

DESIGNING WITH FREESCALE

LINUX ON THE I.MX6 (1 DAY)

LAB BOOK



Copyright and licenses



Attribution – ShareAlike 2.5

You are free:

- To copy, distribute, display, and perform the work
- To make derivative works
- To make commercial use of the work

Under the following conditions:



Attribution: You must give the original author credit.



Share Alike: If you alter, transform, or build upon this work, you may distribute the resulting work only under a license identical to this one.

For any reuse or distribution, you must make clear to others the license terms of this work. Any of these conditions can be waived if you get permission from the copyright holder. Your fair use and other rights are in no way affected by the above.

License text: <http://creativecommons.org/licenses/by-sa/2.5/legalcode>

Freescale Semiconductor. Inc © Copyright 2012, based on original work by:
Adeneo Embedded © Copyright 2012, <http://www.adeneo-embedded.com>

Freescale and the Freescale logo are trademarks of Freescale Semiconductor, Inc., Reg. U.S. Pat. & Tm. Off.

ARM is a registered trademark of ARM Limited. Cortex-A9 and ARMv7 are trademarks of ARM Limited. All other product or service names are the property of their respective owners.

Introduction

Lab equipment

These labs use the following equipment:

- i.MX6 SABRE Lite board + LCD screen + Power supply
- USB SD Card reader
- USB to serial adapter
- Micro SD Card + adapter
- Micro USB cable
- Ethernet RJ45 cable

Please leave all the equipment on your table when you leave the training. The instructor and proctors will take care of putting everything back into their boxes.

Setting up the development environment

The development environment has been prepared following the instructions detailed in a separate document named “*Preparing the development environment*”. Please refer to this document if you need to recreate that setup for yourself. Also note that the procedure might change slightly if you decide to use different revisions of the software packages that have been installed (e.g. Ubuntu, LTIB, etc...).

Going further

- This training features more labs than what students can typically achieve. The sections marked as “*Going further*” are optional, but do not hesitate to go through them if you finish the labs in advance.
- Additional labs are available from the 4 day Linux training. You can go through them using the same development environment.

Tips

- The labs contain a lot of step-by-step instructions. The easiest way to go through them is to create a text file where you copy and paste the different command lines that you have been using. This will save you time when repeating those steps and also avoid typos and give you the opportunity to fix issues with new lines.
- Do not hesitate to ask the instructor if you are blocked or if you think that there is a mistake in the instructions.
- The password for all super-user operations is ‘*trainee*’.



- In the lab instructions, ‘>’ refers to the command-line interface of your host, while ‘mx6#’ or ‘#’ refer to the target shell (using the serial port).
- Only use super-user rights (“*sudo <command>*”) when instructed, typically when flashing the boot media.
- Pay attention to the error messages that you might get. They will probably guide you towards the solution.
- Console programs can usually be killed using CTRL-C.
- To copy/paste inside the shell, the shortcut is CTRL-SHIFT-C/V.

Virtual machine tips

- Use CTRL-ALT-ENTER to make the VM window full-screen (or toggle back to windowed mode).
- When the VM is full-screen, you can reach the VMware menu by hovering the mouse at the top of the window.
- If you experience issues with USB devices, unplug and plug the device again. Make sure it is connected to the VM as well.

Lab 1: Booting the board with a precompiled image

Objectives

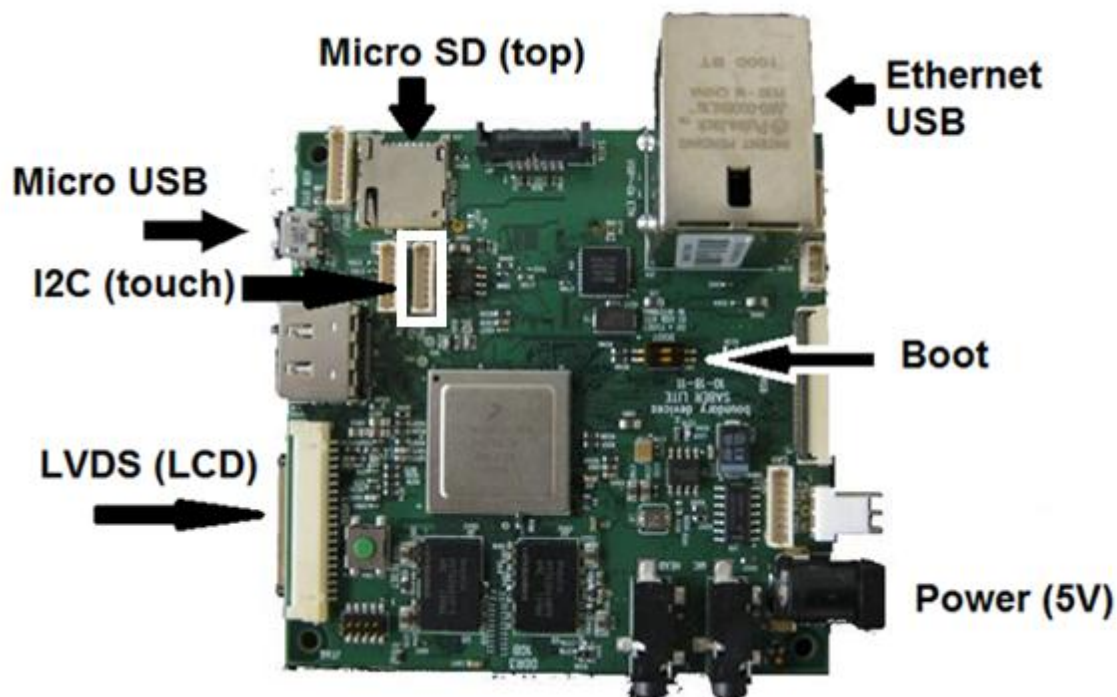
- Update the firmware on the SD Card
- Interact with the board using the serial console
- Use the bootloader to load the kernel

Preparing the SABRE Lite board

Let's connect our development kit.

CAUTION: some connectors are fragile, so please do not use any force when trying to plug the different accessories.

- Plug the LCD screen to the LVDS port and the touchscreen cable to the GPIO/I2C port (see picture)



- Connect your board to your computer:
 - Ethernet RJ45 cable (directly connected to the computer)
 - Serial cable (connected to the USB to serial cable)
- To boot the board, you will need to use the micro SD slot, on the top side of the board.
We will not use the SD Card slot for these labs (bottom side of the board, standard size).

Installing the “shim”

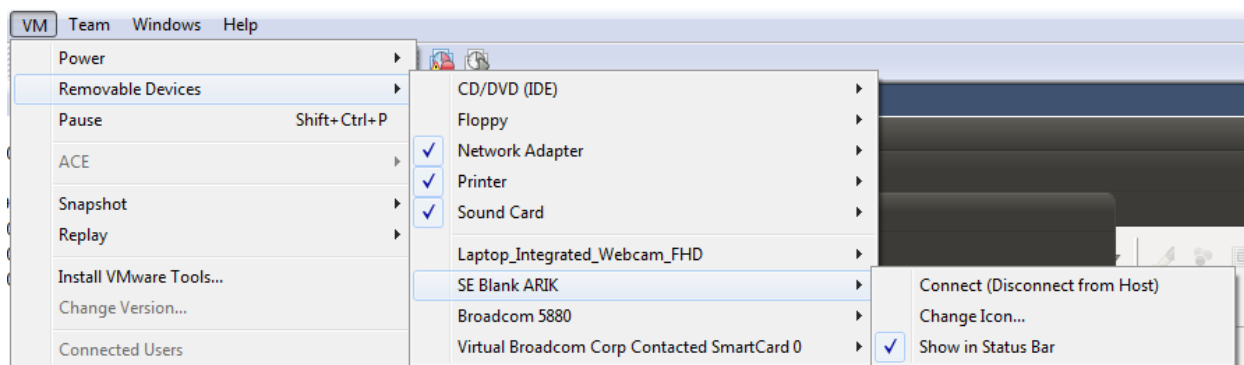
The SABRE Lite board is set to boot from NOR by default (unless a separate connector is used). In order to boot from the SD Card, we need to install a special program in NOR – called a “shim” – that will instruct the i.MX6 boot program to find the bootloader on the SD Card. This procedure typically only needs to be done once for a given board.


We will use the USB loader program and a customized version of U-Boot that will flash everything automatically.

- Set the boot pins to 01 to boot from USB. If you look at the top of the board (when the Ethernet is at the bottom and the audio connectors at the top), this means that the top switch is to the left and the bottom one to the right.



