March 29, 2011

Android™ and Linux™ on the new i.MX53 Quick Start board

Remi Lorriaux (Adeneo Embedded)

Embedded Software Engineer – Linux, Android

freescale™ semiconductor

Adeneo Embedded

Training Overview

freescale™ semiconductor

Adeneo Embedded

Training Agenda

- **Morning**
  - Presentation of the i.MX53 Quick Start Board (*30mins*)
  - Linux on the i.MX53 Quick Start Board
    - Quick introduction to Linux for embedded devices (*1hr*)
    - Building and using Linux for the Quick Start Board (*1hr 30mins*)

- **Afternoon**
  - Android on the i.MX53 Quick Start Board (*3hrs*)
    - General presentation of Android
    - Using Android with the i.MX53 Quick Start Board
    - Writing applications for your device

Who am I?

- **Remi Lorriaux**
  - Embedded Software Engineer at Adeneo Embedded
  - Linux, Android, Windows CE
  - BSP adaptation, board bring-up, driver development
  - Training delivery (Linux and Android)
  - rlorriaux@adeneocorp.com

- **Adeneo Embedded**
  - Turnkey Design
  - BSP and Driver Development
  - Application Development

  - Freescale partner (Linux, Android, Windows Embedded)

Introducing the i.MX53 Quick Start board

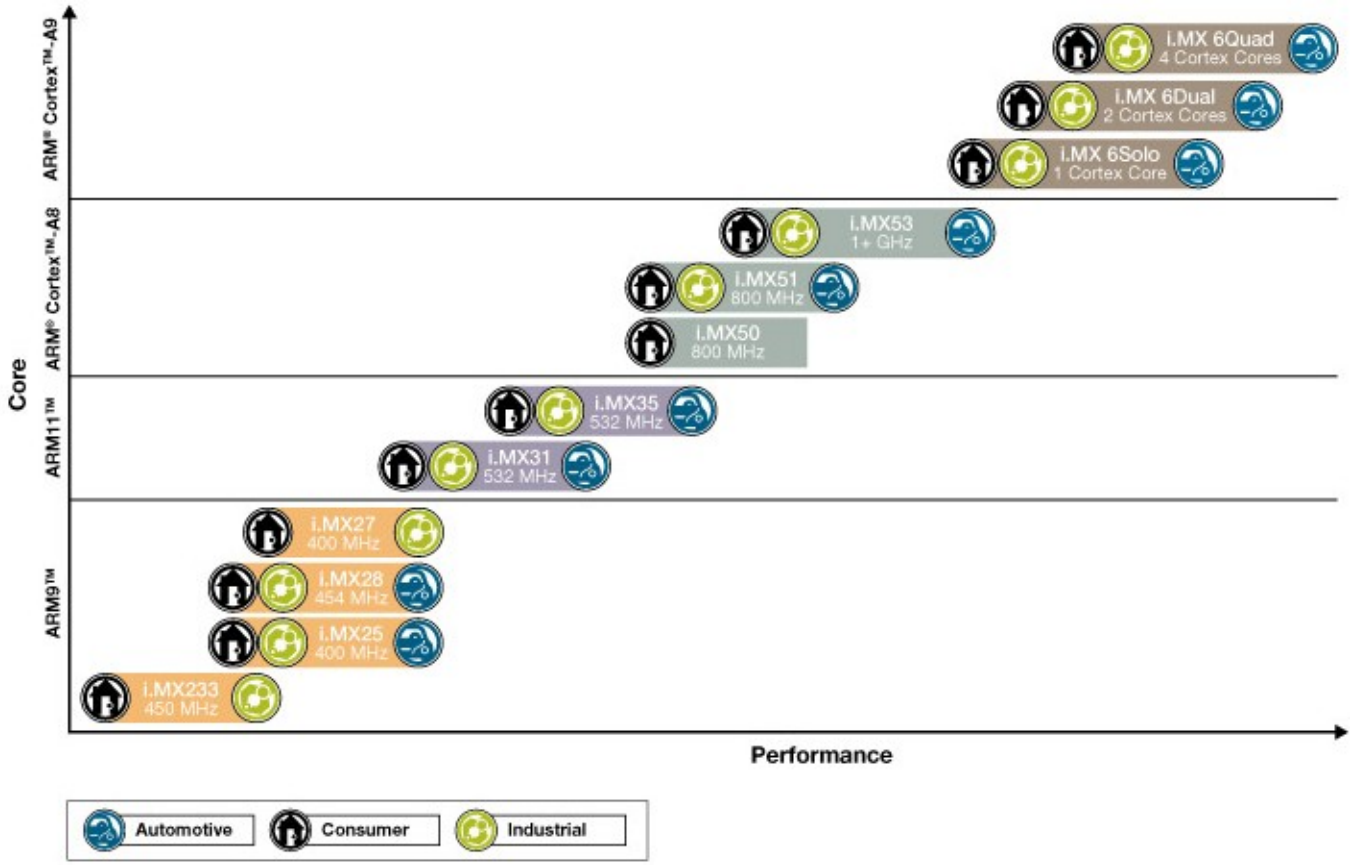
freescale™ semiconductor

Adeneo Embedded

i.MX Applications Processors

- Multimedia applications processors
  - ARM9, ARM11, ARM Cortex™-A8, ARM Cortex-A9
  - From 400 MHz to 1+ GHz
  - Energy efficient
- System-on-chip
  - Display, Network, Communication buses, Multimedia... **all in a single package**.
- Markets:
  - Consumer
  - Automotive
  - Industry
- i.MX family webpage on Freescale's website

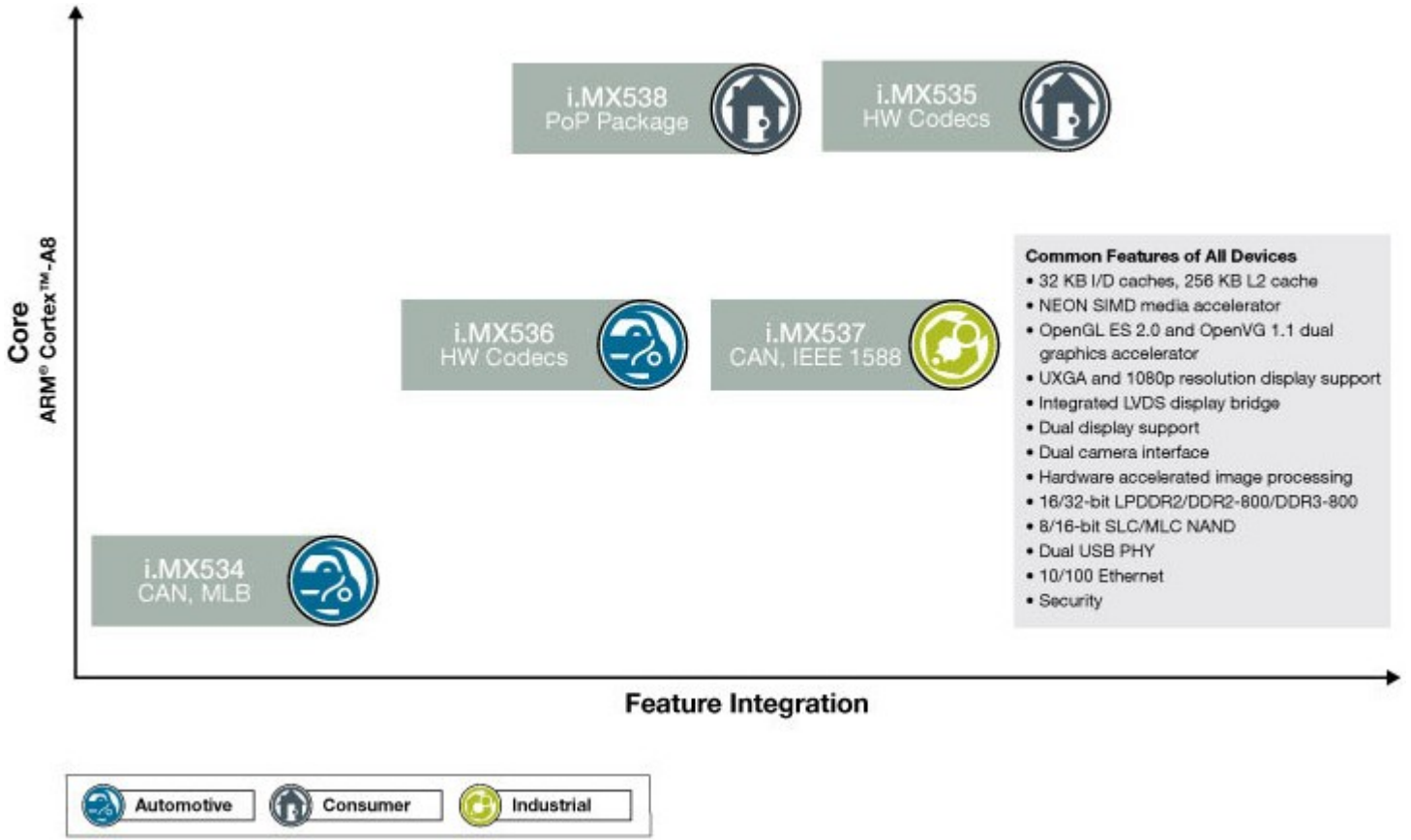
i.MX portfolio



- i.MX2 (ARM9)
  - 454 MHz
- i.MX3 (ARM11)
  - 532 MHz
- i.MX5 (Cortex-A8)
  - 1+ GHz
- i.MX6 (Cortex-A9)
  - Multi-core

i.MX53 Multimedia Applications Processors

- Advanced multimedia / Power efficient
- ARM Cortex-A8 core
- Up to 1.2 GHz



i.MX53 key features

- ARM Cortex-A8 (NEON, VFP)
- DDR2/DDR3
- **Storage**
  - NAND
  - SDCard
- **Connectivity**
  - USB
  - 10/100 Ethernet
  - SATA
  - CAN modules
  - I2C, SPI...

i.MX53 key features

- **Multimedia**
  - OpenGL ES2.0 and OpenVG 1.1 dual graphics accelerator
  - Dual-display and dual-camera interface
  - Hardware accelerated image processing
  - 1080p video decode, 720p video encode
- **Security**
- Package-on-Package (i.MX538)

