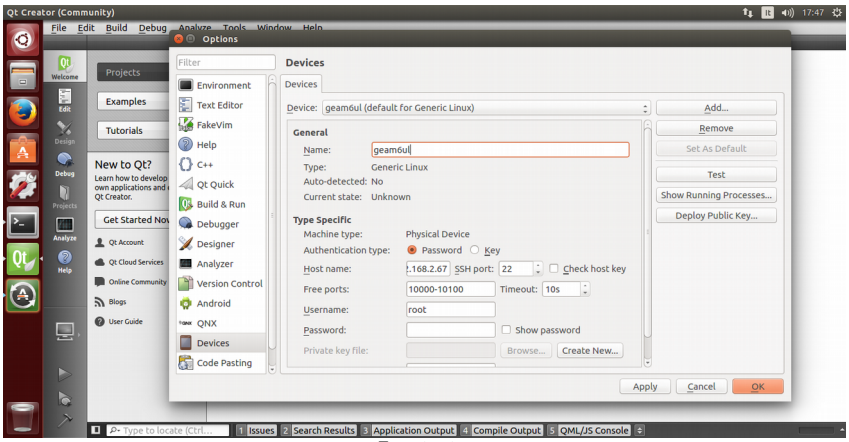
15.2 Starting Qt creator

Run the script from a shell

. /opt/fslc-framebuffer/2.3/environment-setup-armv7at2hf-neon-fslc-linux-gnueabi
~/Qt/Tools/QtCreator/bin/qtcreator

15.3 Qt creator config

This chapter describes how to setup a new installation of Qt creator; this setup is already included in Engicam's *Yocto* BSP. First things to setup is the connection between the virtual machine and the target. The target default ip address is 192.168.2.67, test the connectivity between virtual machine and the target using ping command and then be sure to are using a target image supporting SSH connection. Now you can configure the target device:



Target setup

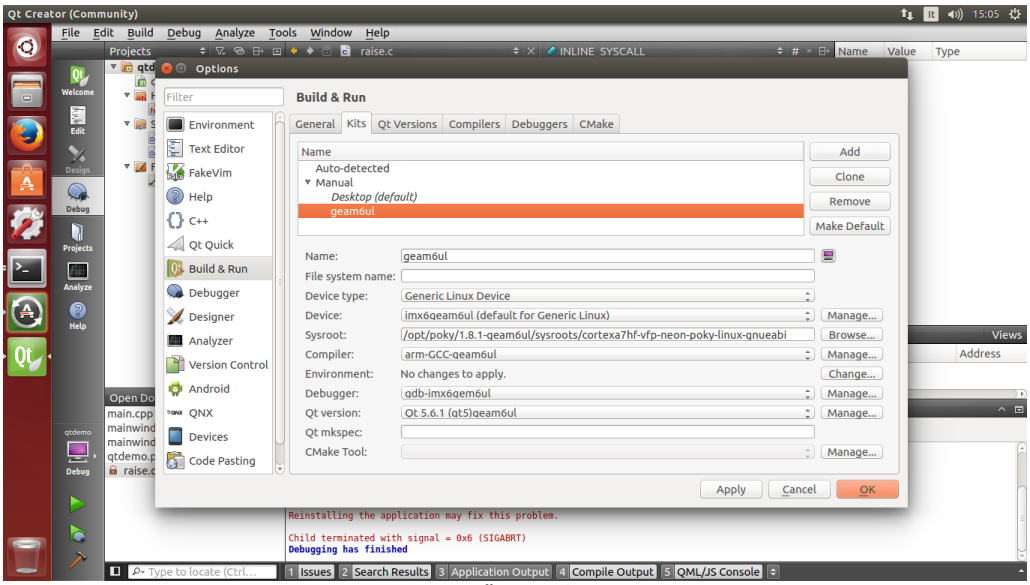
Now setup the link on each compiler, Qt libraries and debugger. All the paths indicate in these figures may change between your meta-engicam distribution version. It's recommended, when indicated, to find the correct path in the toolchain deploy folder of your machine.

The toolchain folder are generically installed in a path like this, this example refer to poky distribution version 2,3

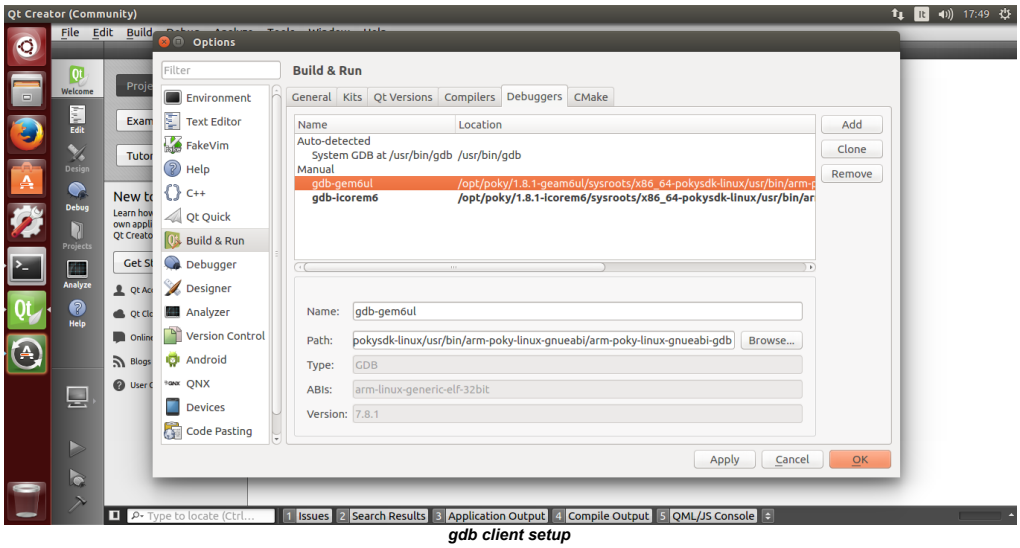
/opt/fslc-framebuffer/2.3

For example to find the compiler path of arm-poky-linux-gnueabi-gcc, run this command to get the correct path:

cd /opt/fslc-framebuffer/2.3
/opt/fslc-framebuffer/2.3/sysroots/x86_64-fslcsdk-linux/usr/bin/arm-fslc-linux-gnueabi/arm-fslc-linux-gnueabi-gcc