- Make sure that there is not micro SD Card in your board.
- Connect the micro USB cable of your board to your PC (**Caution: the connector is fragile!**)
- Turn on the board.
- Your PC should detect a new USB device. Use the VM menu and connect the “*SE Blank ARIK device*” (**Note:** it might also be named “*Freescale Input Device*”). The i.MX6 device might also appear as a HID device



- In Linux (inside the VM), open a terminal (using the icon  in the quick launch bar) and type the following command:

```
> lsusb
```

You should see something like:

```
Bus 002 Device 002: ID 0e0f:0002 VMware, Inc. Virtual USB Hub
Bus 002 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 001 Device 002: ID 15a2:0054 Freescale Semiconductor, Inc.
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

If you cannot connect the device, do not hesitate to let the instructor know.

- Extract the USB loader program:

```
> cd ~/training_mx6_linux
> cd tools
> tar -xzf ../download/imx_usb_loader.tar.gz
> cd imx_usb_loader
```

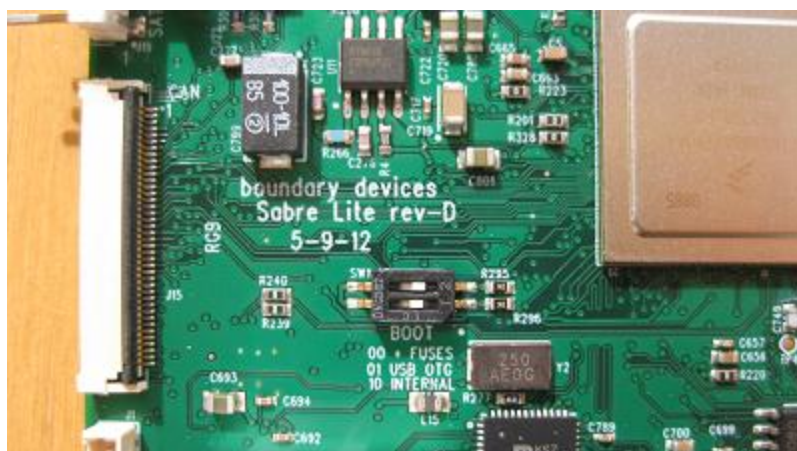
- Start the USB loader:

```
> sudo ./imx_usb
```

Wait 30 seconds (if you have the serial configured, you will see what happens).

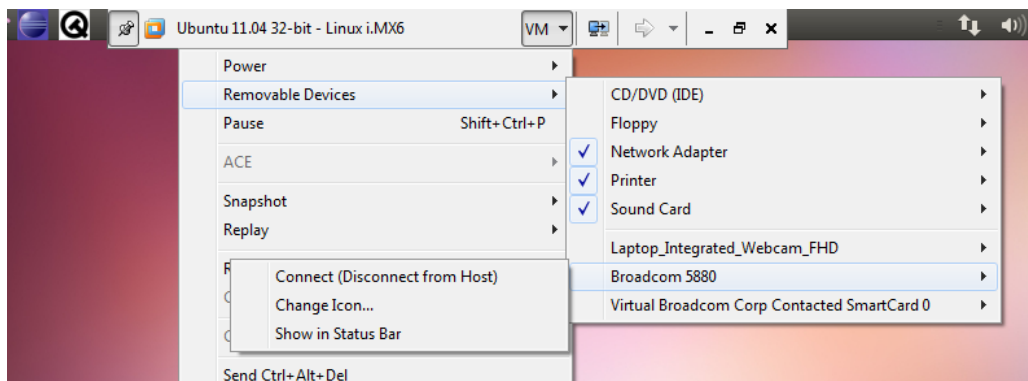
It will load our customized version of U-Boot to the board, along with the shim. U-Boot will be executed and will in turn load the shim from memory and flash it to NOR.

- Unplug the power and the USB from the board to power it off entirely.
- Set the boot pins to back to 00 to boot according to the fuses setting. If you look at the top of the board (when the Ethernet is at the bottom and the audio connectors at the top), this means that both switches are to the right. This will set the board to boot from fuses. Other modes are documented on the PCB itself, but they shall not be used for this training.



Flashing the SD Card

- Connect your SD Card reader to the PC.
- Since you are using a virtual machine, you may need to connect the USB device to the Linux guest system. To do so, use the VMware menu (hover your mouse to the top of the window), select to "VM/Removable Devices/[SD CARD READER NAME]" (look for "Mass storage") and ensure that it is checked. If it is not, select "Connect (Disconnect from host)". The device will be accessible from Linux (inside the VM), but not from Windows.



- Insert the SD Card into the card reader of your PC.
- To see which device the card is mounted as, run:


```
> dmesg
```

The latest message should indicate:

```
[467399.461149] sd 10:0:0:3: [sdc] 3854336 512-byte logical blocks: (1.97
GB/1.83 GiB)
[467399.462514] sd 10:0:0:3: [sdc] Assuming drive cache: write through
[467399.465888] sd 10:0:0:3: [sdc] Assuming drive cache: write through
[467399.465893] sdc: sdc1
```

This tells you that the block device is `/dev/sdc` (might change on your own setup).

- Make sure that you have correctly identified your SD Card reader. Further operations will erase the drive entirely!

- Open a terminal (using the icon  in the quick launch bar) and navigate to `~/training_mx6_linux/scripts/`:

```
> cd ~/training_mx6_linux/scripts/
```

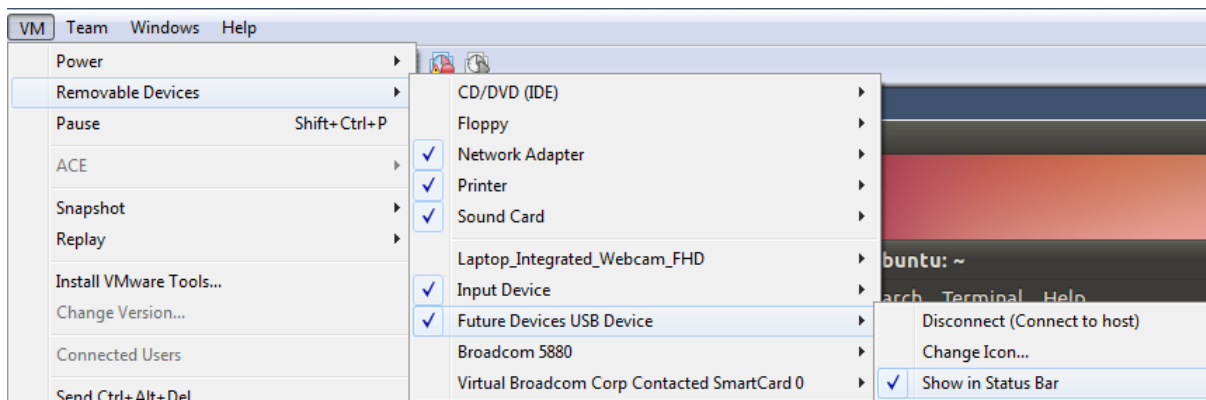
- Run the flashing script with your SD Card device location. Make sure to use the actual SD Card device node (e.g. `/dev/sdc` in our previous example):


```
> ./flash_linux_prebuilt.sh base /dev/[SD CARD DEVICE]
```

Note: For this first lab, we use prebuilt binaries for bootloader, kernel and rootfs.

Setting up the serial console

- Connect your USB-to-Serial adapter to the Host PC.
 - Since you are using a virtual machine, you may need to connect the USB device to the guest system.
To do so, navigate to "VM/Removable Devices/Future Devices USB Device" (the name might differ on your setup, depending on the adapter, e.g. it could also be "Belkin Components") and ensure that it is checked. If it is not, select "Connect (Disconnect from host)". The device will be accessible from Linux (inside the VM), but not from Windows.



- Run and configure minicom:

```
> sudo minicom -s
```

Note: you only need super-user rights when configuring minicom. This only needs to be done once, so make sure you only use minicom with regular rights afterwards.

- Minicom start with its configuration menu (can be invoked later using CTRL-A, followed by O).
- Choose "Serial port setup" and set the value to `"/dev/ttyUSB0"` (since you are using a USB to serial device).
Also set the *Bps/Par/Bits* to 115200 8N1 and disable both the Hardware and Software flow control.

```

+-----+
| A -   Serial Device       : /dev/ttyUSB0
| B - Lockfile Location    : /var/lock
| C -   Callin Program      :
| D -   Callout Program     :
| E -   Bps/Par/Bits        : 115200 8N1
| F - Hardware Flow Control : No
| G - Software Flow Control : No
|
| Change which setting?
+-----+

```

- Validate the settings and save the configuration: in the main menu, choose “*Save setup as df*”.
- Exit the main menu. You will see the main window of *minicom*, where you can actually interact with the device.
- Press Ctrl+A then Q to exit *minicom* and select ‘yes’ to exit without reset.

Booting with the sample root filesystem

Note: The file `~/training_mx6_linux/files/uboot.env` contains all uboot commands for each by lab. Do not hesitate to use them if you got lost.

- Make sure you have closed the previous *minicom* window.
- Start *minicom* again, using standard rights:

```
> minicom -D /dev/ttyUSB0 -o -w
```

Note: “-w” enables line wrapping while “-o” prevents *minicom* from sending useless AT commands once connected.

Make sure to remember that command-line as it is an essential tool for the rest of the labs (and development in general).

- Insert the SD Card that you have flashed into the top micro SD slot of the board (SD4 on the SABRE Lite).
- Turn on or reboot the board. You should see traces on the serial port.
- In the serial terminal press **SPACEBAR** to halt the bootloader.
- In U-Boot, clear the previous environment settings (saved in NOR):

```
U-Boot> destroyenv
```

This will clear the user environment and use the default one (by erasing the region where the environment is permanently stored).

- In U-Boot, specify the root filesystem location by changing the kernel command-line parameters. Issue the following commands to set the corresponding environment variables:

```
U-Boot> setenv loadaddr 0x10800000

U-Boot> setenv bootargs_lcd 'video=mxcfb0:dev=ldb,LDB-XGA,if=RGB666'

U-Boot> setenv bootargs_base 'setenv bootargs console=ttymx1,115200
${bootargs_lcd}'

U-Boot> setenv bootargs_mmc 'setenv bootargs ${bootargs} root=/dev/mmcblk0p1
rootwait rw'

U-Boot> setenv bootcmd_mmc 'run bootargs_base bootargs_mmc; mmc dev 1; mmc
read ${loadaddr} 0x800 0x2000; bootm'

U-Boot> setenv bootcmd 'run bootcmd_mmc'
```

- **Explanations:**

setenv loadaddr 0x10800000	Address (in RAM) where the kernel image will be copied
setenv bootargs_lcd [...]	Kernel command-line arguments to enable the LCD LVDS display
setenv bootargs_base [...]	Base kernel command-line arguments. Debug console assigned to the proper debug port
setenv bootargs_mmc [...]	Kernel command-line argument to mount the root filesystem from the first partition of the SD Card (top slot)
setenv bootcmd_mmc	Instructs the bootloader to load the kernel image from the SD Card (top slot) sector 0x800, length 0x2000. <i>bootm</i> : jump to the image at <i>\${loadaddr}</i>
setenv bootcmd [...]	Sets the default bootloader command (executed automatically)

- Save the environment for subsequent reboots (it is saved in NOR for SABRE Lite by default. This setting can be changed in the U-Boot configuration header file for the board).