- … and more

  See the i.MX53 webpage and Reference Manual

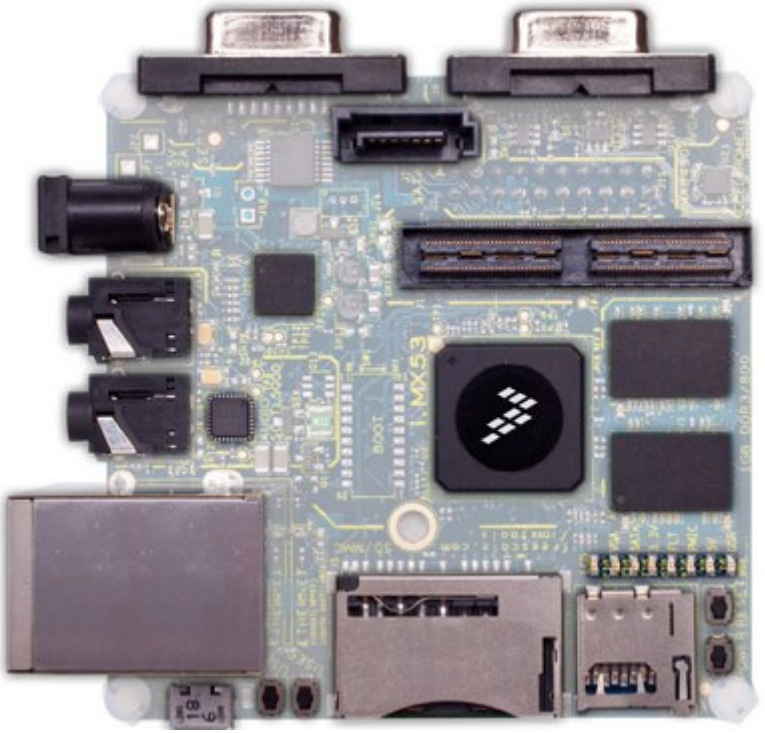
Evaluating the i.MX53

- SABRE Platform for Tablets

- i.MX53 Quick Start Board

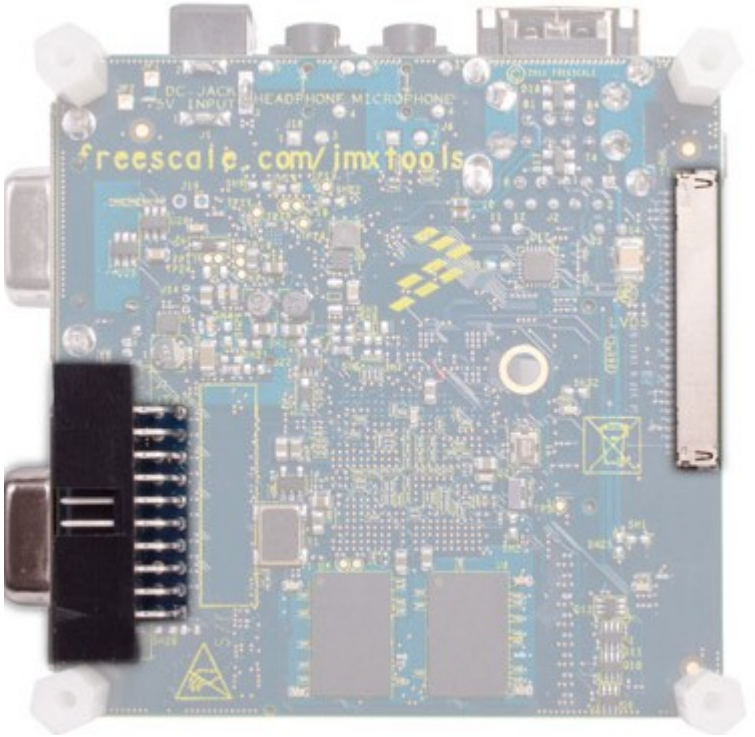
i.MX53 Quick Start Board

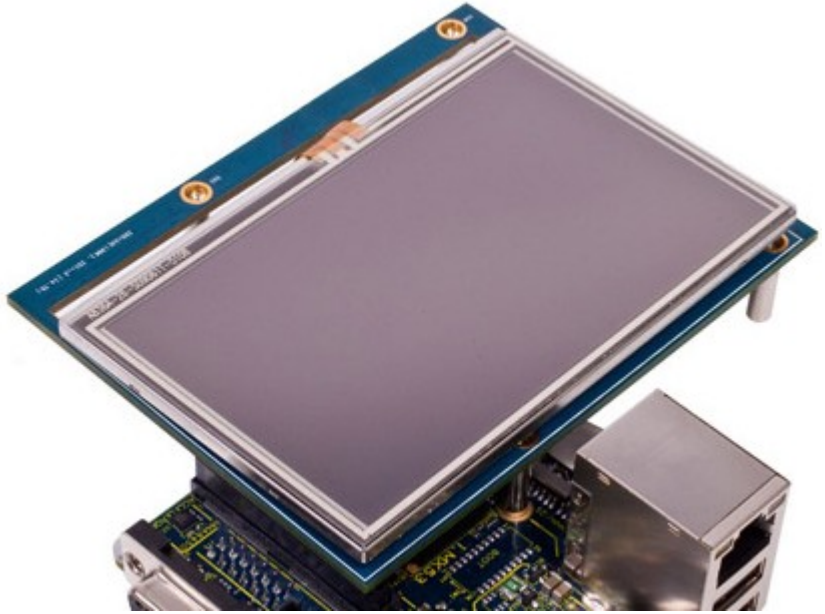
- i.MX53
- 1GB DDR3 SDRAM
- Debug UART connector
- VGA connector
- SATA 7-pin connector
- Wall 5V power jack
- Headphone out
- Microphone in
- Ethernet
- Dual USB
- SD Card (data)
- MicroSD Card (system)



i.MX53 Quick Start Board

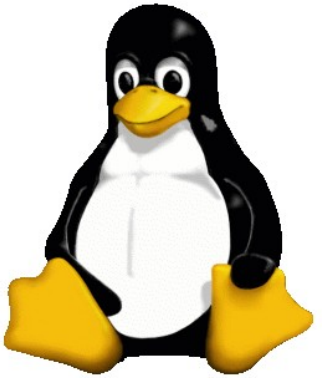
- 3-axis accelerometer
- Optional LCD screen 4.3” 800x480 WVGA and touchscreen
- HDMI and SPDIF add-on card via Expansion connector
- JTAG connector
- 30-pin LVDS connector





Software for the i.MX53 Quick Start Board

- OS Support:
  - **Linux** (from Freescale)
  - **Android** (from Adeneo)
    Froyo, Gingerbread
  - **Windows Embedded Compact 7** (from Adeneo)
- Freescale also provides a large portfolio of optimized video, speech and audio codecs
- Inflexion™ UI for i.MX processors by Mentor Embedded™
- Get latest software from Freescale's website



Linux on the i.MX53 Quick Start Board

freescale™ semiconductor

Adeneo Embedded

Linux Agenda

- Introduction to Linux for embedded systems
  - Key Linux features
  - The different software components of a system (bootloader, kernel, root filesystem)
  - How to build a system using different tools
- Linux for the i.MX53 Quick Start Board
  - LTIB
  - Labs:
    - Assembling your i.MX53 kit
    - Using prebuilt images
    - Building and using your own images with LTIB
    - Flashing and using Ubuntu

Linux history

- Created by Linus Torvalds in 1991
- Thousands of people and companies contributed code to the project
- Very popular in the embedded world

Date	Version	Lines of code
March 1994	1.0.0	176,250
March 1995	1.2.0	310,950
January 1999	2.2.0	1,800,847
January 2001	2.4.0	3,377,902
December 2003	2.6.0	5,929,913
March 2011	2.6.38	14,294,439

Linux key features

- **Portability**
  - Supported architectures (see *arch* directory in the Linux sources): alpha, arm, m68k, x86, mips, powerpc, sparc...
- **Scalability**
  - Used on small embedded devices to super-computers
- **Security**
  - The code is constantly being reviewed by the community
- **Reusability**
  - Many drivers and platforms are part of the mainline. No need to reinvent them!
  - Well-defined coding standards
- **Community support**
  - Easy to find support and documentation

Linux development model

- Latest version is 2.6.38
  - Part of the 2.6 branch (released in 2003)
  - About one release every 3 months
  - Stable branches are maintained by a dedicated team (only the security fixes are backported)
- Kernel sources available on http://kernel.org/
  - Can be downloaded as archives or with git
  - "*Mainline*" or "*Vanilla*" kernel: contain the main, generic branch of development
  - Released by Linus Torvalds after integrating the changes made by all other programmers
- Not all the Linux code is part of the *mainline*
  - Silicon Vendors typically manage their own tree

Licensing considerations

- The Linux kernel is licensed under the GPLv2
- The GPL does not require you to release your modified version, or any part of it. You are free to make modifications and use them privately, without ever releasing them.
- The GPL requires you to make the modified source code available to the program's users, under the GPL.
- GPL FAQ

- Before reusing code and libraries, check the license of the different software packages!

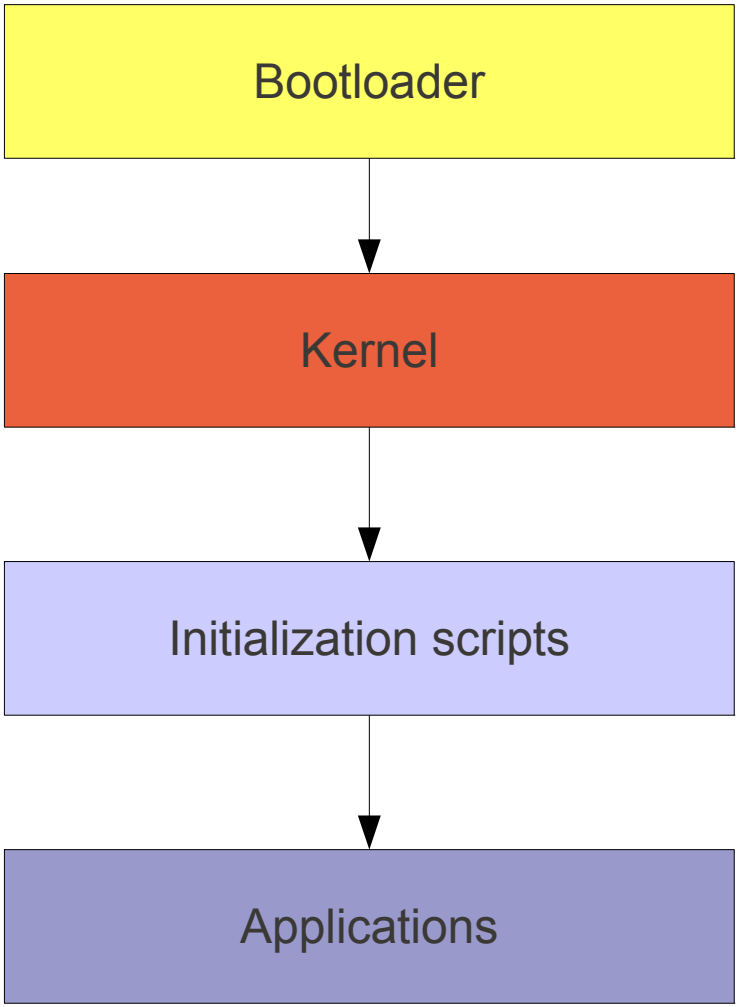
- Other open licenses exist (Apache, BSD, GPLv3, LGPL)
  - Different possibilities/constraints