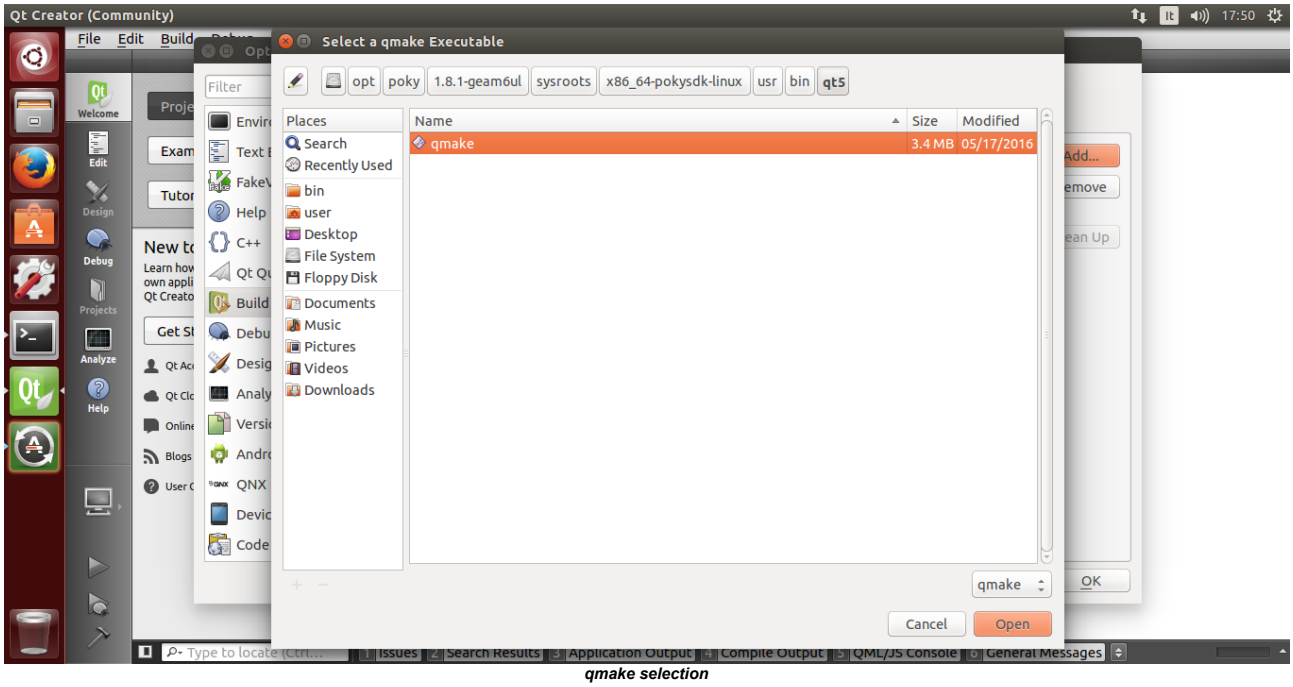
compiler arm setup



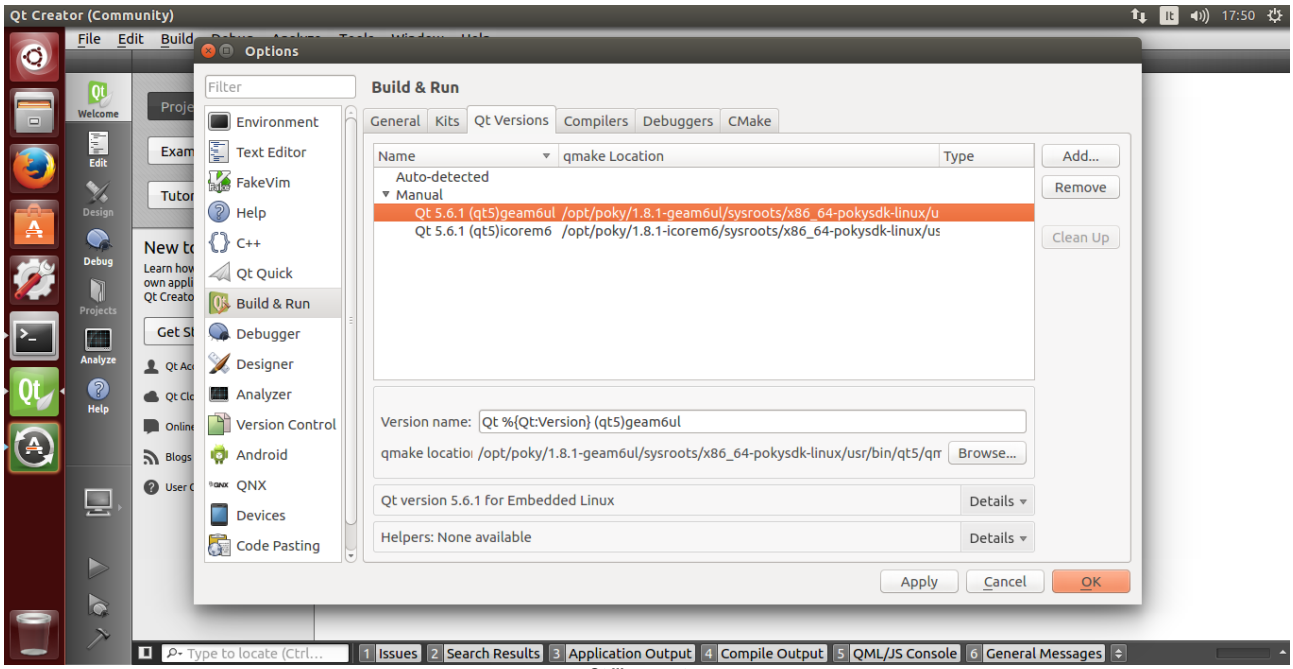
gdb client setup

Setting the Qt libraries finding the executables file qmake included in *Yocto*'s sysroot and indicate it to the Qt creator, now all the Qt libraries are linked, e.g. inside the version 2.3 the qmake is included in the path:

/opt/fslc-framebuffer/2.3_armv7/sysroots/x86_64-fslcsdk-linux/usr/bin/qt5/qmake

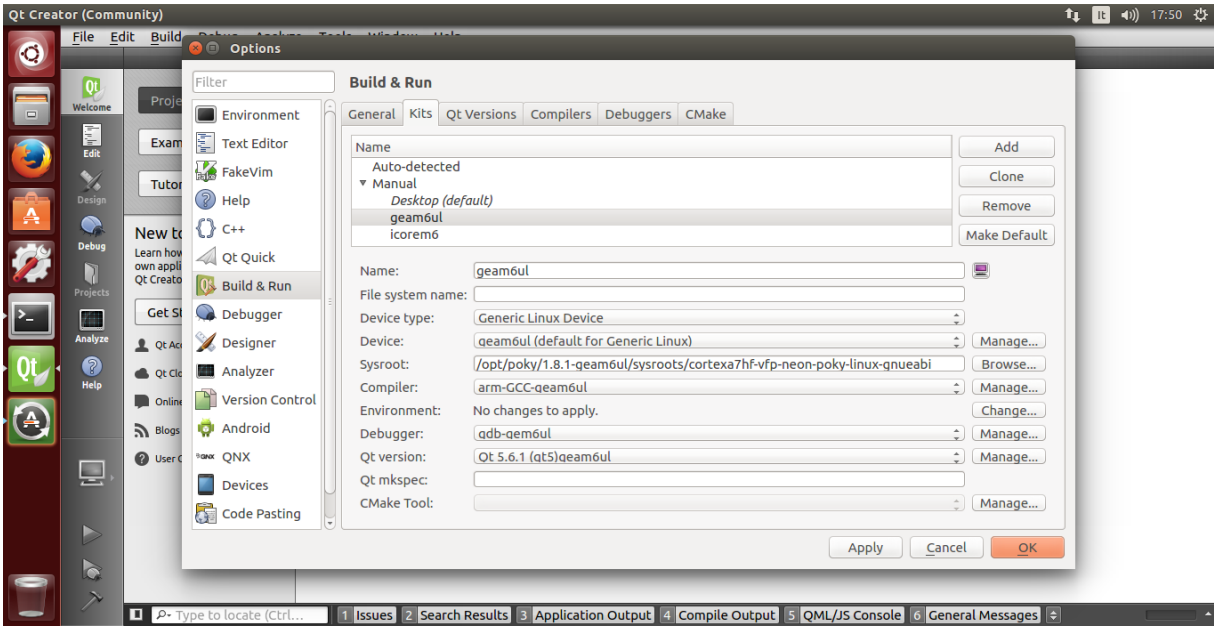


qmake selection



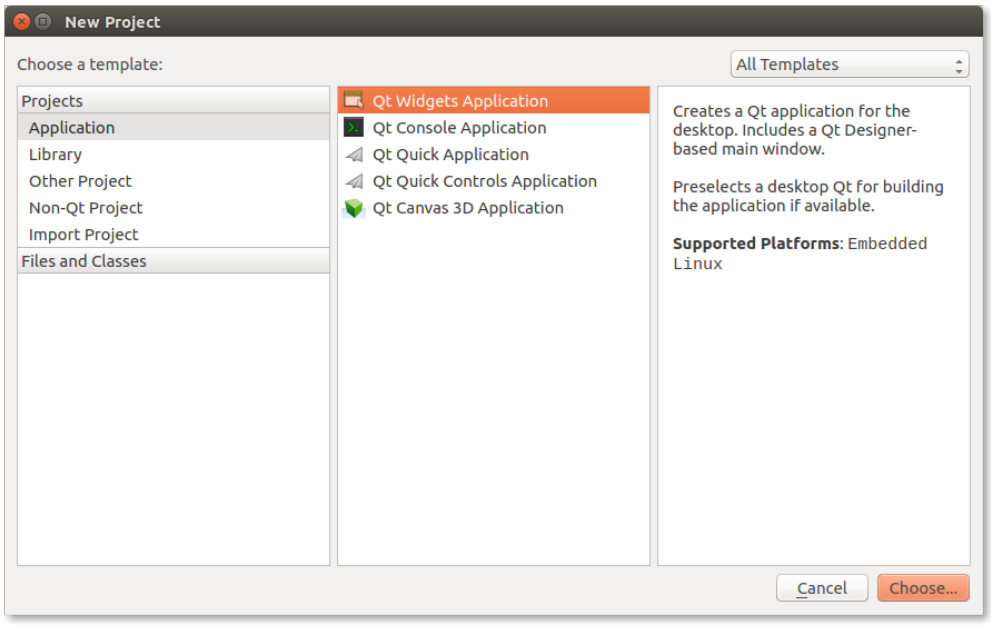
Qt library setup

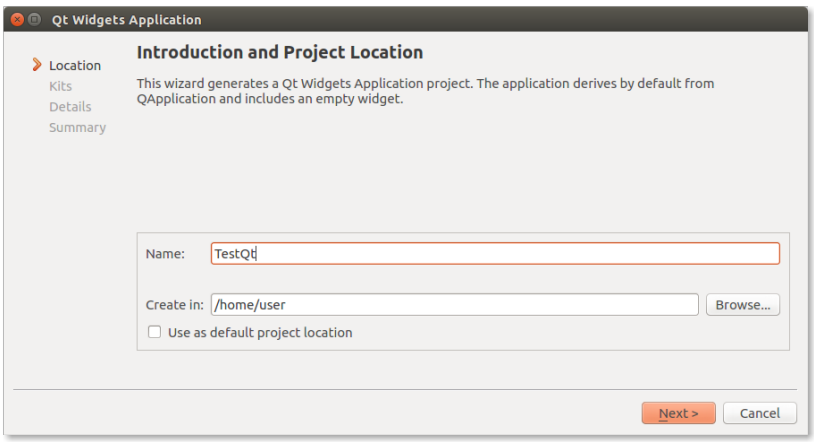
Now setup the compiling kit for arm-gcc and the Qt libraries for the target as shown in the image below:

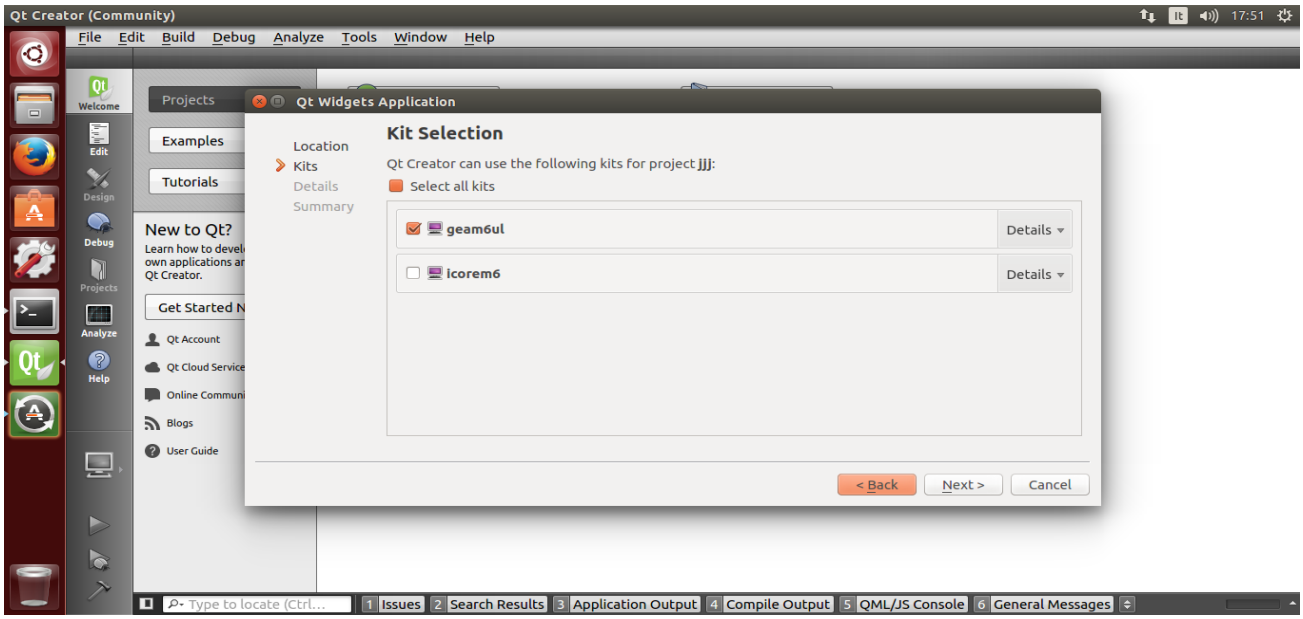


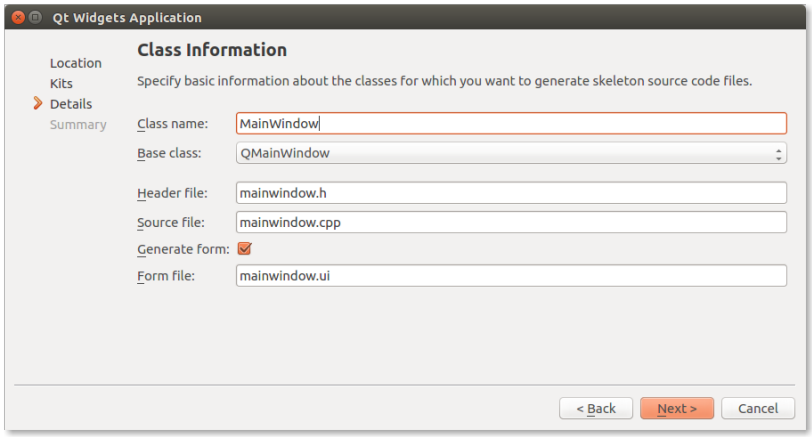
15.4 Create a new Qt project

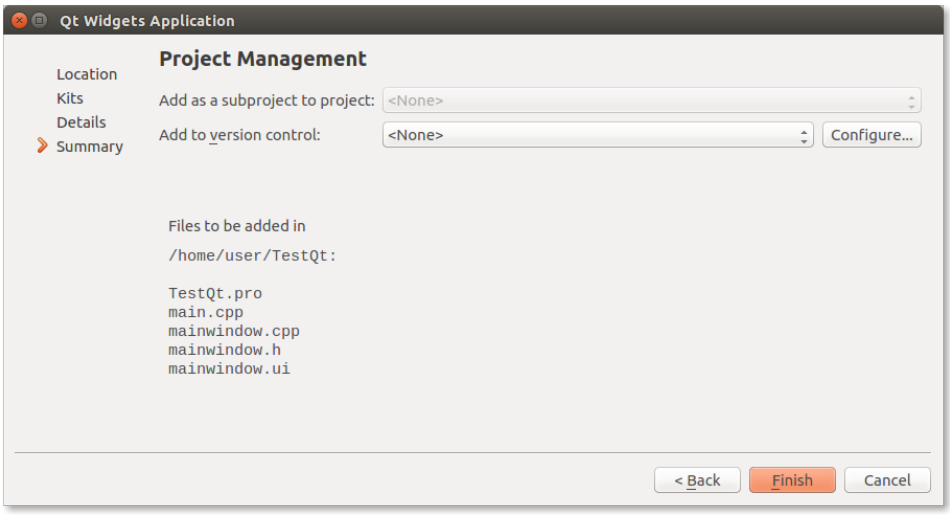
To create a new project use an automatic wizard and select the own compiling kit (see the previous chapter to configure it). Once the project is done run the commands shown in the following chapter to setup the deploy on target.




Qt Widgets Application
Location
Kits
Details
Summary
Introduction and Project Location
This wizard generates a Qt Widgets Application project. The application derives by default from QApplication and includes an empty widget.
Name:
TestQt
Create in:
/home/user
Browse...
Use as default project location
Next >
Cancel


Qt Creator (Community)
File Edit Build Debug Analyze Tools Window Help
Welcome
Edit
Design
Debug
Projects
Analyze
Help
Projects
Examples
Tutorials
New to Qt?
Get Started N
Qt Account
Qt Cloud Service
Online Communi
Blogs
User Guide
Qt Widgets Application
Location
Kits
Details
Summary
Kit Selection
Qt Creator can use the following kits for project JJJ:
Select all kits
geam6ul
Details
icorem6
Details
< Back
Next >
Cancel
Type to locate (Ctrl...
1 Issues
2 Search Results
3 Application Output
4 Compile Output
5 QML/JS Console
6 General Messages


Qt Widgets Application
Location
Kits
Details
Summary
Class Information
Specify basic information about the classes for which you want to generate skeleton source code files.
Class name:
MainWindow
Base class:
QMainWindow
Header file:
mainwindow.h
Source file:
mainwindow.cpp
Generate form:
Form file:
mainwindow.ui
< Back
Next >
Cancel



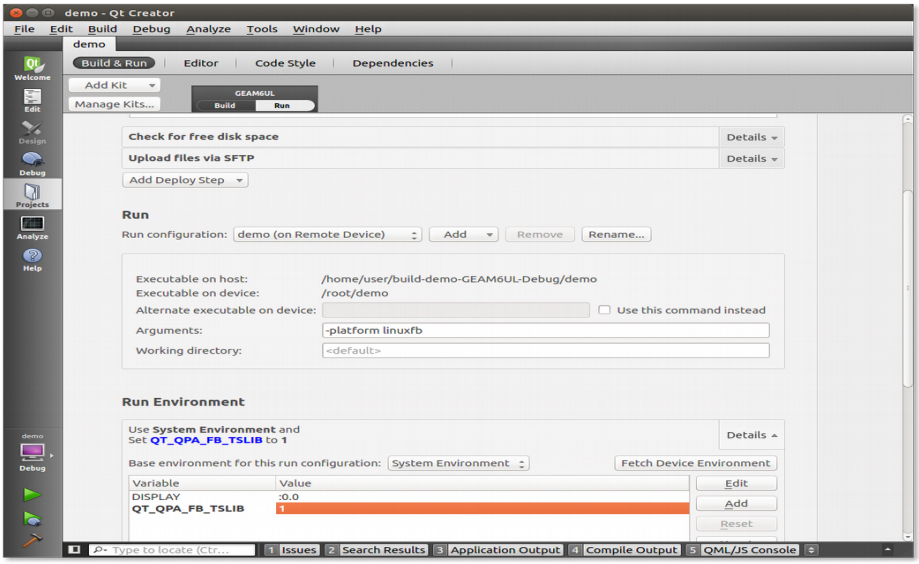
15.5 Qt deploy on target

It's possible to create an own application and deploy it directly on the target. Open the .pro file and add the target directory where the binary file will be deployed on the target. Before starting to deploy check the network connection from the virtual machine to the target.
For widget application only it's mandatory to specify the target path in .pro files by the syntax:

```
unix {
        target.path = /root
        INSTALLS += target
}
```

Remember to set the suffix for the execution of the application.

```
-platform linuxfb
```



15.6 Input device options

In order to successfully run a QT application it is necessary that it is correctly attached to the corresponding input device which can be a USB mouse, a touch screen resistive or capacitive touch screen. If you run an application that requires the TSLIB (e.g. the resistive touch screen of our evaluation board), is necessary to set the following environment variables in the session (shell) from which you run the application, this command is used to attach the Tslib to the correct device's touchscreen present on the board. Remember to run the touch screen calibration at the first time you powered the module after the environment variables setting.

```
export TSLIB_TSDEVICE="/dev/input/touchscreen0"
```

WARNING: the **/dev/input/touchscreen#(0 .. 1 .. 2)** represent the various input devices. Depending on the touch screen used, the device on kernel may be different.