```
U-Boot> saveenv
```

- Since you have saved the environment, you will not need to type it every time you boot the board (it is stored on the SD Card).

Note: remember to look at `~/training_mx6_linux/files/uboot.env` if you need to need to see the whole set of commands for each lab.

- Boot the board:

```
U-Boot> run bootcmd
```



Note: this command can be abbreviated to '*boot*'.

- The board should boot and you will be greeted with a prompt. Login as 'root' with an empty password.

Lab 2: Building LTIB

Objectives

- Configuring LTIB
- Building a bootloader, kernel and root filesystem for the board
- Customizing the contents of the root filesystem
- Adding a package to LTIB

Installing LTIB

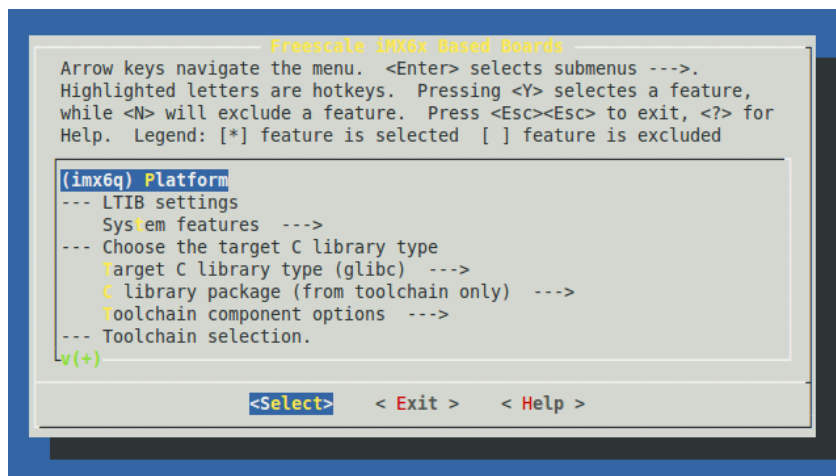
- In order to save time, LTIB has been preinstalled in the virtual machine. The installation steps are documented in “*Preparing the development environment*”.
- The i.MX6 platform has been selected by default (this can be changed later using `./ltib –selectype`)

Building the filesystem for the board

- Start the LTIB configuration menu:

```
> cd ~/training_mx6_linux/ltib
> ./ltib -c
```

- You will reach the main configuration screen of LTIB:



- Make sure to remember how to get to this menu as this an essential step for development on i.MX platforms.
- Using the menu, choose your board for u-boot: `mx6q_sabrelite`
- Check that the `Linux 3.0.15-imx` kernel has been selected.
- Review the toolchain settings:

- Target C library type: *glibc*
- C library package: *from toolchain only*
- Toolchain component options:
 - libc shared libraries
 - C++ shared libraries
 - libgcc*.so*
- Toolchain: *ARM, gcc-4.6.2, multilib, neon optimized, gnueabi/eglibc2.13*
- Toolchain command-line parameters: *-O2 -march=armv7-a -mfpv=vfpv3 -mfloat-abi=softfp*
 - **Note:** this is where you can change the optimization settings for the toolchain.
- Go the *Target Image Generation* options and increase the *tmpfs* size to “8192k”.
- Exit and save the configuration. LTIB will build the packages specified in the configuration.
- Once the build has completed, check that you have the following:

[LTIB]/rootfs/	Root filesystem (for NFS or direct copy to the SD Card). For production, some additional files need to be removed (e.g. header files and other files not used at run-time).
[LTIB]/rootfs/boot/u-boot.bin	Bootloader image
[LTIB]/rootfs/boot/ulmage	Linux kernel image for U-Boot

We will flash these images using a similar method as the previous lab.

Note: To speed up the build process, LTIB automatically makes use of multi-core CPUs and *ccache*. This can be modified in the *.ltibrc* file (at the root of LTIB's install directory). Open [LTIB]/*.ltibrc* to see the different settings that can be changed (you can use *cat .ltibrc* for that).

- Insert the SD card into your USB card reader. Use the flashing script (it will look for the right images). Remember to use the right device for the SD Card reader:

```
> cd ~/training_mx6_linux/scripts
> ./flash_linux.sh /dev/[YOUR SD CARD READER]
```

- Insert the micro SD card into the board and boot the board.
Note: As you had already saved the environment earlier, there is no need to interrupt U-Boot to modify the boot settings.

Customizing the root filesystem

- Go back to the configuration of LTIB:

```
> cd ~/training_mx6_linux/ltib
> ./ltib -c
```

- From the Package list (towards the bottom of the main menu), select the following options:

- *Configure BusyBox at build time*
- *strace*

Note: the packages have actually been selected when preparing the virtual machine for the training.

- Exit and save. LTIB will build the selected packages.
- During the build, another option menu will appear after a while. This is the configuration menu of *BusyBox*.
 - Take some time to browse through the menus. You will see all the features of *BusyBox*.
 - Verify that *BusyBox* will not be compiled as a static binary:
 - In the menu *Busybox Settings > Build options*.
 - Browse the menus to see the different programs that *BusyBox* implements.
 - Enable the lightweight HTTP server from *Networking utilities > httpd*. We will set up the network in the next labs.
 - Exit from the menu and save the configuration. *BusyBox* will now be compiled and deployed to the root filesystem.
- Following the instructions provided in the previous section, flash your images to the SD Card and test your system:

```
> cd ~/training_mx6_linux/scripts
> ./flash_linux.sh /dev/[YOUR SD CARD READER]
```

- Insert the micro SD card into the board and boot the board.
The login and password are now '*root*/'*root*' (the settings have been changed when preparing the VM).



Going further: Studying LTIB's spec files

- Go into the ltib directory

```
> cd ~/training_mx6_linux/ltib
```

LTIB uses *.spec* files to describe packages. These files are located in *[LTIB]dist/lfs-5.1/*

```
> ls dist/lfs-5.1
```

- Let's open a simple *.spec* file to understand its structure:

```
> gedit dist/lfs-5.1/less/less.spec
```

- The first lines describe some basic information about the package:

```
Summary      : text file browser, like more, but you can go back too.
Name         : less
Version      : 381
Release      : 1
License      : GPL
Vendor       : Freescale
Packager     : Stuart Hughes
Group        : Development/Tools
Source       : less-381.tar.gz
BuildRoot    : %[_tmppath]/%{name}
Prefix       : %{pfx}
```

- The % sign is used for internal LTIB commands or access to variables (example:
%define pfx /opt/freescale/rootfs/%{_target_cpu})
- Declaration of basic rules:

```
%Description
...
%Prep
...
%setup
...
%Build
...
```



```
%Install
```

```
...
```

```
%Clean
```

```
...
```

```
%Files
```

```
...
```

- Each rule contains the command to be executed at this stage. These commands will be called when cross-compiling the package. They essentially map to the typical *configure*, *make*, *make install* steps of open-source packages.

Lab 3: Working with U-Boot

Objectives

- Build U-Boot from the sources
- Boot the board and work with the U-Boot shell
- Download a kernel image using TFTP

Building U-Boot

- LTIB provides a version of U-Boot that is specifically patched to support the i.MX6 reference boards.
By default, LTIB builds U-Boot automatically, but it is usually more convenient to make a separate copy of the source tree when modifying U-Boot. We will use LTIB to extract the sources and create a working copy that we will modify and build on the side.

- **Note:** this procedure can be used for any package that LTIB provides. When modifying a given package, it is typical to work on a separate copy and integrate the changes back to LTIB when the development is done.

- Extract the sources:

```
> cd ~/training_mx6_linux/ltib
> ./ltib -m prep -p u-boot
```

This command instructs LTIB to unpack and to apply the patches for our board. You can have a look at [\[LTIB\]/config/platform/imx/u-boot.spec.in](#) to see how this is implemented.

You will find the sources in `~/training_mx6_linux/ltib/rpm/BUILD/u-boot-2009.08`.

- Copy the sources to a separate directory:

```
> cp -R ~/training_mx6_linux/ltib/rpm/BUILD/u-boot-2009.08
~/training_mx6_linux
```

- Work from your new copy:

```
> cd ~/training_mx6_linux/u-boot-2009.08
```

- Add the cross-compiling toolchain to your PATH environment variable:

```
> export PATH=/opt/freescale/usr/local/gcc-4.6.2-glibc-2.13-linaro-multilib-
2011.12/fsl-linaro-toolchain/bin/:$PATH
```

Note: this only needs to be done once per terminal, every time you open a new terminal. If you forget to set it, you will see errors like “*cannot find gcc*” or your host gcc (for x86) might be used, leading to architecture issues.

- Configure U-Boot for our board:

```
> make CROSS_COMPILE=arm-fsl-linux-gnueabi- mx6q_sabrelite_config
```

- Compile U-Boot:



```
> make CROSS_COMPILE=arm-fsl-linux-gnueabi- -j4
```

This creates the U-Boot image (*u-boot.bin*) at the root of the source directory.

- **Note:** If you have a multi-core CPU, you can speed up the compilation by adding “-jN” to the previous command-line – where N is the number of cores.
- We will copy the bootloader to the SD Card. The flashing script that we have used before was doing this automatically, but we will do it manually here.

- **WARNING:** Make sure to use the correct device node because no confirmation will be asked! Using the wrong device node might cause you to lose all your data!