Contributing to Linux

- Contributing to Open Source projects is not mandatory, but it has several benefits:
  - Code reviewed and corrected by experts
  - Code maintained by the community
  - Increasing the popularity of your company

- Open Source projects typically use:
  - Mailing-lists
    - Also a way to find solutions to issues that you might have
  - Version-control servers

Booting Linux



Bootloader

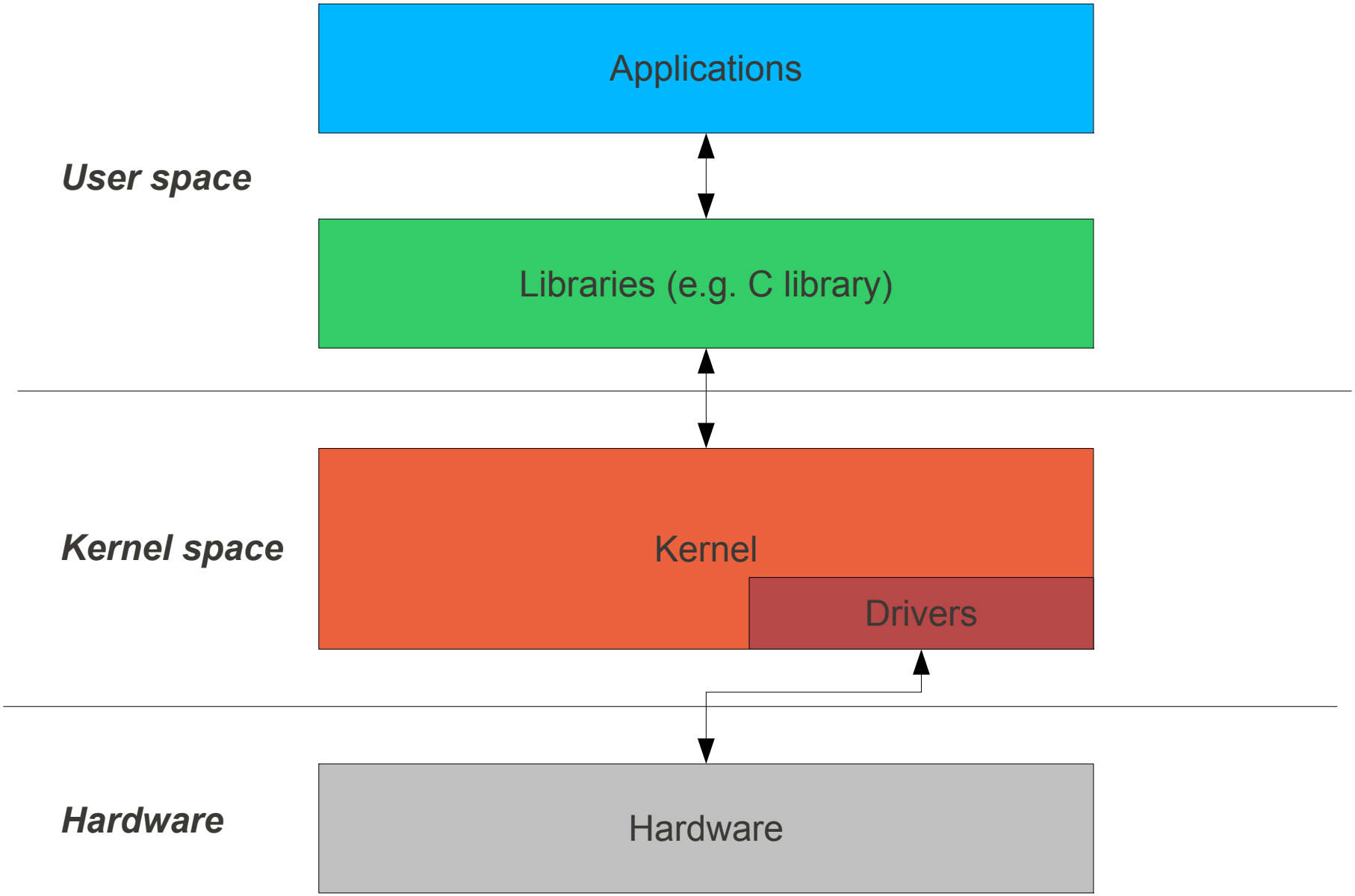
- Implementations in the embedded world:
  - Redboot, **U-Boot**, Barebox
  - Typical x86 bootloaders (LILO, GRUB) are not appropriate
- Executed first when the board boots
- Located at a predefined spot (hardware-dependent):
  - Flash memory, SD Card, Hard-drive...
- **Initializes:**
  - CPU
  - Clocks
  - Memory
  - Hardware required to load the kernel

Bootloader

- **Loads the kernel:**
  - From: Flash memory, SD Card, LAN
  - To: RAM
- **Jumps to the kernel**
- **Provides a command prompt**
  - Configuration
  - Debugging

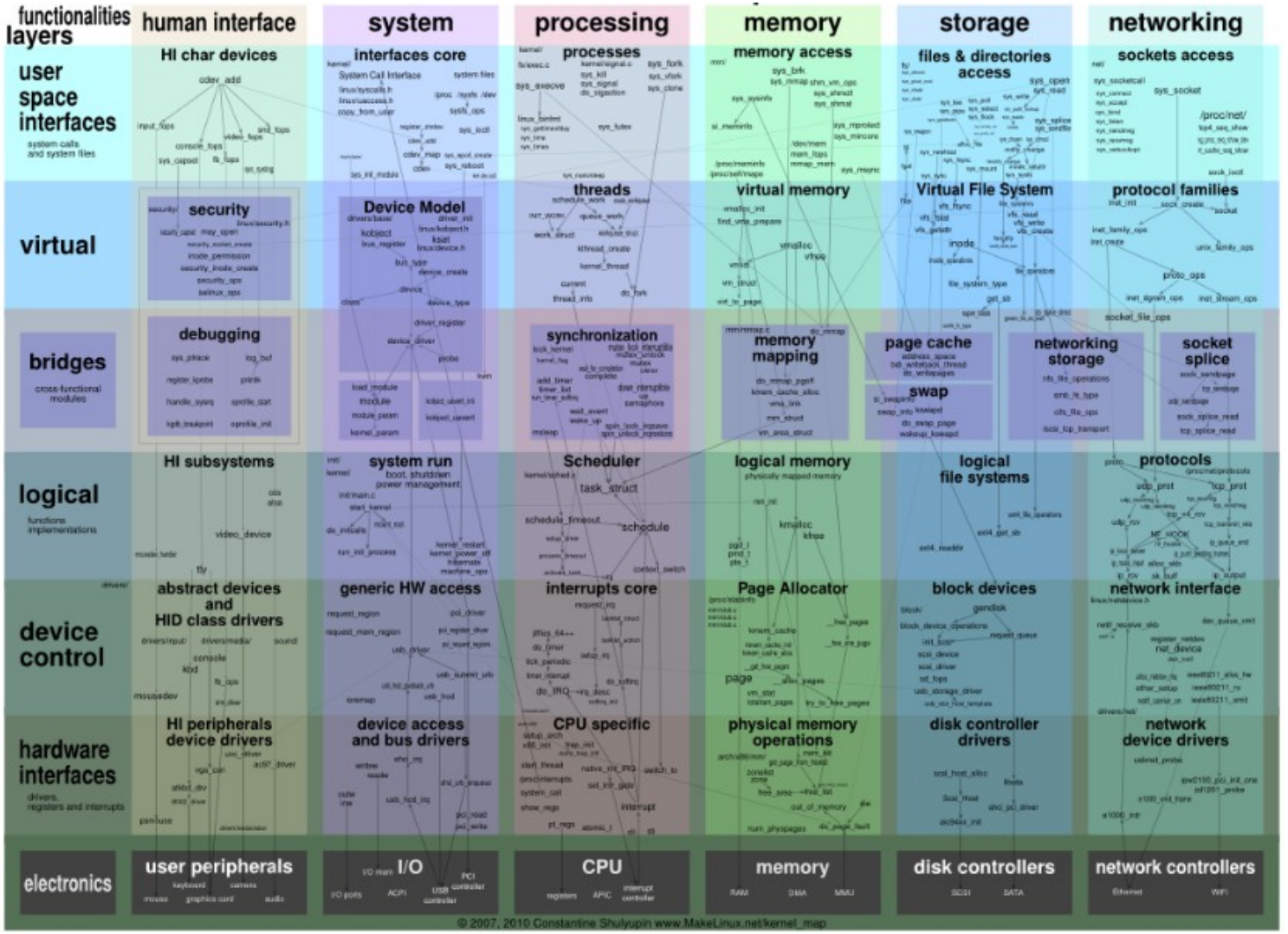
```
U-Boot 2009.08 (Oct 15 2010 - 13:03:11)

CPU:   Freescale i.MX51 family 3.0V at 800 MHz
mx51 pll1: 800MHz
mx51 pll2: 665MHz
mx51 pll3: 216MHz
ipg clock     : 66500000Hz
ipg per clock : 665000000Hz
uart clock    : 66500000Hz
cspi clock    : 54000000Hz
Board: MX51 BABBAGE 3.0 [POR]
Boot Device: MMC
I2C:   ready
DRAM:  512 MB
MMC:   FSL_ESDHC: 0, FSL_ESDHC: 1
In:    serial
Out:   serial
Err:   serial
PMIC Mode: SPI
Net:   got MAC address from IIM: 00:04:9f:01:13:34
FEC0 [PRIME]
Hit any key to stop autoboot:  0
BBG U-Boot >
```

Role of the kernel



Kernel architecture

- Interactive map of Linux kernel
  http://www.makelinux.net/kernel_map



Main kernel subsystems

- Interface with applications (through system calls)
- Process management
- Memory management
- Filesystems
- Networking
- Device drivers

Drivers

- Usually part of the kernel space
- Can be linked statically with the kernel (*built-in*) or dynamically (*modules*)
- Different types of drivers:
  - Block
  - Character
  - Specific interfaces (e.g. network)
- Many drivers are available in the kernel tree. Check before you start coding!