15.7 LCD cursor

Using the following command to remove the cursor from LCD, which can disturb during the execution of an application with a graphical interface.

```
echo -e -n '\033[9;0]' > /dev/tty1
```

15.8 LCD standby status

Using the following command to disable the standby mode of the LCD that happens automatically after a few minutes.

```
echo -e '\033[?17;0;0c' > /dev/tty1
```

15.9 LCD hide cursor

For to hide the LCD cursor on the screen

```
echo 0 > /sys/class/graphics/fbcon/cursor_blink
```

```
 6 require conf/machine/include/iCoreM6.inc
 7
 8 SOC_FAMILY = "mx6:mx6dl:mx6s"
 9
10 KERNEL_DEVICETREE = "imx6dl-examples.dtb"
11
12 PREFERRED_PROVIDER_u-boot = "u-boot-imx"
13 PREFERRED_PROVIDER_virtual/bootloader = "u-boot-imx"
14
15 UBOOT_CONFIG[sd] = "mx6s_icore_sd_config,sdcard"
16 UBOOT_CONFIG[nand] = "mx6s_icore_config,ubifs"
17
18 IMAGE_FSTYPES = "tar.bz2 ext3 sdcard ubi"
19 UBINIZE_ARGS = "-m 2048 -p 128KiB -s 2048"
20 MKUBIFS_ARGS = "-m 2048 -e 126976 -c 1944"
21 UBI_VOLNAME = "rootfs"
```

15.10 Using of a recipe

Using the recipes it is possible to manage *Yocto* on how to compile the package. It's also possible to add the recipes inside the own meta-layer finding them using internet or creating it independently. In a meta-layer it's possible to realize the recipes which depends on the type of machine selected in which the customers may add the patch or the code operations resulting from the meta-layer application placed at other levels. This chapter shows how to manage of the recipe of the kernel depending on the machine created following the instruction of the previous chapter. Every recipe is included inside the folder that conventionally have the name of the packet with inside the following file:

- .bbappend a file including the list of the all the operation of file moving and patch application to run to compile a packet
- The folder linux, including the list of file and all the patch which should be applied before of the first compilation
- The folder linux-geam6ul including the file .config and the file patch which will be moved automatically in the application build folder

These steps to customize the kernel recipe:

1. Enter the directory Linux cd meta-example/recipes-kernel/linux.
2. Replace the "example" with the name of the own machine modified as described in the previous chapter.
3. Edit the file .bbappend including the patch to be applied to the kernel:

```
1 FILESEXTRAPATHS_prepend := "${THISDIR}/${PN}:"
2
3 SRC_URI += "file://defconfig                          \
4             file://0001-examples.patch     \
5 "
6
7
```

4. Copy and rename the file .config of the kernel as defconfig. This file includes all the compiling options of the kernel that will be apply during the build as described in section: Kernel config.

15.11 Creating of a new image

It's possible to create the own recipes for custom image generation or to keep on use the image provided by Engicam. The recipes to generate the images will be to include inside the following directory:

/home/user/yocto/pyro/sources/meta-example/recipes-core/images

Inside this directory is include the image example.bb, it may be modified, or a new image may be added at the default one provided by Engicam.

16. Move files on the target

There are several ways to move files from the local machine to the target. In the following chapter some examples are shown, but it's also possible to use an ftp connection or transfer files trough SSH. For using it the related service should be included in the filesystem recipes.

16.1 NFS shared folder

Inside the virtual machine exists a NFS folder /nfs_icore already preconfigured to share the files on network. For further information refer to the Install NFS Server in Ubuntu into the Appendix section, that describes the installation and configuration of a NFS server. To mount a directory available on a NFS server into a target's local directory on Linux use the following command:

```
mount -t nfs -o nfsvers=3,nolock <server IP>:<server directory> <target directory>
```

For example:

```
mount -t nfs -o nfsvers=3,nolock 192.168.2.57:/nfs_icore /mnt
```

Remember that the network must be properly functioning between the target and the virtual machine.
See Install NFS Server in Ubuntu for installing an NFS server into the Engicam virtual machine.

16.2 Example of mounting MMC card

Use the following command to mount the SD Card on the i.Core

```
mount -t vfat /dev/mmcblk1 /mnt
```

16.3 Example of mounting USB Memory

Use the following command to mount the USB Memory

```
mount -t vfat /dev/sda1 /mnt
```

17. Appendix

This section of the manual contains some additional content on the module and how to use some software packages useful for development.

17.1 Use of TFTP

On the module i.CoreM6 is possible to reprogram the kernel via TFTP using the script ker.scr already pre-compiled. This option can be very useful working on the kernel as it speeds up the programming process than using SD card.

To properly configure the module to work via ftp refer to the NAND programming by uboot-tftp about the network settings. Once you have set the system, load the pre-configured ker.scr script:

```
tftp ker.scr
```

and run it from the U-Boot prompt:

```
so
```

17.2 How to use GPIO

It's possible to move a GPIO from kernel space and from user space. In both cases, the pin must be initialized inside the device tree.

17.2.1 IOMUX configuration

Each pin of the processor can be configured and connected to various peripherals. The IOMUX takes care of set each pin on a device.

To find what function may have a particular pin and what values must be set on the IOMUX is necessary to identify through Engicam hardware manual the name given to the PIN from NXP. Once you have located it, you just open the reference manual NXP and in the section IOMUX find the selected pin. As you can see it shows all the functions available for each MUX value.

Register (IOMUXC_SW_PAD_CTL_PAD_NAND_DQS)

SW_PAD_CTL Register

Address: 20E_0000h base + 444h offset = 20E_0444h

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	Reserved															HYS
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	PUS		PUE	PKE	ODE	Reserved			SPEED		DSE			Reserved		SRE
Reset	0	0	0	1	0	0	0	0	1	0	1	1	0	0	0	0

IOMUXC_SW_PAD_CTL_PAD_NAND_DQS field descriptions

Field	Description
31–17 -	This field is reserved. Reserved
16 HYS	Hyst. Enable Field Select one out of next values for pad: NAND_DQS 0 **HYS_0_Hysteresis_Disabled** — Hysteresis Disabled 1 **HYS_1_Hysteresis_Enabled** — Hysteresis Enabled
15–14 PUS	Pull Up / Down Config. Field Select one out of next values for pad: NAND_DQS 00 **PUS_0_100K_Ohm_Pull_Down** — 100K Ohm Pull Down 01 **PUS_1_47K_Ohm_Pull_Up** — 47K Ohm Pull Up 10 **PUS_2_100K_Ohm_Pull_Up** — 100K Ohm Pull Up 11 **PUS_3_22K_Ohm_Pull_Up** — 22K Ohm Pull Up
13 PUE	Pull / Keep Select Field Select one out of next values for pad: NAND_DQS 0 **PUE_0_Keeper** — Keeper 1 **PUE_1_Pull** — Pull
12 PKE	Pull / Keep Enable Field Select one out of next values for pad: NAND_DQS 0 **PKE_0_Pull_Keeper_Disabled** — Pull/Keeper Disabled 1 **PKE_1_Pull_Keeper_Enabled** — Pull/Keeper Enabled

Fig13

To move the GPIO at user space level initialize the pin inside the device tree, selecting the level at which insert them (see Device Tree note) in the following example is been included in:

linux/arch/arm/boot/dts/imx6ul-gea.dts

```
iomuxc {
        pinctrl-names = "default";
        pinctrl-0 = <&pinctrl_hog_1>;
        imx6ul-evk {
                pinctrl_hog_1: hoggrp-1 {
                        fsl,pins = <
                                MX6UL_PAD_NAND_CE1_B__GPIO4_IO14        0x17059 /* WF111 reset */
```

```
                                MX6UL_PAD_UART1_RTS_B__GPIO1_IO19  0x17059 /*CD sdhc1 */
                                MX6UL_PAD_NAND_DQS__GPIO4_IO16 0x17059 /*LED DL2*/
                                >;
        };
```

The pad value is found inside the file:

```
linux/arch/arm/boot/dts/imx6ul-pinfunc.h
```

The hexadecimal value readable after the pad's "define" specifies the pad's options. Refer to the manual imx6 for the values that these registers may assume.
Refer to the iMx6 reference manual for the values that these registers may assume.
Using the following function to set value and direction of gpio directly in user space :

```
gpio_export {
        compatible = "gpio-export";
        #size-cells = <0>;
        GPIO_DL2 {
            gpio-export,name = "GPIO_DL2";
            gpio-export,output = <1>;
            gpio-export,direction_may_change;
            gpios = <&gpio4 16 0>;
        };
};
```

for more details check the file "Documentation/devicetree/bindings/gpio/gpio.txt" in the kernel source.