```
> sudo dd if=u-boot.bin of=/dev/[SD CARD reader] bs=1k skip=1 seek=1  
> sync
```

- Boot the board with the newly flashed bootloader and verify that it works correctly. Look at the debugging messages and verify that the build date corresponds to today's date.

Using the bootloader shell

- Once the serial terminal starts to display something, press the space bar to stop the boot process and use the shell.
- During the first lab, you have set variables to describe:
 - Linux kernel command line arguments
 - Copy the Linux kernel from SD Card to RAM
 - Start the Linux kernel at the specified RAM address
- Use *help* in the prompt to see the list of available commands:

```
U-Boot> help
```

- Use *help* boot to display the help of this specific command:

```
U-Boot> help boot
```

- Use *printenv* to display all the variables set:

```
U-Boot> printenv
```

Note: U-Boot accepts shorter commands as long as they are not ambiguous, e.g. *'pr'* for *'printenv'* or *'save'* for *'saveenv'*.

- Load the kernel from SD Card to RAM (at address *\${loadaddr}*) and dump the first bytes:

```
U-Boot> mmc dev 1  
U-Boot> mmc read ${loadaddr} 0x800 0x2000  
U-Boot> md ${loadaddr}
```

You will see the kernel tag (“*Linux-3.0.15-...*”).

Load the kernel image using TFTP

Using a TFTP to load the kernel can greatly speed up development times. A TFTP server has been pre-installed and configured in the virtual machine (procedure is documented separately).

- Review the TFTP server settings:

```
> cat /etc/default/tftpd-hpa
```

We can see that the server root ("*TFTP root*") is in */srv/tftp*.

- (Re)start the TFTP service on your host (training purposes only, since Ubuntu starts it at boot time once installed):

```
> sudo service tftpd-hpa restart
```

- Copy your kernel image to the TFTP root so that the board can access it:

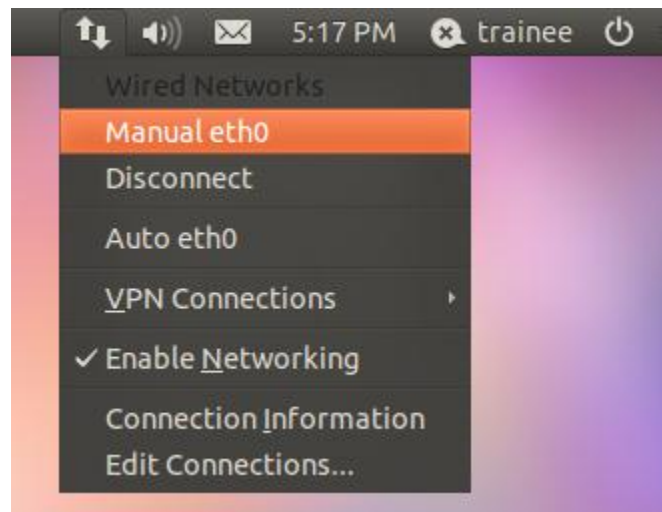
```
> cp ~/training_mx6_linux/ltib/rootfs/boot/uImage /srv/tftp/
```

- Test if the TFTP server is working by downloading a file to the host itself:

```
> cd /tmp
> tftp 127.0.0.1
> get uImage
> ^D (ctrl + D)
```

If you received the file *uImage* in your *tmp* directory, your TFTP server has been set up properly.

- Configure the host IP address. A preset has been created in the Network Manager. To use it, left-click on the networking icon (top-right bar), then on "Manual eth0". It will be connected when the board is booted.



- Configure U-Boot to load the kernel from TFTP (IP addresses assigned manually - host IP = 192.168.234.2, board IP = 192.168.234.3):


```

U-Boot> setenv kernel uImage
U-Boot> setenv autoload no
U-Boot> setenv bootcmd_tftp 'run bootargs_base bootargs_mmc; tftpboot
${loadaddr} ${serverip}:${kernel}; bootm'
U-Boot> setenv serverip 192.168.234.2
U-Boot> setenv ipaddr 192.168.234.3
U-Boot> saveenv
U-Boot> run bootcmd_tftp

```

Note: If you destroyed previous environment, you will need to setup the u-boot environment from lab 1 to 4. Please refer to `~/training_mx6_linux/files/u-boot.env` for the full list of commands.

Going further

- Display a message at the end of `int board_late_init(void)`:
See *u-boot-2009.08/board/freescale/mx6q_sabrelite/mx6q_sabrelite.c*
- Change the boot prompt:
See *u-boot-2009.08/include/configs/mx6q_sabrelite.h*
- Add a default environment variable (`CONFIG_EXTRA_ENV_SETTINGS`):
See *u-boot-2009.08/include/configs/mx6q_sabrelite.h*
- Information only – do not do this during the labs:
Use *destroyenv* to remove the user environment and use the default one (this will erase the region where the environment is permanently stored).



Lab 4: Cross Compiling the Linux kernel

Objectives

- Get the kernel sources from LTIB
- Apply the platform-specific patches
- Configure and cross compile the kernel
- Boot your own kernel

Get the kernel sources from LTIB

- We can get the kernel sources from LTIB as we did with U-Boot:

```
> cd ~/training_mx6_linux/ltib
> ./ltib -m prep -p kernel
```

- The sources are now present in `[LTIB]/rpm/BUILD/linux-3.0.15`. Let's create a separate copy on the side. We will use the fact that LTIB creates a hidden (since it renames the ".git" directory) Git repository for the kernel sources.
- Restore the Git repository:

```
> cd ~/training_mx6_linux/ltib/rpm/BUILD/linux
> mv .gitsaved/ .git
```

- Create a new copy by cloning the repository:

```
> cd ~/training_mx6_linux
> git clone ~/training_mx6_linux/ltib/rpm/BUILD/linux linux-3.0.15
```

- See how the patches for your platform have been applied (done by LTIB using "*git am*"):

```
> cd linux-3.0.15
> gitk
```

Configure and cross compile the kernel

- To cross-compile Linux, you will use the the cross-compiling toolchain provided by LTIB. Add the cross-compiling toolchain to your PATH:

```
> export PATH=/opt/freescale/usr/local/gcc-4.6.2-glibc-2.13-linaro-multilib-2011.12/fsl-linaro-toolchain/bin/:$PATH
```

Note: this only needs to be done once per terminal, every time you open a new terminal. If you forget to set it, you will see errors like "*cannot find gcc*" or your host gcc (for x86) might be used, leading to architecture issues.

- Since we are using U-Boot, we need to generate a *ulmage*, i.e. an image that is specifically packaged for this bootloader (it essentially adds a header to the actual

image).

The kernel build system will automatically invoke U-Boot's *mkimage* but we still need to add it to the PATH. We assume U-Boot has been built during the previous labs.

```
> export PATH=~/.training_mx6_linux/u-boot-2009.08/tools/:$PATH
```

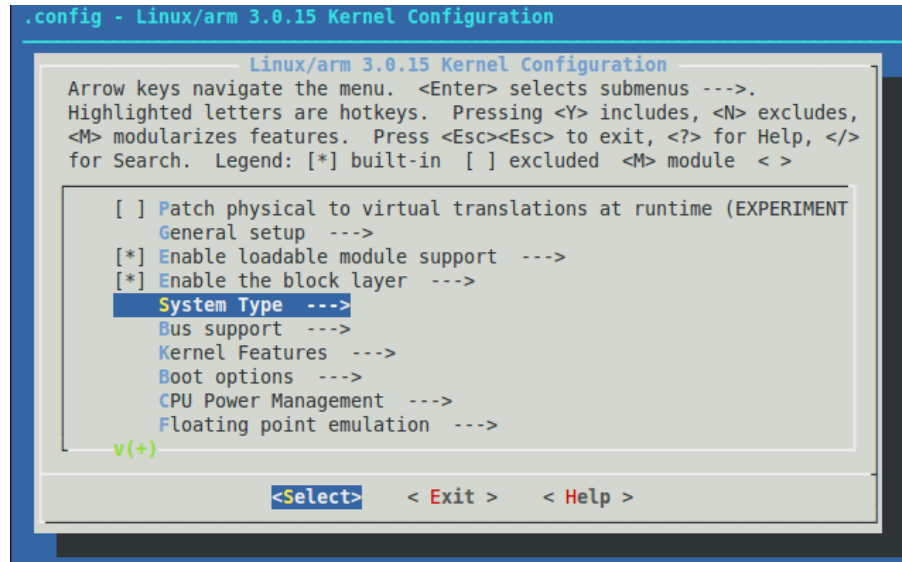
- Set the default configuration for our board (only needs to be done once - generic for all i.MX6 reference boards):

```
> cd ~/.training_mx6_linux/linux-3.0.15
> make ARCH=arm CROSS_COMPILE=arm-fsl-linux-gnueabi- imx6_defconfig
```

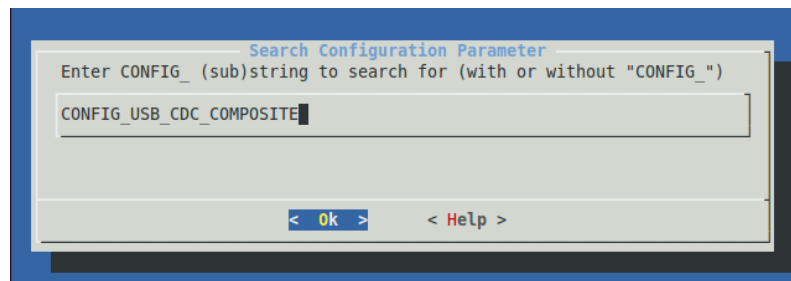
- Check the default configuration using the menu system. This is where you will want to go back if you need to make changes to the kernel.