Why using open-source drivers?

- Licensing issues
  - Proprietary drivers cannot be statically linked with the kernel (GPLv2)
  - GPL drivers can freely reuse GPL code
- Mainline drivers are constantly being reviewed and tested

  => Stability

  => Security
- The kernel internal API changes all the time (Why?)

  When a driver is part of the mainline, these changes are taken care of.

The root filesystem

- The root filesystem is where all the files contained in the file hierarchy (including device nodes) are stored
- Many different components:

Filesystem conventions

- /bin: Essential command binaries
- /dev: Devices
- /etc/: Configuration files
- /home/: User's directories
- /lib/: Libraries
- /usr/bin/ and /usr/lib: Non-essential command binaries and libraries
- /sbin/: Essential system binaries
- /proc: Information about processes
- /sys/: Kernel devices and drivers information

Initialization scripts

- After the kernel is done initializing the drivers, the system typically calls initialization scripts to:
  - Start services in the background
  - Create a debug console
  - Start your application
- If you use a build system (e.g. LTIB), you will get a default set of scripts
- Usually written using shell scripts
- You want to customize these scripts to:
  - Remove unnecessary features and reduce boot times
  - Implement your own scenarios (recovery, normal boot...)

Busybox

- **BusyBox** combines tiny versions of many common UNIX utilities into a single small executable. e.g.
  - shell
  - coreutils (*cat, dd, head, tail...*)
  - process utilities (*ps, top...*)
  - … and much more

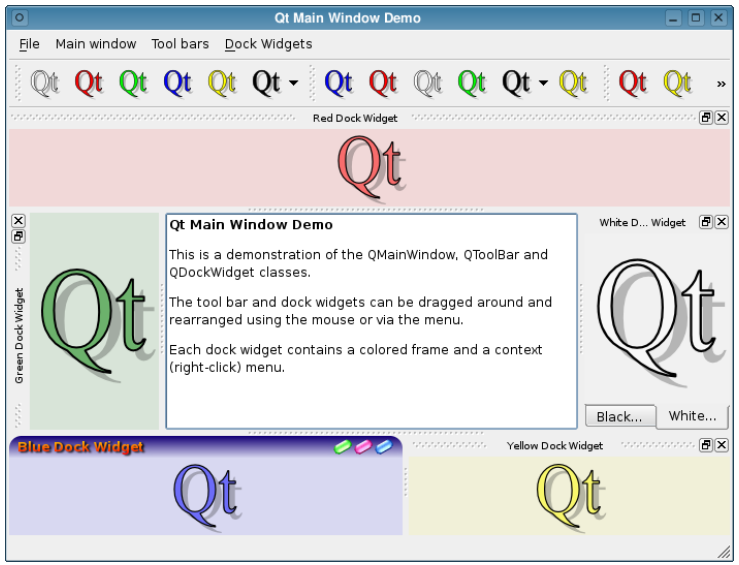
- Less features than the standard GNU implementations...
- … but must often sufficient for embedded usage

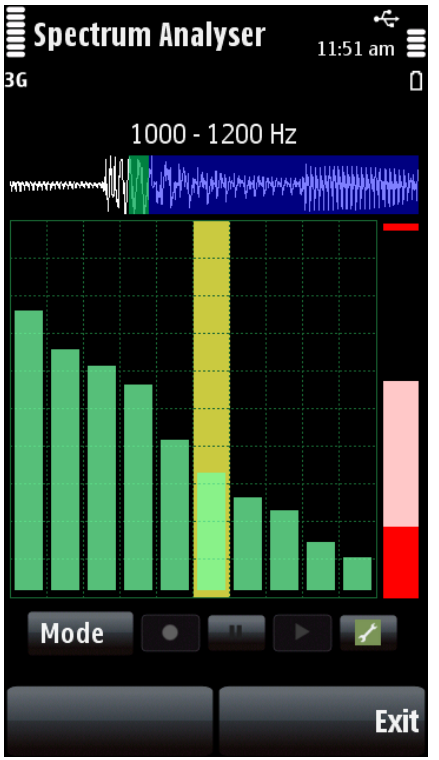
- Features can be enabled or disabled at build-time

Network utilities

- **Dropbear**
  - SSH server and client
  - Small memory footprint (can be configured at build-time)
- **Busybox**
  - FTP/TFTP/DHCP/Telnet servers and clients
  - HTTP server
  - Basic network utilities (ping, ifconfig...)
- Most desktop/server projects can be ported easily to embedded devices (e.g. Apache)
- Alternative lightweight implementations often exist

Graphical interfaces: Qt



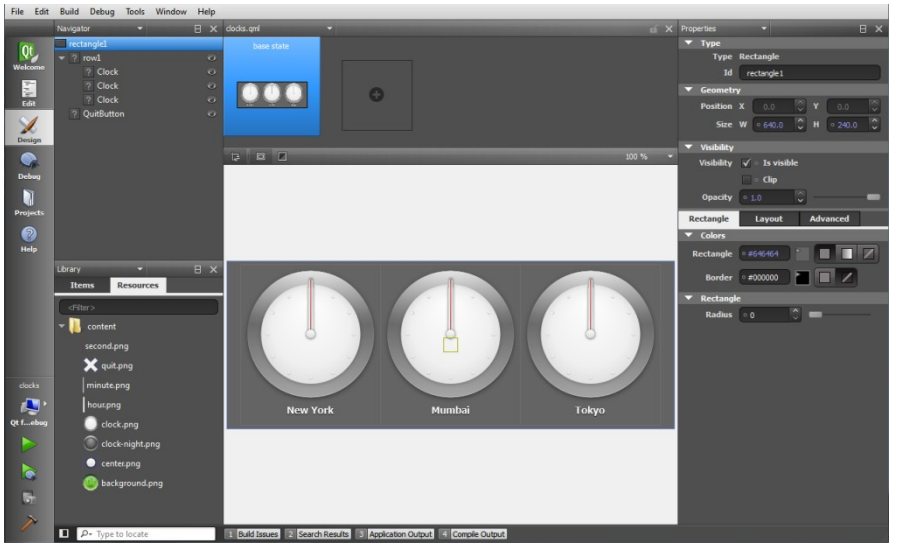


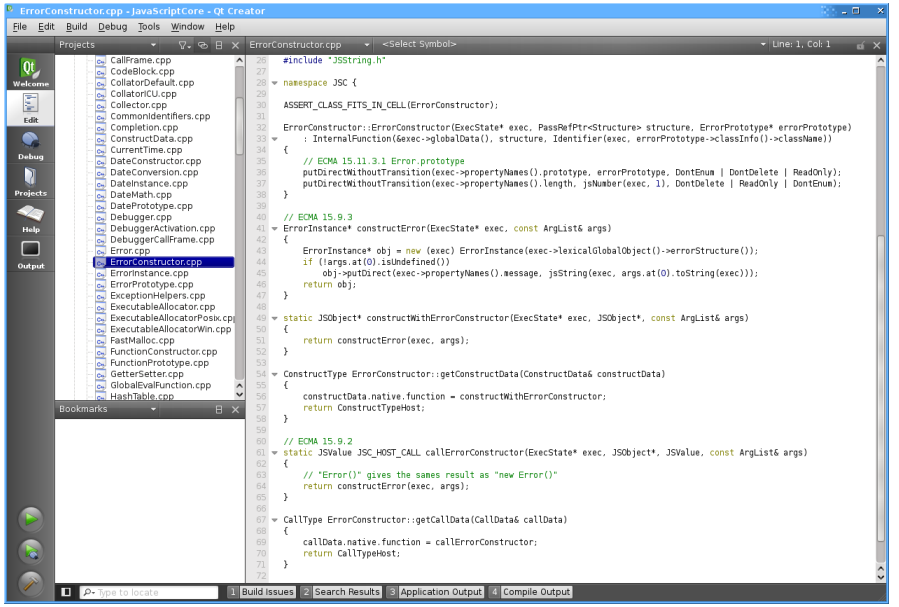
Graphical interfaces: Qt

- Cross-platform application and UI framework
  - Maintained by Nokia
- Written in C++
- Features:
  - GUI
  - XML parsing
  - Database access
  - File handling
  - Internationalization support
  - Graphics hardware acceleration
- Native performance
- Easy to prototype on PC

Graphical interfaces: Qt Creator

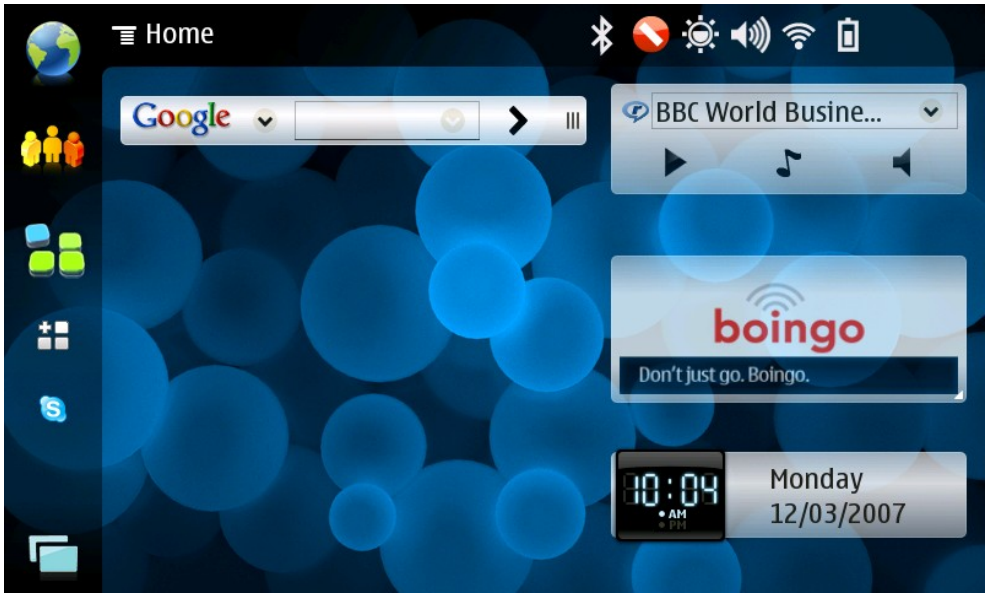
- Create interfaces with Qt Creator:
  - C++ and JavaScript code editor
  - Integrated UI designer
  - Project and build management tools
  - gdb and CDB debuggers
  - Support for version control
  - Simulator for mobile UIs
  - Support for desktop and mobile targets