17.2.2 Use GPIO from user space

After the user has configured the IOMUX and loaded the modified Kernel you need to add the support for sys/class/gpio. Enter in the configuration tool of the Kernel and set:

```
Device Drivers --->
   [*] /sys/class/gpio... (sysfs interface)
```

Now to calculate the number of GPIO to move using the following formula:

```
GPIO1_3, which corresponds to GPIO number [(1-1)*32 + 3] = 3.
```

Then set the GPIO as output and move it

```
echo 3  > /sys/class/gpio/export
echo out  > /sys/class/gpio/gpio3/direction
echo 1  > /sys/class/gpio/gpio3/value
echo 0  > /sys/class/gpio/gpio3/value
```

Or set the GPIO as input and read its value

```
echo 3    > /sys/class/gpio/export                Export GPIO1_3 to user space
echo in   >/sys/class/gpio/gpio3/direction        Set the GPIO as input
cat       /sys/class/gpio/gpio3/value             reading value
```

17.3 How to use the CAN BUS

For the use of the CAN Bus on the iCoreM6X modules, you need the iproute and the cantest package on the *Yocto* recipes. These the instructions for enable the can and some example to how to use:

1) Configure the bit rate on target:

/ # ip link set can0 type can bitrate 125000

2) Enable the interface on target:

/ # ifconfig can0 up

3) To send a frame:

/ # cantest can0 5A1#11.2233.44556677.88

4) To receive a frame:

/ # cantest can0

17.4 Customize Kernel splashscreen

This procedure is used to modify the graphic image of the kernel which appears when the machine is started. The image must be converted in .ppm format using GIMP, which is free downloadable program (Engicam recommends the using of windows version).
After created an image with the size of the screen, follow these steps to convert it into a readable format. Note that the images larger than the screen resolution will not be displayed.

- **Create an image of the desired size (e.g. 800x480 for 7" display)**
- **Open the image using GIMP 2.8**
- **Right-click on the image**
  Image->Mode->indexed (if already indexed, move it on RGB)
  Then, put it again on Indexed to open the options window.
- **Select: Generate optimum palette, maximum number of colors 220 and push conversion**
- **Select File-->export as... end export file with extension .ppm**
- **Select as ASCII**
- **Rename the image as "logo_linux_clut224.ppm"**
- **Enter into the folder "kernel-source/drivers/video/logo"**
- **Replace the old file with this one**
- **Delete the file logo_linux_clut224.c logo_linux_clut224.o**

Rebuild the kernel.

Warning: to see the kernel splashscreen is necessary to use the framebuffer console

17.5 Customise filesystem splashscreen

The following steps are used to customize the splashscreen of the filesystem provided by Engicam (the one with pictures of the Engicam's logo and the progress bar)

1. Open the virtual machine and install the following package:

sudo apt-get libgtk2.0-dev

2. Enter into the directory:

yocto/pyro/sources/meta-engicam/recipes-images/images

3. Open the file .bb to be compiled and add the psplash package if it's not already present

4. Copy the script make-image-header.sh into the directory:

yocto/pyro/sources/meta-engicam/recipes-core/psplash/files

5. Into the same directory insert the file .png to display
6. Running the script to create the file .h for displaying the image by *Yocto*:

   ./make-image-header.sh /<PATH>/<IMAGE>.jpg POKY

7. Rebuild the desired image.

17.6 Remove filesystem SplashScreen

1. Enter into the directory containing the images:

   yocto/pyro/sources/meta-engicam/recipes-images/images

2. Open the file .bb to be compiled
3. Remove the psplash package if it's already present, which is the file that displays the splashscreen image
4. Rebuild the desired image

17.7 MAC address

Each ENGICAM module has a serial number uniquely associated with an own MAC Address and regularly reserved. The MAC address is written in U-Boot environment variable:

```
printenv ethaddr
ethaddr=11:22:33:44:55:66
```

This value must be set in the kernel to be used by the network interface.

WARNING:
Remember that, if you delete bootargs parameters, or reprogrammed u-boot (which implies the loss of all parameters of the u-boot) the modules will have the default MAC address and therefore, if you connect them to the same physical network, there will be communication conflicts. Using U-Boot mode you can change the value by editing the environment variable with command set ethaddr. Engicam's modules, used on the starterkit, will already have the MAC address assigned.

17.8 Set up of display's backlight

It's possible to modify the backlight brightness using the following command:

echo x > /sys/class/backlight/backlight/brightness

The x variable has a range gap from 0 to 100. Setting the 0 value the backlight is switched off, the value 100 sets the brightness at the highest rate.

17.9 TSLIB Calibration of touchscreen

For the calibration of touch screen we use the ts_calibration utility included in Tslib packet.

Run the command and execute the instructions on the device's display:

```
./ts_calibrate
```

If you miss the /etc/ts.conf file this is an example of how to edit a new one.

```
# Uncomment if you wish to use the linux input layer event interface
module_raw input grab_events=1

# Uncomment if you're using a Sharp Zaurus SL-5500/SL-5000d
# module_raw collie

# Uncomment if you're using a Sharp Zaurus SL-C700/C750/C760/C860
# module_raw corgi

# Uncomment if you're using a device with a UCB1200/1300/1400 TS interface
# module_raw ucb1x00

# Uncomment if you're using an HP iPaq h3600 or similar
# module_raw h3600

# Uncomment if you're using a Hitachi Webpad
# module_raw mk712

# Uncomment if you're using an IBM Arctic II
# module_raw arctic2

module pthres pmin=1
module variance delta=30
module dejitter delta=100
module linear
```

17.10 Install NFS Server in Ubuntu

For the NFS server installing, create the directory nfs_icore and then following this instruction for enabling the service, root privileges are required.

Create the directory:

```
sudo mkdir /nfs_icore
```

Install the NFS server with the command line:

```
sudo apt-get install nfs-kernel-server
```

Edit the server configuration:

```
sudo gedit /etc/exports
```

Add the line:

```
/nfs_icore *(rw,sync,no_subtree_check,no_root_squash)
```

Restart the server:

```
sudo /etc/init.d/nfs-kernel-server restart
```

18. VMware Player sample user guide

During the first installation of the virtual machine some problems could occur for with the VMPlayer. In this chapter we'll give some recommendations to facilitate this work.
To work properly with the virtual machine you should be able to

- Access the network
- Read and write a SD card
- Copy and move files

18.1 Use of the last VMware Player version

From the player version 5.0 have been were introduced substantial changes and improvements in the detection of devices between the physical machine and the virtual machine. We suggest to use always the last version or a version after the 5.