```
> make ARCH=arm CROSS_COMPILE=arm-fsl-linux-gnueabi- menuconfig
```

- Make sure the system type corresponds to your device: go to *System Type* and verify that you have an i.MX based processor.

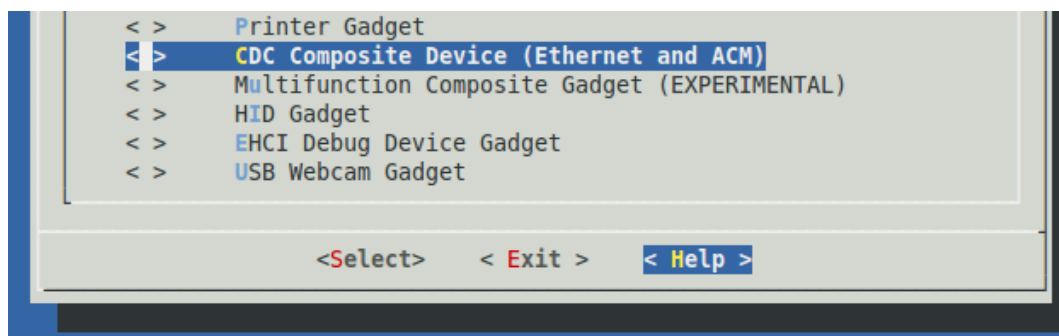


- Browse through the different options to see what the kernel provides. Take a look at the *drivers* section.
- When you are in the menu press the '/' (slash) key to use the search feature. Search for "*CONFIG_USB_CDC_COMPOSITE*".

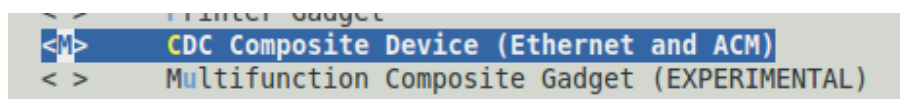


- The search result will show you that the option is located in *Device Drivers / USB support / USB Gadget Support / USB Gadget Drivers*.

Go to that section, highlight the item and use the Help feature to get more information.



When the item is highlighted (see picture above), press *space* to change how the driver is compiled. Make it compile as a module by pressing space until you see an *M*.



- Compile the kernel image for U-Boot:

```
> make ARCH=arm CROSS_COMPILE=arm-fsl-linux-gnueabi- uImage -j4
```

Note: you can speed up the compilation by adding “-jN” to the make command-line, where N is the number of processors on your machine.

- Verify that you can find the kernel image in *arch/arm/boot/*
- Compile the modules:

```
> make ARCH=arm CROSS_COMPILE=arm-fsl-linux-gnueabi- modules -j4
```

- Install the modules to your root filesystem:

```
> sudo make ARCH=arm INSTALL_MOD_PATH=~/.training_mx6_linux/ltib/rootfs modules_install
```

Note: we are using *sudo* here because the root filesystem is owned by *root*.

- Check the */lib/modules* directory in your target root filesystem and make sure everything has been installed correctly.

Testing the custom kernel

- Copy the kernel image to the TFTP root:

```
> cd ~/.training_mx6_linux/linux-3.0.15
> cp arch/arm/boot/uImage /srv/tftp
```

- Boot the board (see instructions given in the previous lab – you should not have to modify anything) and make sure that you are indeed using the kernel that you have just compiled (check the logs that the kernel emits when booting).

Flashing the kernel on an SD Card

- We will copy the kernel image to the SD Card. The flashing script that we have used before was doing this automatically, but we will do it manually here.

- **WARNING:** Make sure to use the correct device node because no confirmation will be asked! Using the wrong device node might cause you to lose all your data!

```
> sudo dd if=arch/arm/boot/uImage of=/dev/[SD CARD reader] bs=1k seek=1024
> sync
```

Going further

- Edit *arch/arm/mach-mx6/board-mx6q_sabrelite.c* and add a message at the end of *mx6_sabrelite_board_init()*. You can use *printf("MESSAGE");*. Verify that the message is displayed when you boot.
- Look at *arch/arm/mach-mx6/board-mx6q_sabrelite.c* and see how the board is initialized, how the board-specific resources are declared.

Lab 5: Using NFS

Objectives

- Mounting the root filesystem using NFS

NFS server setup

The NFS server has been prepared in the VM. We will only review the settings here (the setup procedure is provided separately).

- See which directories are exported by NFS (i.e. made accessible on the network), use the following command on your host:

```
> cat /etc/exports
```

You will see the following line at the end of the file:

```
/home/trainee/training_mx6_linux/ltib/rootfs
*(rw,no_subtree_check,async,no_root_squash,insecure)
```

This setting allows us to make our root filesystem accessible to anyone (most importantly, our board) on the network. If you need to share other directories, you can replicate this line.

- Restart the NFS service (training purposes only, since Ubuntu starts it at boot time once installed):

```
> sudo service nfs-kernel-server restart
```

- Let's test that our setup is correct by mounting the NFS share on the PC itself (think of it as a loopback test):

```
> cd /tmp
> mkdir test_rootfs
> sudo mount -t nfs localhost:/home/trainee/training_mx6_linux/ltib/rootfs
test_rootfs
```

If everything goes well, you should see your root filesystem contents inside `/tmp/test_rootfs`. You can now unmount the directory on your host:

```
> sudo umount test_rootfs
```

Using NFS on the target device

- Make sure that your device is still correctly connected to your PC.
- Reboot your board and press space to get to the U-Boot shell.



- **Note:** The network settings have been set and saved during the previous labs. If not:

```
U-Boot> setenv serverip 192.168.234.2
U-Boot> setenv ipaddr 192.168.234.3
U-Boot> setenv bootcmd_tftp 'run bootargs_base bootargs_mmc; tftpboot
${loadaddr} ${serverip}:${kernel}; bootm'
U-Boot> save
```

- Configure the kernel command line to mount the root filesystem from NFS:

```
U-Boot> setenv nfsroot /home/trainee/training_mx6_linux/ltib/rootfs
U-Boot> setenv bootargs_nfs 'setenv bootargs ${bootargs} root=/dev/nfs
ip=${ipaddr} nfsroot=${serverip}:${nfsroot}'
U-Boot> setenv bootcmd_nfs 'run bootargs_base bootargs_nfs; mmc dev 1;mmc
read ${loadaddr} 0x800 0x2000; bootm'
U-Boot> save
```

- Boot your device:

```
U-Boot> run bootcmd_nfs
```

Going further: Testing Qt using NFS

We will now use NFS to use a prebuilt root filesystem that contains the Qt framework libraries and sample programs.

- Extract the prebuilt root filesystem:

```
> cd ~/training_mx6_linux
> mkdir rootfs_qt
> cd rootfs_qt
> sudo tar -xjf ../files/prebuilt/rootfs_qt.tar.bz2
```

- Add the new directory to the list of NFS exports:

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

Insert the following line at the end of the file and save it (make sure it spans on a single line):

```
/home/trainee/training_mx6_linux/rootfs_qt
*(rw,no_subtree_check,async,no_root_squash,insecure)
```

- Restart the NFS server:

```
> sudo service nfs-kernel-server restart
```

- Reboot your device and update the bootloader environment to update the kernel command-line:

```
U-Boot> setenv nfsroot /home/trainee/training_mx6_linux/rootfs_qt
```

- Boot your device:

```
U-Boot> run bootcmd_nfs
```

Now that your device has booted it “sees” the same files as your host, in the NFS shared directory.

- From your host, create a script that will be used later on the target to set up environment for Qt:

```
> sudo gedit /home/trainee/training_mx6_linux/rootfs_qt/root/setup_qt.sh
```

And add the following lines:

```
#!/bin/sh

export TSLIB_CONFFILE=/usr/etc/ts.conf
export TSLIB_PLUGINDIR=/usr/lib/ts
export TSLIB_FBDEVICE=/dev/fb0
export TSLIB_TSDEVICE=/dev/input/ts0
export QWS_MOUSE_PROTO=Tslib:/dev/input/ts0
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/Trolltech/lib
```

Save and exit.

- On the device, you should now see the file in */root*:

```
mx6# ls /root/
```

- Source that file:

```
mx6# source /root/setup_qt.sh
```

Note: as opposed to merely running the script, sourcing it will define the variables in your current shell (instead of a forked copy that disappears when the script returns).

- Calibrate the touch screen by launching the *ts_calibrate* utility and following the instructions printed on the screen:

```
mx6# ts_calibrate
```

- Run the Qt *fluidlauncher* example (quit with CTRL-C on the console):