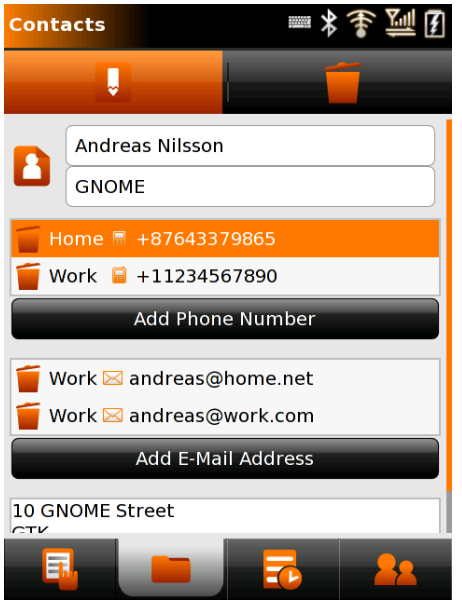




Graphical interfaces: GTK+

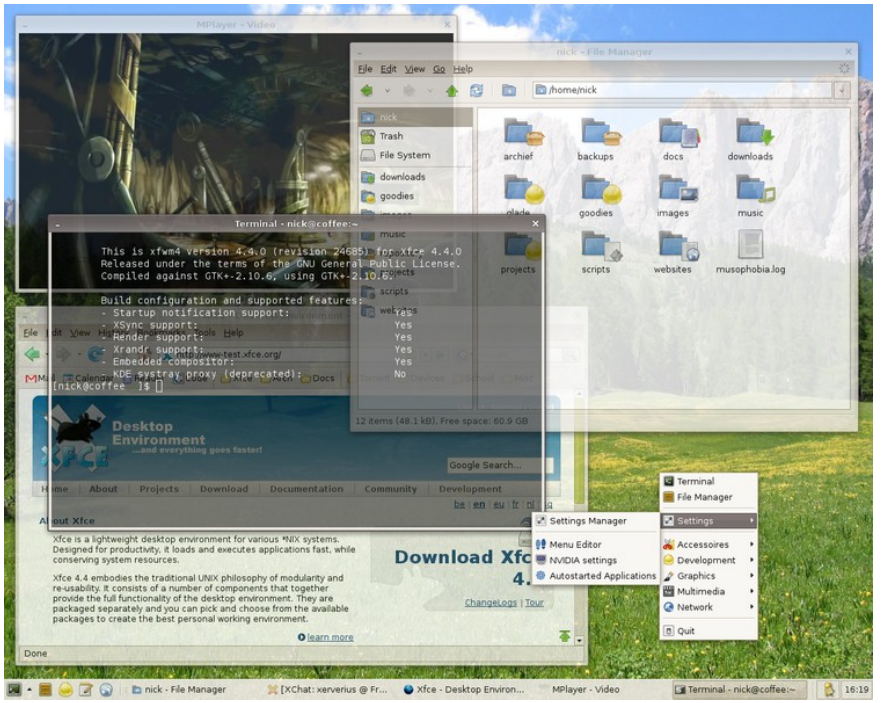
- Toolkit for creating graphical user interfaces
- Cross-platform
- Available in many languages (through bindings)
- Used by GNOME, Maemo, Openmoko, OLPC





Graphical interfaces: Other solutions

- Direct access the framebuffer device
- Use libraries that provide a framebuffer abstraction (DirectFB, SDL)
- X Window
  - Lightweight implementations exist e.g. KDrive)



xfce on X Window

Multimedia on Linux

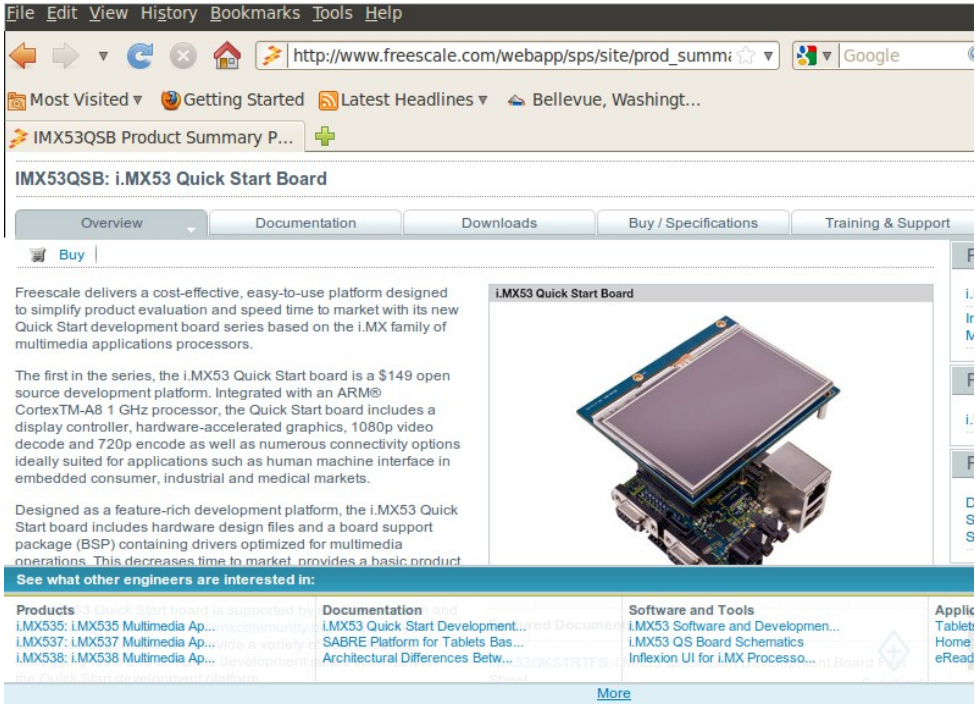
- **Audio**
  - ALSA (Linux kernel + User-space library)
  - Pulseaudio (sound server in user-space)
- **Media streaming**
  - Gstreamer
    - Multimedia framework
    - Pipeline architecture
    - Can be used to easily create multimedia applications
  - ffmpeg
  - Cross-platform solution to record, convert and stream audio and video
- **Graphics**
  - DirectFB, SDL, OpenGL
- **Video capture and broadcast**
  - V4L2 (Video for Linux)
- **Pictures**
  - libpng, libjpeg

Databases: SQLite

- SQLite is a software library that implements a transactional SQL database engine
- Most widely deployed SQL database engine in the world (Android, Linux, iOS...)
- Well-suited for embedded systems:
  - Self-contained
  - Serverless (it is a library, not a process)
  - The entire database is stored in a file
  - Zero-configuration
- The source code for SQLite is in the public domain

Web browsers

- Firefox
- Webkit
  - Also supplied with Qt and Android
- Dillo (lightweight browser)



Toolchain

- The toolchain is the set of tools that is used to produce binaries
  - Compiler: *gcc, g++*
  - Libraries: *glibc, libstdc++*
  - Binutils: *ld* (linker), *as* (assembler), binary manipulation programs

- Typical scenario:
  - You are compiling using your host machine (e.g. x86)
  - You are compiling for an ARM target

    => Cross-compilation (requires a specific toolchain)

Choosing a toolchain

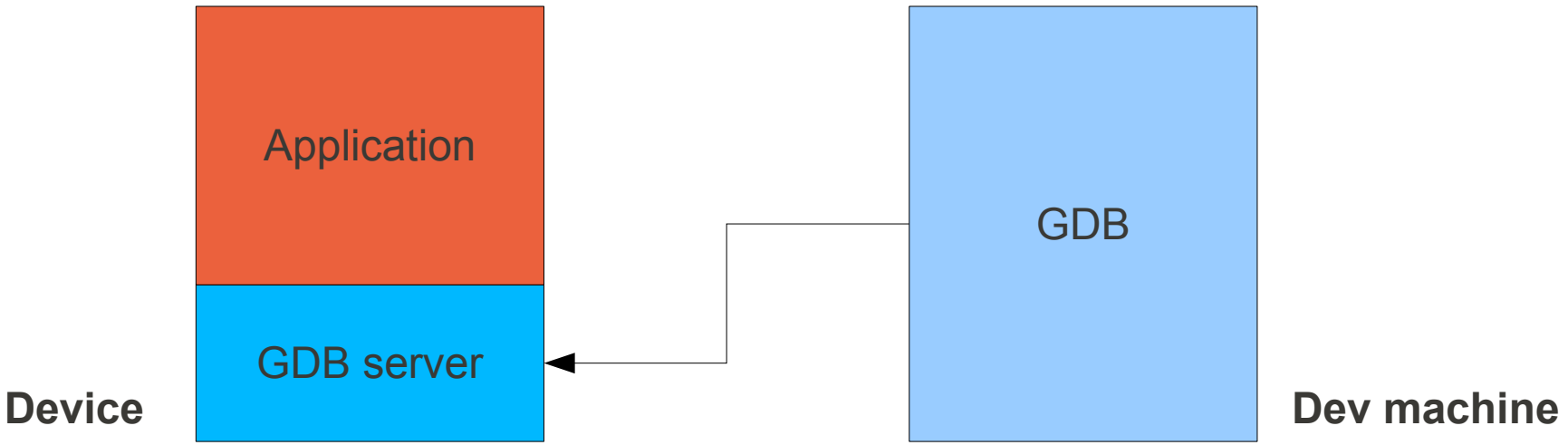
- Build it manually
  - Tedious and difficult
  - Interesting read: Linux from scratch
- Use toolchain-building tools (e.g. crosstools-ng)
  - Also requires a fair amount of knowledge
- Use prebuilt toolchains (e.g. CodeSourcery)
  - Good way to obtain the latest toolchains
  - Easy to integrate with LTIB
  - Latest toolchains are not as well tested
- From LTIB
  - LTIB provides a toolchain tested by Freescale
  - Fastest way to get up and running

Choosing a C library

- The C library is a fundamental component of a Linux system
  - Many Linux programs are written in C
  - Implements all the C functions declared in standard headers (e.g. *printf()*)
- Built and provided with the toolchain
- Different implementations exist:
  - glibc: aims for compatibility and standard compliance
  - uclibc: smaller, configurable implementation
  - ...

Development tools: GDB

- GNU Project debugger
- Most popular debugger for Linux
- Supports many languages (including C and C++)
- Interface
  - Command prompt
  - Graphical frontends (ddd, Eclipse, ...)
- Remote debugging:



Development tools: Serial console

- Debugging on embedded devices is usually done using a serial link to your device:
  - Read Kernel messages
  - Interact with a remote shell
  - Log application messages

- **Minicom**
  - Highly configurable
  - Supports logging to a file, file transfers
  - Can also be ported to devices
  - GUI alternatives

Development tools: TFTP and NFS

- During development, you will likely modify the kernel and/or the contents of the root filesystem

  Having to flash the software for every modification can be very tedious!