18.2 Installing the latest version of VMware Tools

Re-install the VMware Tools to the latest version, follow the instructions and once the installation is finished restart the virtual machine

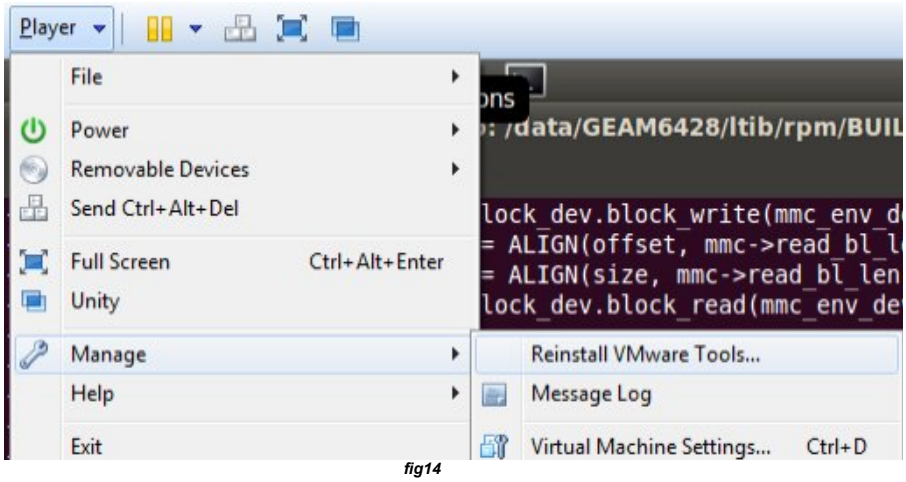


fig14

18.3 Network troubleshooting

The virtual machine can access to the network and communicate with the module, therefore via virtual machine you must be able to:

- ping from the virtual machine to the PC and vice versa
- ping from the virtual machine to another address on the Net
- ping from the virtual machine to the module and vice versa

For problems of communications you can have the following solutions:

- Disable any WiFi device before turning on the virtual machine
- Do not connect the PC to a point to point network. Each time you shut down the connection, you must restart the PC since the virtual network between the physical machine and VMplayer will lose the connection.

- Disable any anti-virus and firewall

For any problems on network, always reboot the PC. The default network configurations is the following:

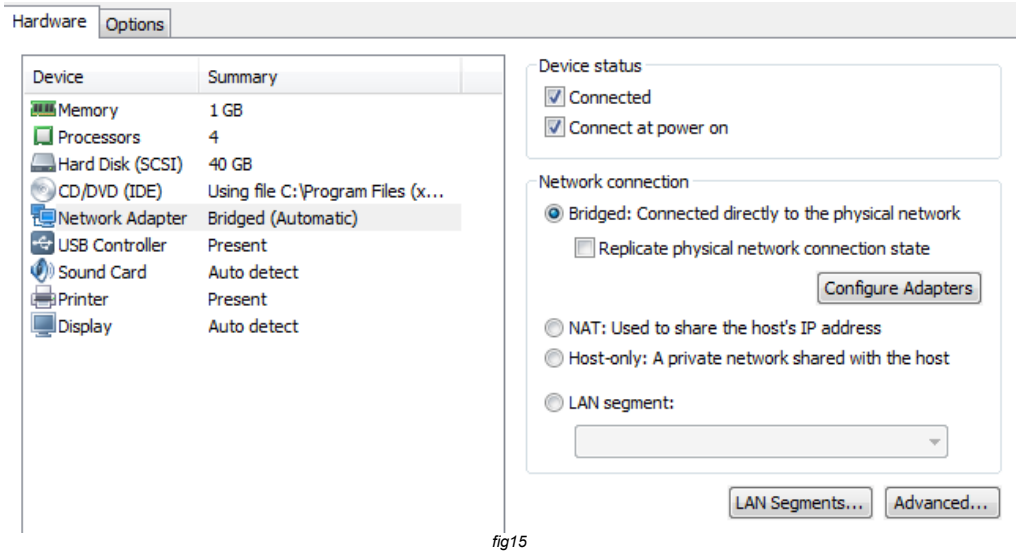


fig15

Set the IP Address manually or via DHCP. Verify the network connection using ifconfig command and check the icon in the top right position as shown in figure below.

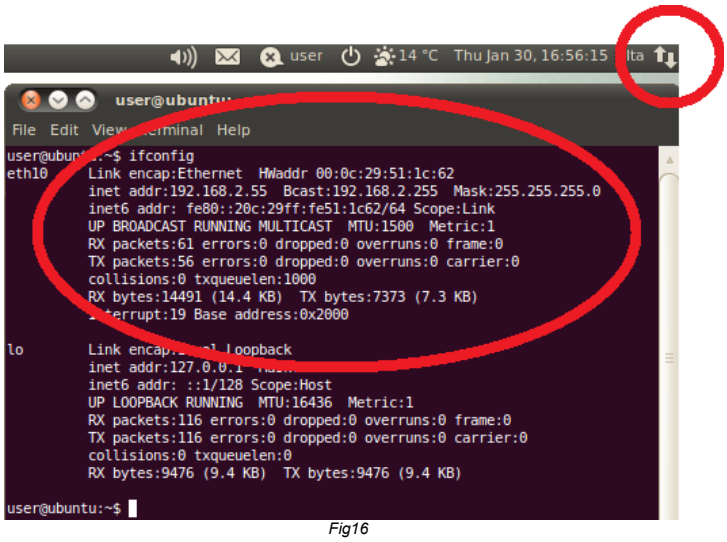


Fig16

18.4 Troubleshooting SD

The write operation of an SD card can create some problems for the correct writing of the files. Sometimes there may be problems to recognize the micro SD to the virtual machine that contains the Linux BSP. In case of trouble, try the following steps.
You may have problems associated with the use of SD card readers on the physical bus of the machine. In case of problems it is recommended to use USB SD readers who are generally more compatible with VMPlayer than those on the bus.

18.4.1 Verify that the physical machine is able to see the driver

Sometimes there may be also physical problems with the SD reader. If you connect the drive to the Windows machine having the virtual machine closed and the card is formatted in ext3 you should see the following message:

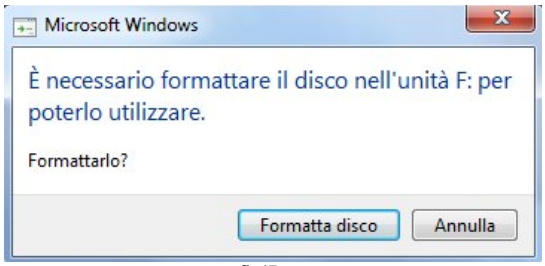


fig17

If the message does not appear, there may be hardware problems on the SD reader, or driver problems with Windows. If the SD card is formatted as FAT32 or with another type of Windows capable filesystem, it 'll be open a normal Windows' folder.