```
mx6# cd /usr/local/Trolltech/Qt-4.8.2
mx6# cd demos/embedded/fluidlauncher
mx6# ./fluidlauncher -qws
```

Lab 6: Working with applications

Objectives:

- Cross compile an application
- Use Eclipse to edit and build the source code
- Remote debugging using Eclipse
- Adding a package to LTIB

Cross compile an existing project

- Extract the *multicrunch* sample application:

```
> cd ~/training_mx6_linux
> tar xzf files/multicrunch-1.0.tar.gz
> cd multicrunch-1.0
```

- Look at the *Makefile*. See how it uses a variable named *CC* to refer to the compiler. We will use it to specify the path to the cross-compiler.
- Add the cross-compiling toolchain to your *PATH*:

```
> export PATH=/opt/freescale/usr/local/gcc-4.6.2-glibc-2.13-linaro-multilib-2011.12/fsl-linaro-toolchain/bin/:$PATH
```

Note: this only needs to be done once per terminal, every time you open a new terminal. If you forget to set it, you will compile the program with the default compiler, i.e. your x86 GCC.

- Now, it is time to compile and install our application to the target root filesystem:

```
> cd multicrunch-1.0
> CC=arm-fsl-linux-gnueabi-gcc make
> sudo make install DESTDIR=~/training_mx6_linux/ltib/rootfs
```

- Check that the root filesystem contains the target binary and that it has indeed been compiled for ARM:

```
> file ~/training_mx6_linux/ltib/rootfs/usr/bin/multicrunch
/home/trainee/training_mx6_linux/ltib/rootfs/usr/bin/multicrunch: ELF 32-bit
LSB executable, ARM, version 1 (SYSV), dynamically linked (uses shared libs),
for GNU/Linux 2.6.31, not stripped
```

Booting the device

- If you have mounted your root filesystem using NFS:
 - Make sure you are not using the Qt root filesystem from the previous lab (in `/home/trainee/training_mx6_linux/ltib/rootfs`)

```
U-Boot> setenv nfsroot
/home/trainee/training_mx6_linux/ltib/rootfs
```

```

U-Boot> setenv bootargs_nfs 'setenv bootargs ${bootargs}
root=/dev/nfs ip=${ipaddr} nfsroot=${serverip}:${nfsroot}'

U-Boot> setenv bootcmd_nfs 'run bootargs_base bootargs_nfs; mmc
dev 1; mmc read ${loadaddr} 0x800 0x2000; bootm'


U-Boot> save

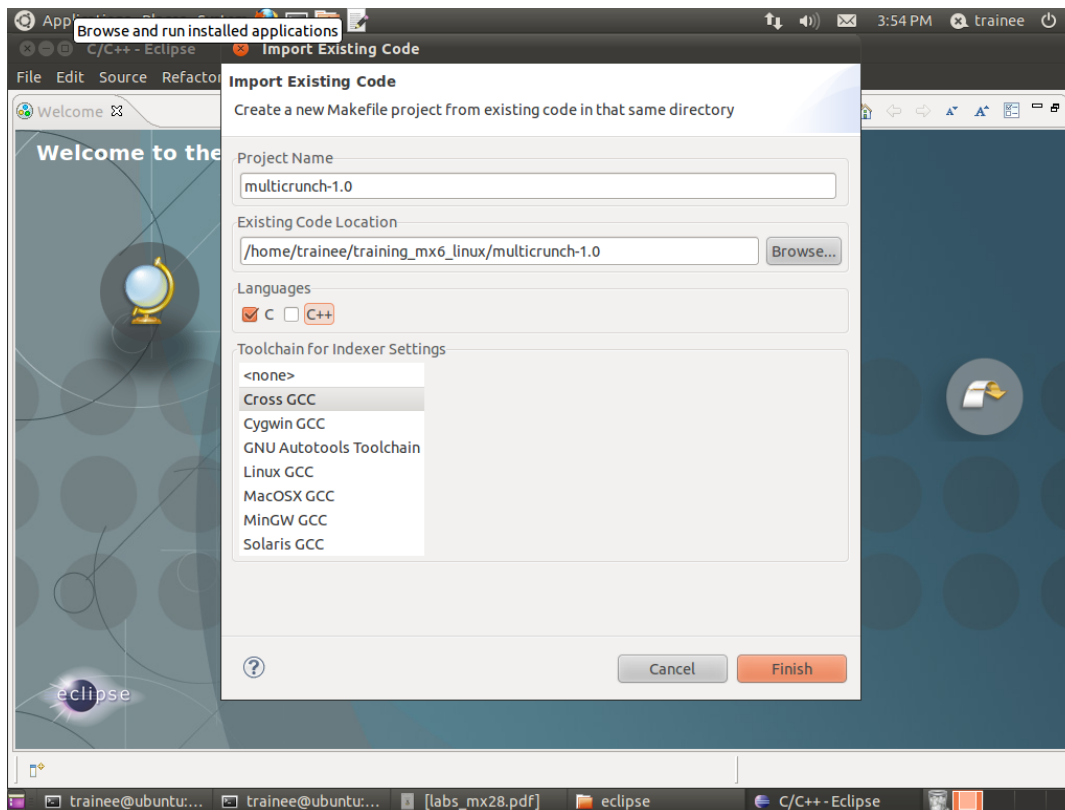
U-Boot> run bootcmd_nfs

```

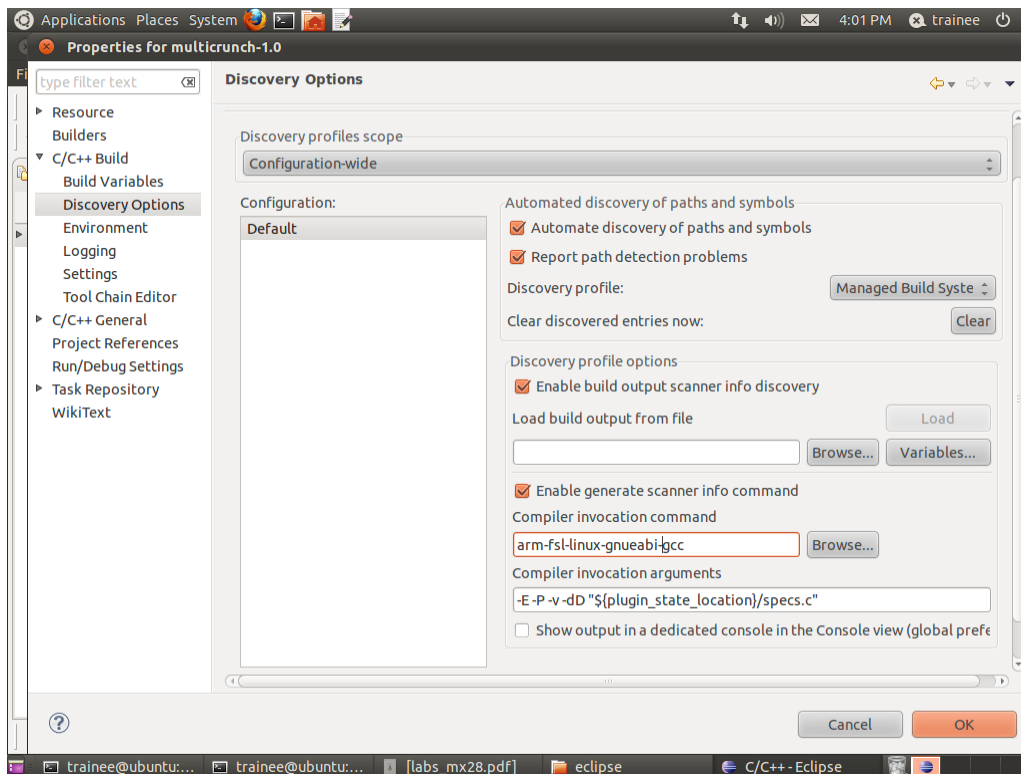
- You will see the executable on the target without having to reboot.
- If it is on the SD Card, you can copy the executable to it, then reboot (and see how efficient NFS is compared to that method).

Import and cross compile an existing project in Eclipse

- Eclipse has been pre-installed in the virtual machine (instructions provided separately).
- Start Eclipse by using the shortcut in the quick launch bar  or by using Nautilus (the file explorer) into `~/training_mx6_linux/tools` and double-clicking on the *eclipse* executable.
- Choose `'/home/trainee/training_mx6_linux/eclipse-ws'` as the current workspace.
- Create a new project (*File / New / Makefile project with existing code*)
- Fill the different fields:
 - **Project name:** use any name you want, e.g. *multicrunch-1.0*
 - **Code location:** `/home/trainee/training_mx6_linux/multicrunch-1.0`
 - **Toolchain options:** *Cross GCC*
 - **Languages:** *C*



- Click on *Finish* and close the Welcome panel by clicking on the arrow-shaped icon (on the right) to go to the *Workbench*.
- Right-click on your project in the *Project Explorer* panel (to the left) and select *Properties*.
- Select '*C/C++ Build | Discovery Options*' in the left panel (inside the C/C++ Build) and in the main panel choose:
 - '*Discovery profiles scope*' (drop-down list) select *Configuration-wide*.
 - '*Compiler invocation command*': *arm-fsl-linux-gnueabi-gcc*



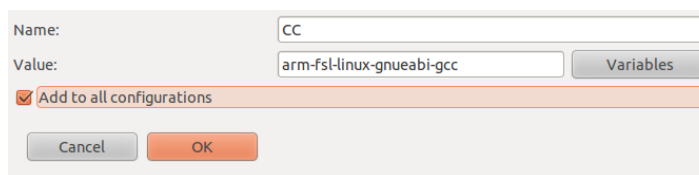
- Select '*Environment*' in the left panel and edit the variable **PATH**. Add the path of your toolchain (*/opt/freescale/usr/local/gcc-4.6.2-glibc-2.13-linaro-multilib-2011.12/fsl-linaro-toolchain/bin/*) at the beginning of the string and separate it with ':'. The entire field should look like this:

```
/opt/freescale/usr/local/gcc-4.6.2-glibc-2.13-linaro-multilib-2011.12/fsl-linaro-toolchain/bin:/bin:/usr/sbin:/usr/bin:/usr/games
```

Click on **OK**.

- Click on **Add** to define another variable named **CC** and set it to '*arm-fsl-linux-gnueabi-gcc*'. Also select '*Add to all configurations*'.

We have seen that the *Makefile* uses $\$(CC)$ to invoke the cross-compiler.



- Click **OK** to validate your modifications.
- Use the editor to look around the *.c* file and see how you can jump to a symbol declaration by CTRL-clicking on it. You can also generate a call-graph by highlighting a function and using CTRL-ALT-H.

Building the project

- You can now build by right clicking the project in the *Project Explorer* panel (to the left) and select '*Build Project*'.
 - If everything went well, you will see two binary files:
 - multicrunch – [arm/le]*
 - multicrunch.o – [arm/le]*
 - If not, you might have encountered an error. Look at the *Console* to see the build log. It is likely that the PATH to the cross-compiler or its name have not been set correctly.
- You can also verify that cross compilation worked by typing in a console:

```
> file ~/training_mx6_linux/multicrunch-1.0/multicrunch
```

which should result in:

```
multicrunch: ELF 32-bit LSB executable, ARM, version 1 (SYSV), dynamically
linked (uses shared libs), for GNU/Linux 2.6.31, not stripped
```

Enabling support for remote debugging

Eclipse is able to connect to the target using the SSH protocol and to use a remote debugger (*gdb/gdbserver*) to debug your application.

- To be able to remotely debug your application, you should first install all required package on the target. They have been installed by default, so we will just verify the settings:

```
> cd ~/training_mx6_linux/ltib
> ./ltib -c
```

- From the package list, select:
 - gdb*
 - cross gdb*
 - gdbserver (auto-selected no possibility to modify)*
 - openssh*
- Make sure that *dropbear* is not selected (this version does not support SFTP, which is required to interact with Eclipse).
- Go back to the main menu and select the Target System Configuration options. Check '*start openssh server*'.
- Save and exit.
- If NFS is working, skip to the next step, otherwise, flash the root filesystem on the SD Card and boot using the SD Card.

- Reboot the board and log in as *root*. Use the NFS setup from the previous lab.
- Once the board has booted, make sure that it has an IP address:

```
mx6# ifconfig
```

You should see the following output:

```
root@freescale ~$ ifconfig
eth0      Link encap:Ethernet  HWaddr 00:19:B8:00:F0:21
          inet addr:192.168.234.3  Bcast:192.168.234.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

If not, assign an IP address manually using the serial shell:

```
mx6# ifconfig eth0 192.168.234.3
```

- Now, you should be able to connect to your target using SSH. If you are asked to accept a key, say 'yes'.

```
> ssh root@192.168.234.3
```

You should be logged on the remote system (you can use it like the serial console). To exit that remote shell, use the 'exit' command:

```
root@freescale ~$ exit
```

- If you get an error saying that your password has expired, you will need to change the root password for SSH to work (it cannot be empty for SSH to work). On the target, set the using:

```
mx6# passwd
```

Then, make sure that the password does not expire:

```
mx6# awk 'BEGIN { OFS=FS = ":" } ; /^'root/' {$3=99999} {print}' <
/etc/shadow > /etc/shadow.tmp && mv /etc/shadow.tmp /etc/shadow
```

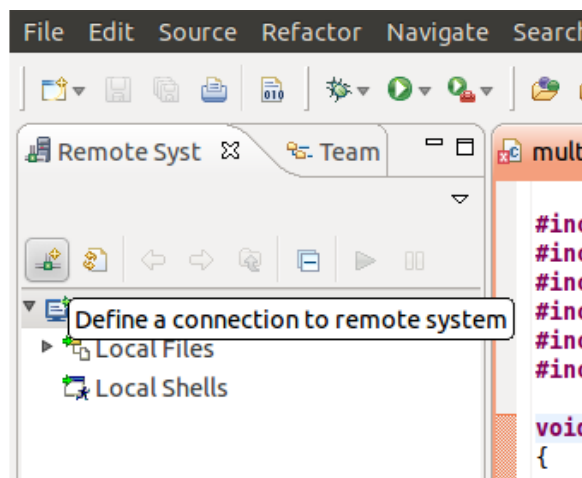
- From your host, copy a recent version of *gdbserver* that works with our toolchain.
Note: the version built by LTIB is too old to be used with our current toolchain so we have precompiled our own from GDB 7.4. We will also not overwrite it:

```
> scp /home/trainee/training_mx6_linux/files/gdb/gdbserver
root@192.168.234.3:/root
```

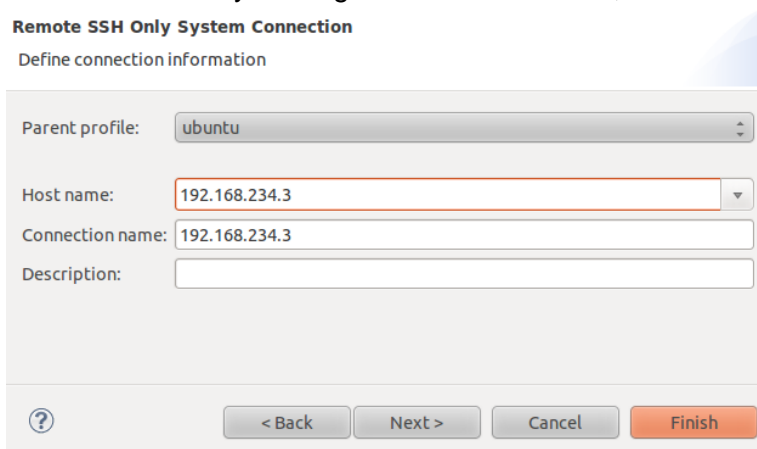
Remote debugging with Eclipse

To debug our application remotely, we need to setup a new connection to the target inside Eclipse.

- Click on the menu *Window / Open Perspective / Other...*
- Select '*Remote system explorer*'.
- In the '*Remote Systems*' panel (on the left), click on the button 'Define a connection to remote system'

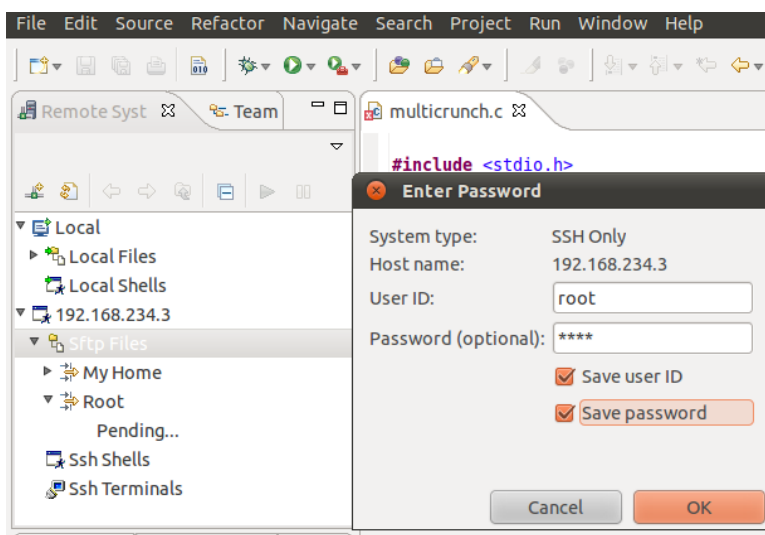


- Select '*SSH only*', then *Next*.
- Set the Hostname field to your target IP: 192.168.234.3, and click on *Finish*.



- Use the *Remote Systems* panel and expand the connection that you have created.
- You should be able to navigate into your target filesystem through *Sftp Files / Root*. You will be prompted you login information:
 - User: '*root*'
 - Password: '*root*'

- Check save boxes to save user ID and password

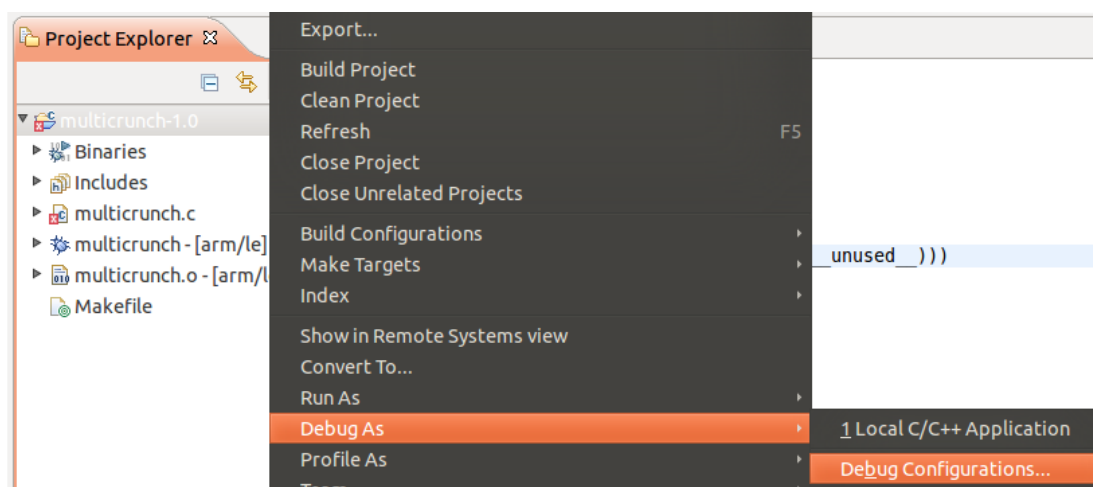


Now, we only need to configure the remote debugger.