- **Solution**:
  - Download your kernel from your host over TFTP
  - Mount your root filesystem using NFS

  => All the files are on your dev machine. No need to reflash everything

Development tools: QEMU

- Open source machine emulator and virtualizer
- Two different operating modes:
  - User mode emulation, which allows you to run a simple cross-compiled executable
  - Full system emulation, which emulates a full system including the corresponding hard disk image.
- Good performance
- Can be used to test applications on a virtual device
- ARM support

Development tools: Misc

- Profiling:
  - oprofile (system-wide profiler)
  - gprof, gcov (for applications)
- System calls tracing:
  - strace
- Library calls tracing:
  - ltrace
- Dynamic analysis (memory leaks, cache, profiling):
  - valgrind
- And many more...

Development tools: Version control

- Version control systems are typically used for:
  - Your own project
  - Getting the sources of Open Source projects

- Many different solutions exist:
  - CVS, SVN
  - Mercurial
  - Git
    - Used for many different projects now, most notably the Linux kernel
    - Very powerful
  - ...

Building a system

- Building a system requires to:
  - Select a toolchain
  - Select the different packages that will run on the target
  - Configure and build these packages
  - Deploy them on the device

- Different ways to build a system:
  - Manually (creating your own scripts): tedious, harder to reproduce builds on other machines
  - Using complete distributions, e.g. Ubuntu/Debian ARM: one size fits all – harder to customize
  - Using build systems, e.g. OpenEmbedded, **LTIB**: easy to customize and reproduce builds

Writing applications

- **Portability**
  - Writing code for Embedded Linux mostly similar to writing desktop applications
- **Prototyping**
  - Targeting your Linux host development machine
  - Using QEMU to simulate a device
- **Leverage** existing software (frameworks, libraries, tools...)
  - Avoid reinventing the wheel!
- **Many languages** can be used: scripts, C, C++, Java
- **Tools**
  - Editors/IDEs: vi, emacs, Eclipse CDT...
  - Build system: Makefile, autotools, cmake...

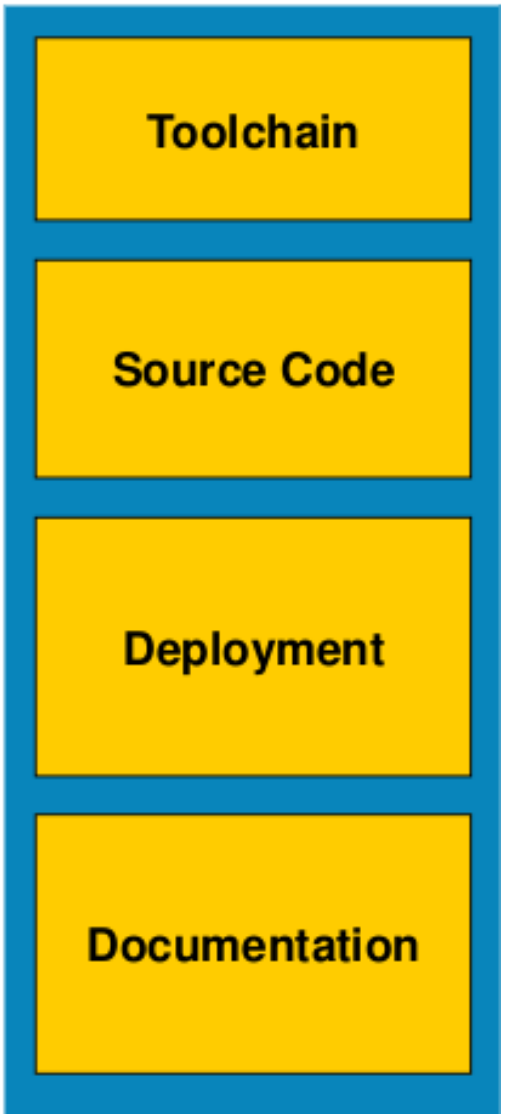
Using Linux on the i.MX53 Quick Start Board

freescale™ semiconductor

Adeneo Embedded

Freescale Linux BSPs

- Typically contain:
  - Build system:
    - LTIB
  - Toolchains
    - Compilers/Linkers
  - Source Code
    - Bootloaders (most)
    - Kernel and drivers
    - Applications
  - Deployment
    - Automated or instructions
  - Documentation
    - BSP usage and hardware docs
    - Device driver docs



Freescale Linux BSPs: key features

- Easy to install and use
- Well documented

- Leverage drivers from the whole i.MX family

- Well tested drivers
- Support for hardware acceleration, codecs...

- Recent kernel

- Support reference designs

Freescale Linux BSPs

- BSPs are starting points (also for our 3rd party BSP Linux vendors)
  - Provide basic functionality on listed set of devices
  - They are not production tested or fully optimized
  - They are not intended to be final solutions
  - Bugs are verified and accepted
  - Fixes/Patches are worked into future revs of the specific BSP

- Professional Services / Third party developers
  - Feature requests or driver enhancement
  - Training
  - Driver / Application development
  - Support

Typical software development cycle

- Build and run the BSP on a reference design (e.g. Quick Start Board)
- Prototype your application on the reference board

- Adapt the BSP for your custom design
  - Write custom drivers
  - Configure the kernel
  - Customize the root filesystem

- Port your application to your custom design

Getting the Linux BSP

- Get the latest BSP, documentation and application notes from Freescale's website:
  - i.MX53 Software and Development Tool Resources

Building your i.MX53 based design just got easier. With the introduction of our Smart Application Blueprint for Rapid Engineering (SABRE) platform for tablets based on i.MX53 reference design and our new low cost i.MX53 Quick Start board you now have two development platforms to get your design to market quickly. These pages contain the software, design and development tools you need to help accelerate your development cycle.

The latest in a series of premiere market-focused reference designs, the SABRE platform for tablets showcases the well integrated and high performing i.MX53 series of multimedia applications processors based on the ARM® Cortex™-A8 with core speeds up to 1.2 GHz.

The first in the series, the i.MX53 Quick Start board is a $149 open source development platform that supports the features of the i.MX53 applications processor and includes support for a VGA display as well as optional add-on boards to support LVDS,LCD and HDMI displays. The i.MX53 Quick Start board and the provided Linux® Board Support Package (BSP) decreases time to market by providing a basic product design and serving as a launching point for more complex designs.

With production-ready software components, an optimized OS and a system-validated BSP, designers have the tools to test and maximize the performance of the applications they have developed.

Supported Device Families

- i.MX53 Processors

See All

i.MX53 Current Software Updates and Releases

Software: Quick Start Board

- **Linux**
  - i.MX53 Linux Source Code
  - i.MX53 Linux Multimedia Codecs Source Code
  - i.MX53 Linux Demo Images
- **Supporting 3rd Party BSPs for Quick Start Board:**
  - Android
  - Windows Embedded Compact

Software: SABRE for Tablet

- **Android**
  - i.MX53 Android 9.4 Source Code
  - i.MX53 Android 10 Source Code

Documentation

- **Quick Start Board**
  - i.MX53 Quick Start Board Fact Sheet

What is LTIB?

- **L**inux **T**arget **I**mage **B**uilder is a tool created by Freescale, that is used to build Linux target images, composed of a set of packages
  - A mechanism to deliver Linux board support packages (BSP)
  - A wrapper around tool chains and standard Linux commands (*cp, make, objcopy, tar, gcc, …*)

- **Provides**:
  - A known working configuration for a target board
  - Functionality to configure and build Linux system components (kernel, bootloader, busybox, …)
  - Functionality to configure and build the Linux target system

    (network configuration, type of file system to use, ...)

LTIB philosophy

- LTIB has been released under the terms of the GNU General Public License (GPL)
- Provides more than 200 applications – originating from open source projects
- "Standard Linux" look and feel (*make menuconfig*)

- Typical LTIB cycle (only requires one single command):
  - Download packages from the network/Internet/local repository
  - Build kernel, boot loader and application packages from source
  - Deploy built packages to a root file system (RFS) tree
  - Prepare appropriate kernel or RFS image files ready for network or flash-based use on the embedded target board
  - Capture source modifications into patches and auto update .spec files

LTIB packages for the i.MX53 QSB

- Optimized toolchain for the Cortex-A8 CPU
  - Supports VFP, NEON
- Linux kernel 2.6.35
- U-Boot
- Packages:
  - Base tools: BusyBox, Dropbear, ...
  - Frameworks: Qt, GTK, ...
  - 2D and 3D acceleration drivers
  - Wireless connectivity drivers
  - … and many more! (We will have a look at them during the hands-on labs)

How to find help

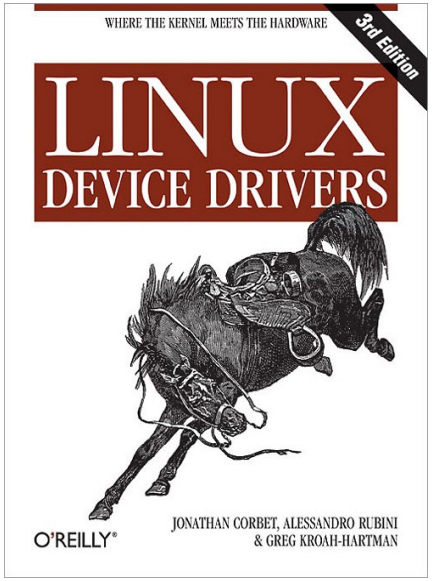
- From Freescale:
  - Documentation
  - Application note
- 3rd parties (e.g. Adeneo Embedded)
- Community websites
  - i.MX Community: lots of information, trainings, vibrant community
  - imxdev.org

- If you encounter an error, just look it up. In most cases, you will directly find a solution

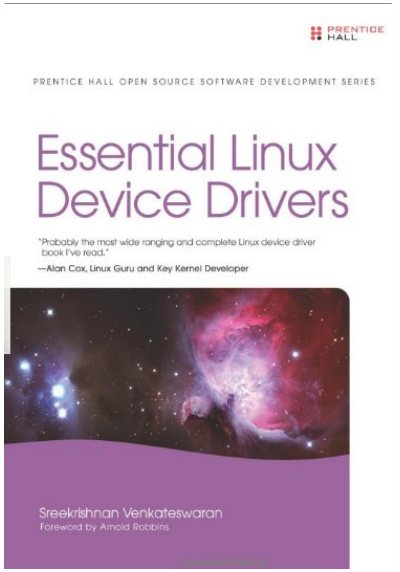
- A lot of information is readily available about the different components of your Linux system