18.4.2 VMPlayer status icons

If the USB driver is not recognized correctly appear on the player VMPlayer the following icons:

- External disk Icon
- Driver connect: if connected will be flagged otherwise press connect

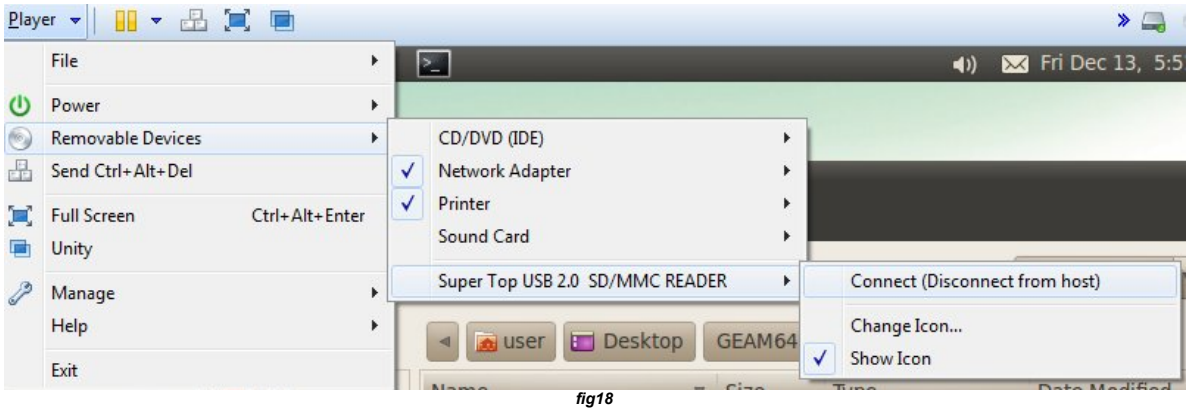


fig18

18.4.3 Verify the device presence on the virtual machine

If the device is properly connected you will see the resource available and access to its content by navigating into it through the shell by entering the path /media/rootfs

fig19

In the event that there is status icon of the VMPlayer, but you cannot access the disk of the virtual machine. Restart the virtual machine and the physical machine and try again. It is recommended to connect the SD card when you have the focus on the virtual machine.

18.4.4 Safe disconnection of the SD card

Once the use of the SD card is ended is recommended to ALWAYS give the sync command to synchronize the files and then safely remove the card using the **umount** command form shell or clicking the button **eject** which is also seen in the previous figure.

18.4.5 General note

Sometimes it may also happen that, with the continued use, the wrong removals, especially when used on the target, the SD card may be damaged and no more readable. In these cases, it's only the partition that was damaged and may need to reformat the partition, following the procedure described in this manual.

18.5 Copy files between the virtual machine and the physical machine

You should be able to copy and move files from the virtual machine using copy and paste or using drag-and-drop mode from the virtual desktop to the real one. If it is not possible to perform all these operations, it depends on the correct installation of VMware Tools.

19. Technical support

Engicam provides to its customers the latest versions of its BSP of development. In the time between one version and the other a few changes or improvements can be published and shared via patch or textual documents.

19.1 Upgrading the BSP using patch

Once started the development and customization of the kernel, it's possible to keep it up to date through the use of patches that are on your account. This chapter gives some information on how to manage and update the BSP. Once achieved a stable version of the system it is advisable to only apply patches that may affect the own application.

19.1.1 Structure of the patch folder

This folder contains any patches of the downloadable version. The patches have the following order:

PATCH_*KERNEL/BOOT*_BSP_*X.X*

In the name it's specified "KERNEL or BOOT", where the patch must be applied and the BSP revision "X.X"

Inside the folder you find the main patches that have the following method:

***00X*_main_patch_*YYMMDD*.patch**

main patch at the date YYMMDD that aligns the whole kernel with our mainline.

Then you can find the singular patch with the following method:

***00X.0Y_argument_name*.patch**

where "00X" specified the membership to the main patch, "0Y" is an incremental number, the "argument_name" is the name of the fixed problem by the patch.

These Patches solves the individual problem and are usually used by the customer who has already customized the kernel so that is not required having to apply the main patch.

19.1.2 Patch structure

A patch consists of several sections of code to add or remove, these sections are localized reporting the previous and subsequent lines of code where to edit the changes.

```
1 diff --git a/arch/arm/mach-mx6/Kconfig b/arch/arm/mach-mx6/Kconfig
2 index 0c6e89e..db1a58b 100644
3 --- a/arch/arm/mach-mx6/Kconfig
4 +++ b/arch/arm/mach-mx6/Kconfig
5 @@ -222,6 +222,14 @@ config MACH_MX6Q_MINIMUM_FREQ400
6           This features set the minimum CPU clock frequency to 400 Mhz instead of 200 Mhz.
7           Recommended option for the use of video codecs.
8
9 +config MACH_MX6Q_ICORE_OPENFRAME_RESISTIVE
0 +       bool "Support i.CoreM6 resistive OpenFrame"
1 +       depends on MACH_MX6Q_ICORE
2 +       help
3 +         Include the support for Engicam openframe. This features enabled
4 +         the correct power up and power down of the on-board LVDS controller.
5 +          This features is mandatory.
6 +
7  config MACH_MX6Q_ICORE_STARTERKIT_CAP_EDT
8         bool "Support i.Core capacitive starterkit"
9         depends on MACH_MX6Q_ICORE
```

fig20

Diff –- indicates the file and the location you want to edit.

@ indicates the code lines where edit the modifies.

Following are shown the lines of code that should not be changed.
With **+** or **–** are shown the lines to add or remove to edit the file. The remaining code is used to identify where to apply the patch

19.1.3 How to apply the patch

A patch is applied if the command patch can correctly identify where to insert the code parts. Otherwise, it generates an error file and the patch must be changed manually.
In case you have customized the code, you can follow the structure of the patch and apply it manually to avoid errors in editing. Refer to the previous chapter for a description of the structure of the patch. To apply a patch follow the below procedures. Enter the folder you want to edit (U-Boot or kernel). Copy the patch inside the folder:

```
cd linux
```

with the dry-run command you can try to apply a patch and see if it returns an error to evaluate whether it is possible to apply the patch.

```
patch -p1 --dry-run < 002.02_iCoreM6_openframe_lvds.patch
```

Once you have tested the application, if there are not too many mistakes, you may apply the patch using the command:

```
patch -p1 < 002.02_iCoreM6_openframe_lvds.patch
```

If the application is successful, rebuild the kernel and update it on the device. In the case of errors you will have an output like this:

```
patching file arch/arm/mach-mx6/Kconfig
Hunk #1 FAILED at 222.
1 out of 1 hunk FAILED -- saving rejects to file arch/arm/mach-mx6/Kconfig.rej
patching file arch/arm/mach-mx6/board-mx6q_icore.c
Hunk #1 FAILED at 107.
Hunk #2 succeeded at 264 (offset -6 lines).
Hunk #3 succeeded at 417 (offset -8 lines).
Hunk #4 succeeded at 1427 with fuzz 2 (offset -19 lines).
Hunk #5 succeeded at 1629 (offset -33 lines).
1 out of 5 hunks FAILED -- saving rejects to file arch/arm/mach-mx6/board-mx6q_icore.c.rej
```

In the above example the patch failed the application of the Kconfig file and board-mx6q_icore.c.
The failed parts are available in the file .rej that are generated by the patch command. Then open the file and manually enter the parts not included.

20. Technical support contact

For help, write an email to:

support@engicam.com

21. Useful links

http://www.imxdev.org/

http://www.imxcommunity.org/

http://www.nxp.com/

http://www.engicam.com/

http://layers.openembedded.org

https://www.yoctoproject.org/

https://community.nxp.com/