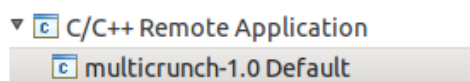
- Click on the menu *Window / Open Perspective / Other...*
- Select 'C/C++'
 - **Note:** the perspectives can be accessed easily using the top-right tab



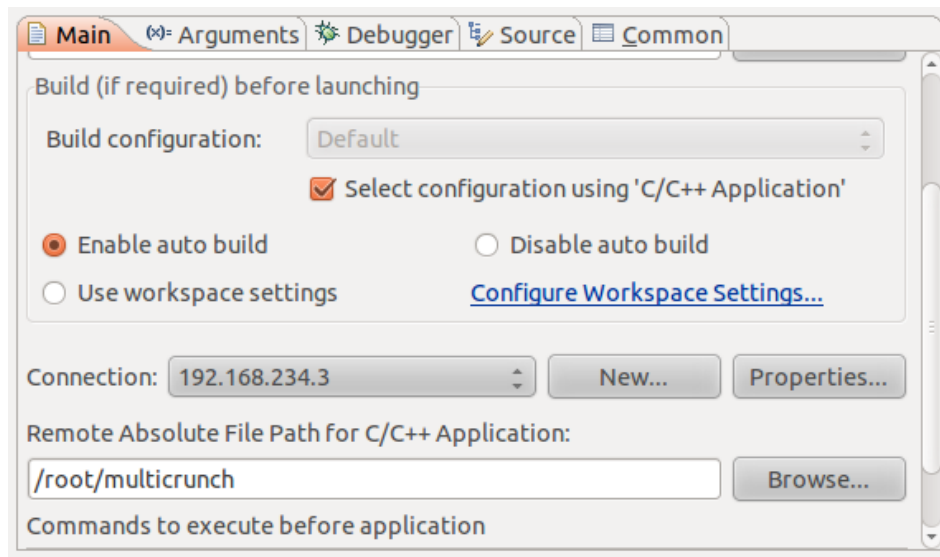
- Right-click on your project in the *Project Explorer* panel and select '*Debug As / Debug Configurations...*'



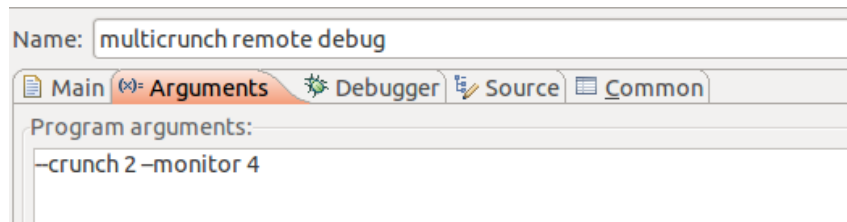
- In the left panel, double-click on the '*C/C++ Remote Application*' entry, then click on '*multicrunch-1.0 Default*'.



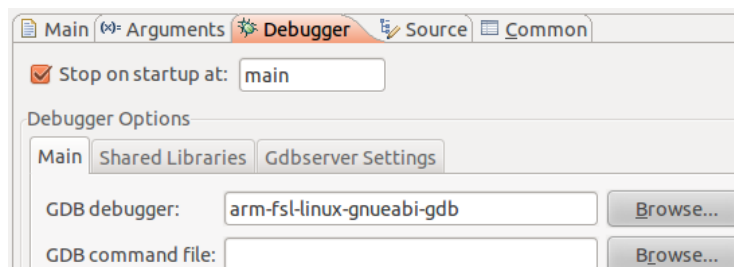
- From the *Main* tab of the right panel:
 - Using the 'Name' field, rename this configuration as: '*multicrunch remote debug*' (for clarity only, as the name does not really matter)
 - Select '*Enable auto build*'.
 - Select the connection you have just created: *192.168.234.3*
 - Select the remote absolute file path for C/C++ Application: */root/multicrunch*



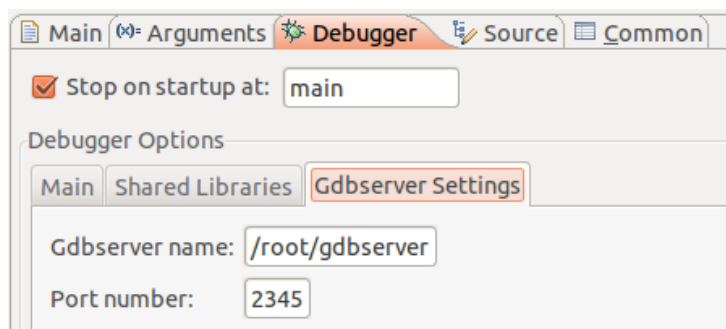
- From the *Arguments* tab, enter the command-line arguments: *--crunch 2 --monitor 4*



- From the *Debugger* tab:
 - From the *Main* tab:
 - GDB debugger: *arm-fsl-linux-gnueabi-gdb*
 - GDB command file: [empty]

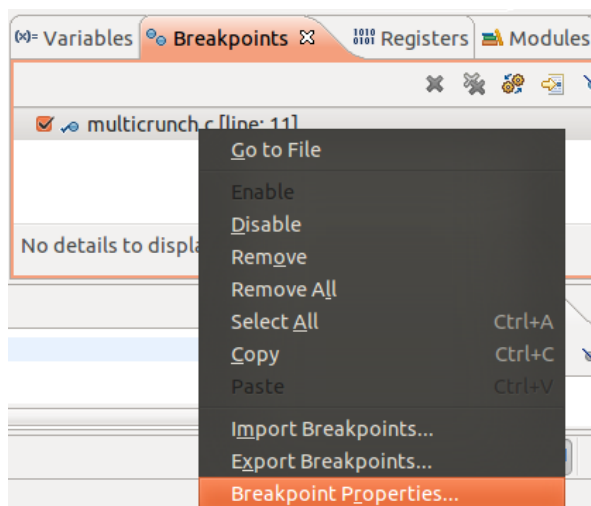


- From the *Gdbserver Settings* tab:
 - Gdbserver name: `/root/gdbserver`
 - Port number: 2345



- Click on **Apply** to validate your modifications.
- Click on **Debug** to start debugging. You will be prompted to open a Debug perspective, say Yes.
 - **Note:** if you encounter deployment errors, use a shell on the target to remove the previous copy of the executable, then try again:


```
mx6# rm /root/multicrunch
```
- The *multicrunch* program is now running (output in the lower panel, in the *Console* tab). By default, Eclipse configures GDB to stop at the first instruction.
- You can now add a breakpoint by double-clicking in the left margin corresponding to the desired line. You can also use the “step” functions and peek at the different variables.
- If you want to restrict a breakpoint to a specific thread, you can use breakpoint filters from the *Breakpoint Properties* window (see below).



Going further: Adding your application to LTIB

Adding your custom application in LTIB is an easy task.

- First, remove the binaries that we have installed previously:

```
> sudo rm ~/training_mx6_linux/ltib/rootfs/usr/bin/multicrunch
```

- Create the .spec file in the LTIB directory:

```
> mkdir ~/training_mx6_linux/ltib/dist/lfs-5.1/multicrunch
> gedit ~/training_mx6_linux/ltib/dist/lfs-5.1/multicrunch/multicrunch.spec
```

- Edit *multicrunch.spec* and copy the following:

```
%define pfx /opt/freescale/rootfs/%{_target_cpu}

Summary      : Multicore Cruncher/Monitor
Name         : multicrunch
Version      : 1.0
Release      : 1
License      : Public Domain, not copyrighted
Vendor       : Adeneo
Packager     : Tristan Lelong
Group        : Applications/Test
Source       : %{name}-%{version}.tar.gz
BuildRoot    : %{_tmppath}/%{name}
Prefix       : %{pfx}

%Description
%{summary}

%Prep
%setup

%Build
make

%Install
rm -rf $RPM_BUILD_ROOT
make install DESTDIR=$RPM_BUILD_ROOT/

%Clean
rm -rf $RPM_BUILD_ROOT

%Files
%defattr(-,root,root)
/usr/bin/multicrunch
```

- Copy the sources to the correct location to avoid downloading



```
> cp /home/trainee/training_mx6_linux/files/multicrunch-1.0.tar.gz  
/opt/freescale/pkgs/
```

- Define the package in ltib config

```
> gedit config/userspace/extra_packages.lkc
```

- And append the following:

```
config PKG_MULTICRUNCH  
    bool "multicrunch"  
    help  
        cruncher/monitor for multicore platforms
```

- Then we need to link it to the build system

```
> gedit config/userspace/pkg_map
```

- and append almost at the end of the file (after PKG_QT_EMBEDDED = qt-embedded)

```
PKG_MULTICRUNCH                                = multicrunch
```

- Configure and build ltib

```
> ./ltib -c
```

- Select the *multicrunch* package (packages list/extra packages)
- Save and exit, the build will start. At the end, you should be able to see that multicrunch is installed in the ltib rootfs.