Books: Driver development

- Linux Device Drivers, Third Edition
  - Available online for free: http://lwn.net/Kernel/LDD3/
  - Advanced programming



- Essential Linux Device Drivers
  - Easier to read
  - Covers many types of drivers



LAB: Preparing the i.MX53 Quick Start Board

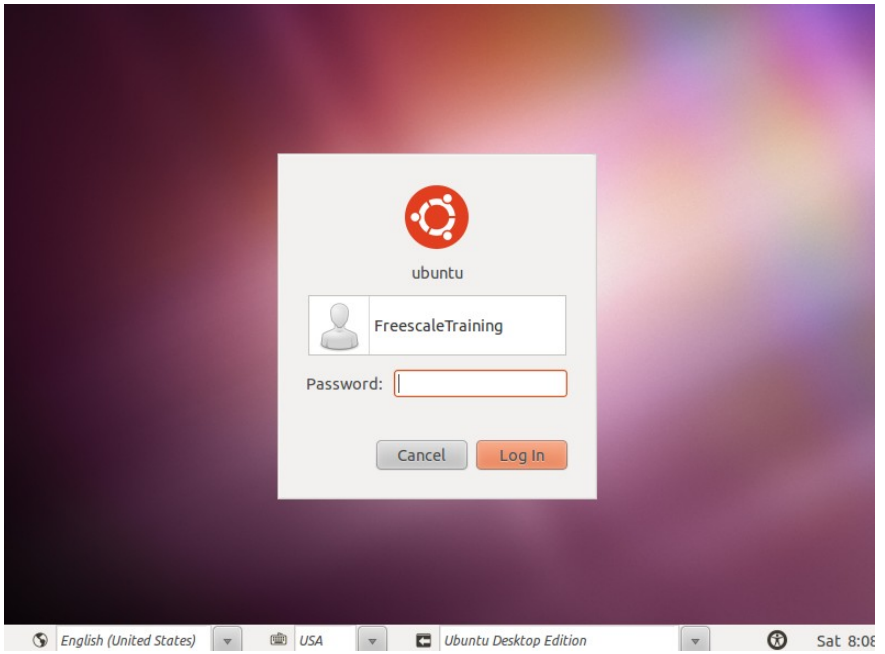
freescale™ semiconductor

Adeneo Embedded

Using the virtual machine

- The Linux (Ubuntu 10.10) development environment is run from within a virtual machine (VMware)
- The performance is not as good as a native OS but remains acceptable
- Start VMWare and load the VM called "*Ubuntu 10.10*"
- Skip the warnings about unconnected devices (if any)

- Login
  - User: freescale
  - Password: freescale
- The admin password is also 'freescale'



Using the shell

- The shell can be started using the quicklaunch bar or through the menu (*Applications* | *Accessories* | *Terminal*)
- Tab Completion is your friend. Start typing a file name, and hit TAB and it will fill in paths
- Everything is done in the "*home*" directory. This is */home/freescale*
  - If you want to get back to this location quick, type: *cd ~*
- To learn your current directory, type: *pwd*
- It is very common to type commands over and over again. The Up Arrow is command history
- You can even search for a command by doing: CTRL-R and start typing the command. It will fill it in!
- If you type: *history,* you will get a list of all typed commands. To execute one, type : *!#* (Where # is the command number you want to execute)

Assembling your i.MX53 kit

- Note where the different buttons are located:
  - Power and reset
  - User buttons
- Connect the LCD screen (see picture)
- Locate the MicroSD Card slot
- Connect 5V power

LAB: Building images with LTIB

freescale™ semiconductor

Adeneo Embedded

Configuring LTIB

- Open a terminal

  Note: In this training “>” indicates the command prompt on your host. Do not type it in.

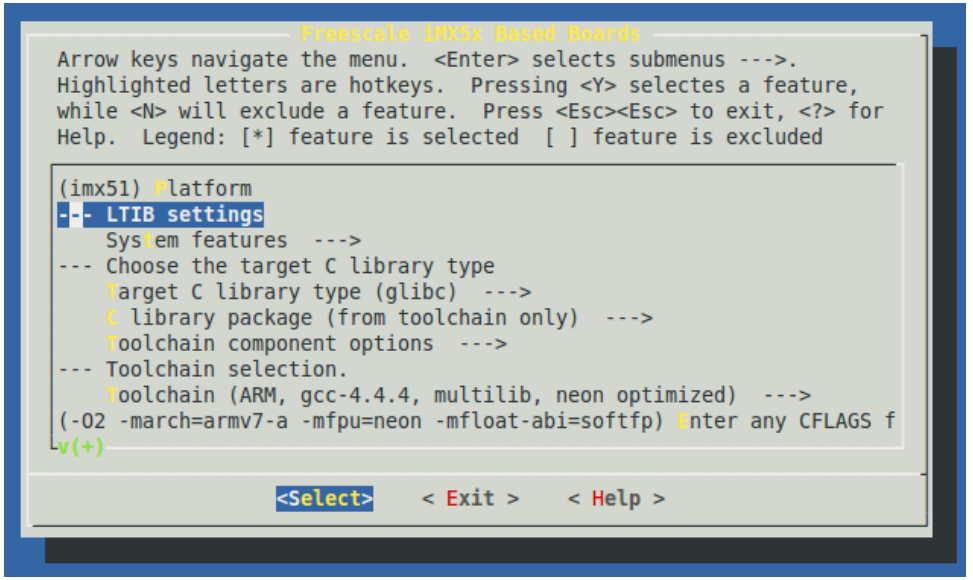
- From the terminal, issue the following commands:

```
> cd ~/training_mx53/linux/ltib
> ./ltib -c
```

- Configuring LTIB only requires a single command.
  - Note: For these labs, LTIB has been preconfigured for the Quick Start Board, but installing LTIB is only a matter a few simple steps.

Configuring LTIB

- The LTIB main menu will appear
- Take some time to browse through the different menus, especially:
  - The toolchain and C library options
  - How can you build different kernel versions (including the bleeding-edge git version)
  - The many items of the package list
  - The *Target Image Generation* options



Building your image

- Select *tslib* in the Package list (use the arrows and space keys to navigate)
- When you are done, select "*Exit*" in the main menu
- LTIB will save your configuration and build your image.

  You will also notice that tslib is getting built in the process (the other packages have been precompiled for this training)

```
Processing: tslib
==================
Build path taken because: no prebuilt rpm,

rpmbuild --dbpath /home/rem/training_mx53/linux/ltib/rootfs//var/lib/rpm --tar
t arm --define '_unpackaged_files_terminate_build 0' --define '_target_cpu arm
--define '__strip strip' --define '_topdir /home/rem/training_mx53/linux/ltib/
m' --define '_prefix /usr' --define '_tmppath /home/rem/training_mx53/linux/lt
/tmp' --define '_rpmdir /home/rem/training_mx53/linux/ltib/rpm/RPMS'  --define
_mandir /usr/share/man' --define '_sysconfdir /etc' --define '_localstatedir /
r' -bb --clean --rmsource  /home/rem/training_mx53/linux/ltib/dist/lfs-5.1/tsl
/tslib.spec
Building target platforms: arm
Building for target arm
Executing(%prep): /bin/sh -e /home/rem/training_mx53/linux/ltib/tmp/rpm-tmp.72
1
```

Building your image

- Once everything has been built, you will find the output in *~/training_mx53/linux/ltib*
  - *rootfs/boot/u-boot.bin*: bootloader image
  - *rootfs/boot/uImage:* kernel image
  - *rootfs*: this is the unpacked filesystem
  - *rootfs.jffs2*: packed image (for flash devices here – can be configured in LTIB)

- These images can be flashed to the SD Card
  - Using standard Linux tools (dd, fdisk...)
  - Using Freescale's dedicated tools

DEMO: Using Ubuntu with the Quick Start Board

freescale™ semiconductor

Adeneo Embedded

Ubuntu demo

- This demo uses the Ubuntu image that you can download from the i.MX53 Quick Start Board page
- Showcases the graphics capabilities of the i.MX53
  - OpenGL (**Lots of information and tutorials in the GPU SDK**)
  - Video decompression and resizing
  - Flashed to the SDCard using *flash_ubuntu_gfx.sh* (script written for these labs)

Extra LAB (homework): Using USB gadgets

freescale™ semiconductor

Adeneo Embedded

LAB: using the USB gadgets

- Configure LTIB and set “Configure the kernel”
- When the kernel configuration menu appears, select *Ethernet Gadget* in *Device Drivers* | *USB support* | *USB Gadget Support.* Choose to build as a module (*M*)
- Let LTIB build the kernel, flash it and boot the device
- From the serial console, load the ethernet gadget module:

  # modprobe g_ether
- Connect the micro USB port to your PC
- Configure the USB ethernet connection on your device:

  # ifconfig usb0 up 192.168.4.2
- Configure the USB ethernet connection on your PC:

  > sudo ifconfig usb0 up 192.168.4.1
- Your PC is now connected to your device over USB!

- Other USB gadgets:
  - USB mass storage
  - USB video
  - … and many more (look at the kernel options)

Lunch break

freescale™ semiconductor

Adeneo Embedded


freescale™
semiconductor


deneo
Embedded

Android on the i.MX53 Quick Start Board

freescale™ semiconductor

Adeneo Embedded

Agenda

- General presentation of Android
- Lab: Flashing and using Android on the i.MX53 Quick Start Board
- The Android architecture
- Writing applications for Android
- Lab: Hello World using the emulator
- Android on the i.MX53
- Lab: Using ADB to connect to the Quick Start Board
- Lab: Deploy an debug an application on the device
- The SDK and the NDK

These slides are based on Eric Gregori's Android training: Hands-On Android Applications Development Training with Eclipse and the i.MX51 ARM Cortex-A8

Introduction to Android

freescale™ semiconductor

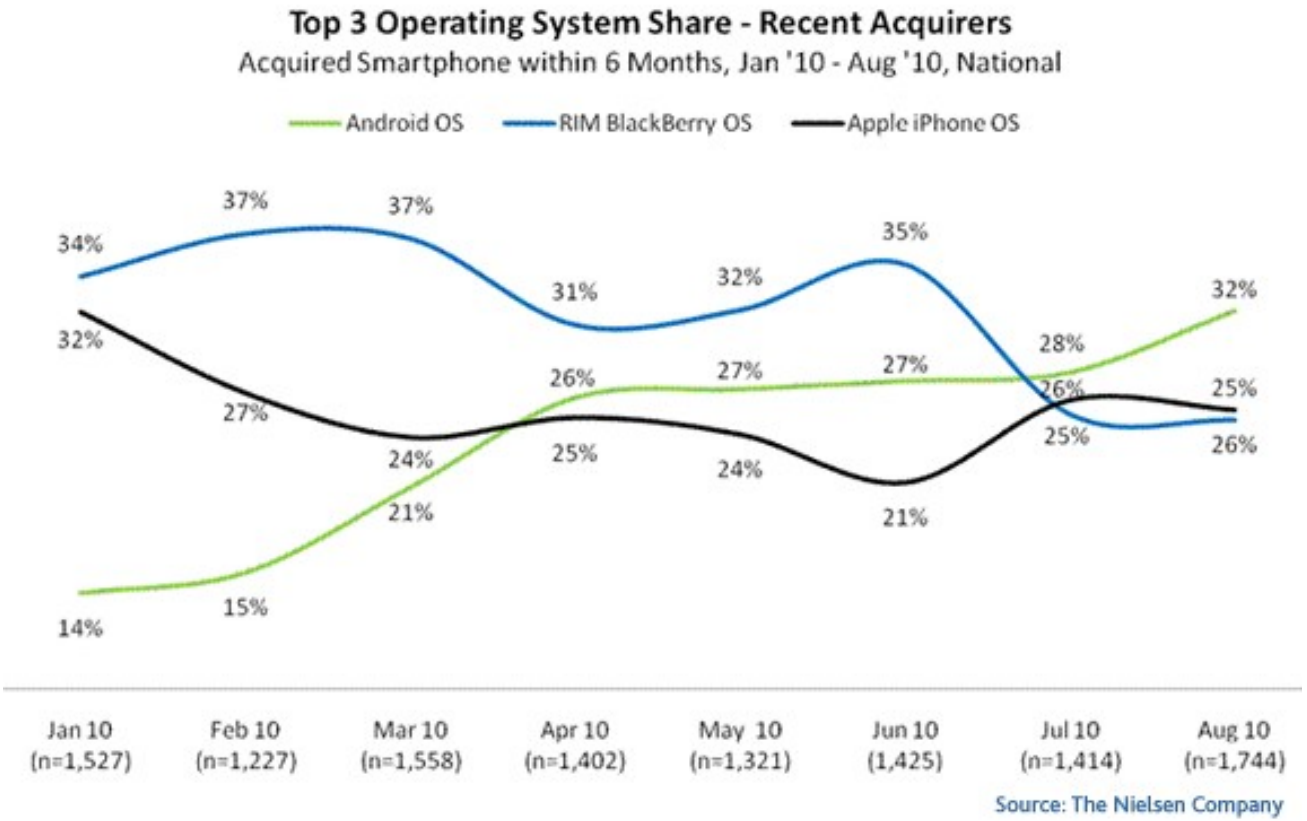
Adeneo Embedded

Introduction to Android

- Android is a Operating system created by Google for smartphones and tablets
- The first Android based phone was released in October of 2008
- Android based phone sales surpassed Apple and Blackberry in July of 2010

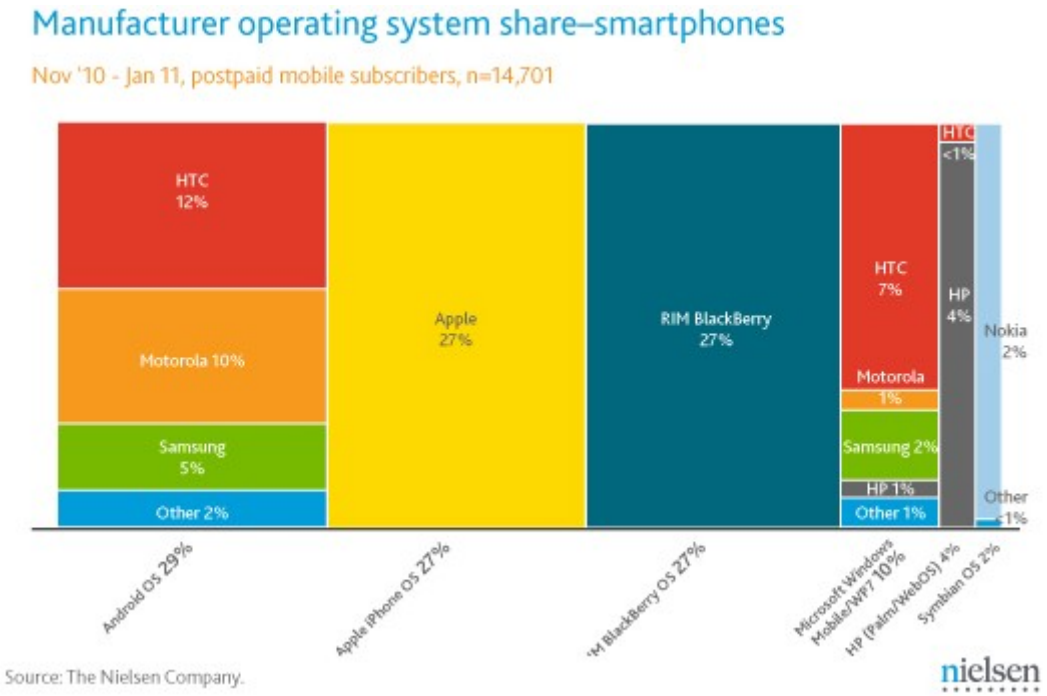
Android market share

- Leading OS for recent smartphones



Android market share

- Very quick growth: 3.9% in 2009, 22.7% in 2010!



Worldwide Smartphone Sales to End Users by Operating System in 2010 (Thousands of Units)

Company	2010 Units	2010 Market Share (%)	2009 Units	2009 Market Share (%)
Symbian	111,576.7	37.6	80,878.3	46.9
Android	67,224.5	22.7	6,798.4	3.9
Research In Motion	47,451.6	16.0	34,346.6	19.9
iOS	46,598.3	15.7	24,889.7	14.4
Microsoft	12,378.2	4.2	15,031.0	8.7
Other Oss	11417.4	3.8	10432.1	6.1
Total	**296,646.6**	**100.0**	**172,376.1**	**100.0**

Source: Gartner (February 2011)

Android phones

- Typical features:
  - 1 GHz ARM Cortex-A8
  - 16 GB Flash
  - 256 MB RAM
  - 3-axis accelerometer
  - GPS
  - Compass
  - Cameras
  - 800x480 LCD touchscreens
  - WiFi, Bluetooth
  - USB

Android tablets

Motorola Xoom

Galaxy Tab

Other Android products

Archos32 Android based Internet Tables

- NOT A CELL PHONE – no monthly fees
- $149.00
- ARM Cortex A8 at 800 MHz – 8GB of flash
- VGA Camera
- Touch screen, 400x240 pixels (WQVGA), 3.2" TFT
- LCD, 16 million colors
- WiFi (802.11 b/g/n) + Bluetooth
- 3-Axis accelerometer

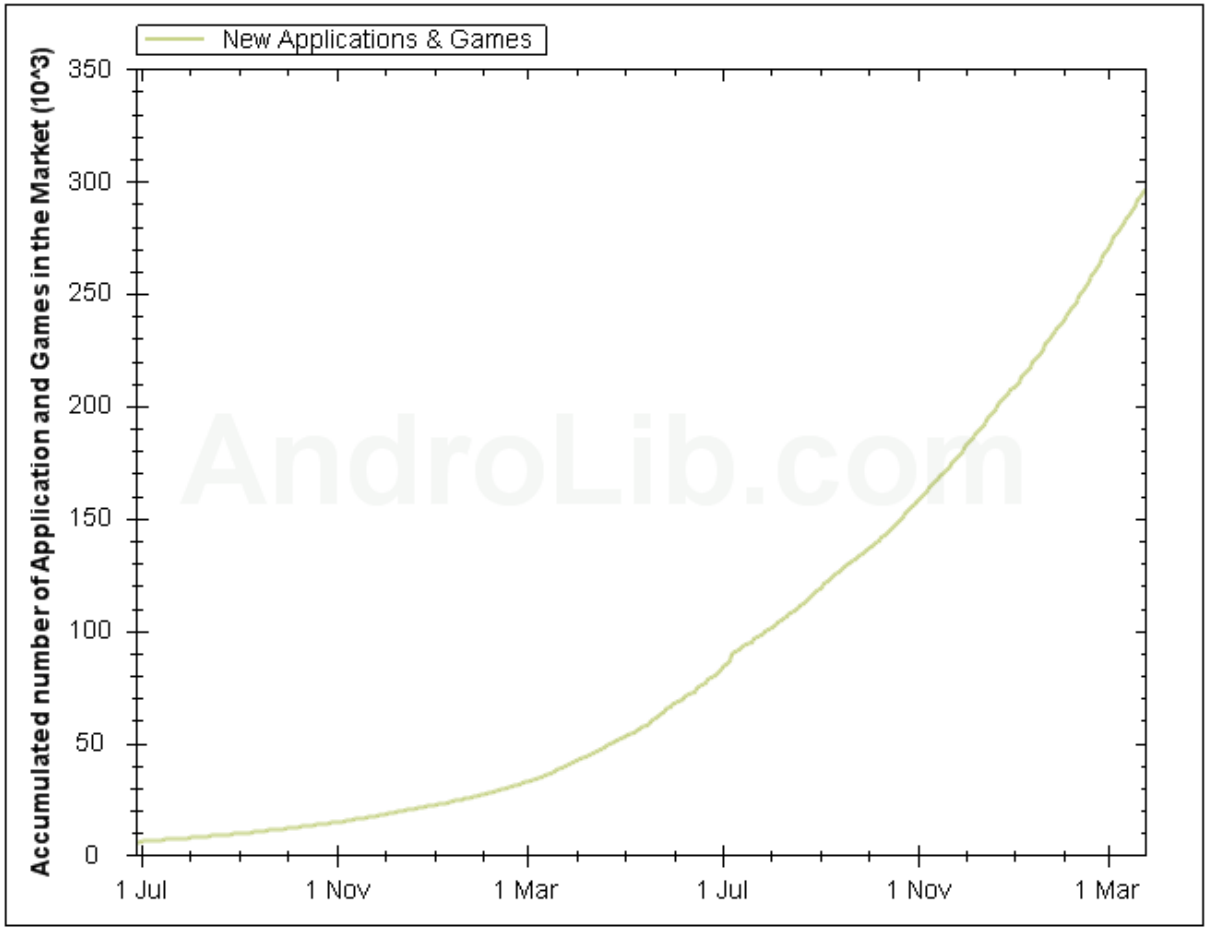
Android Market

- In 2011 (source: http://www.androlib.com/appstats.aspx)

3,799,156,143

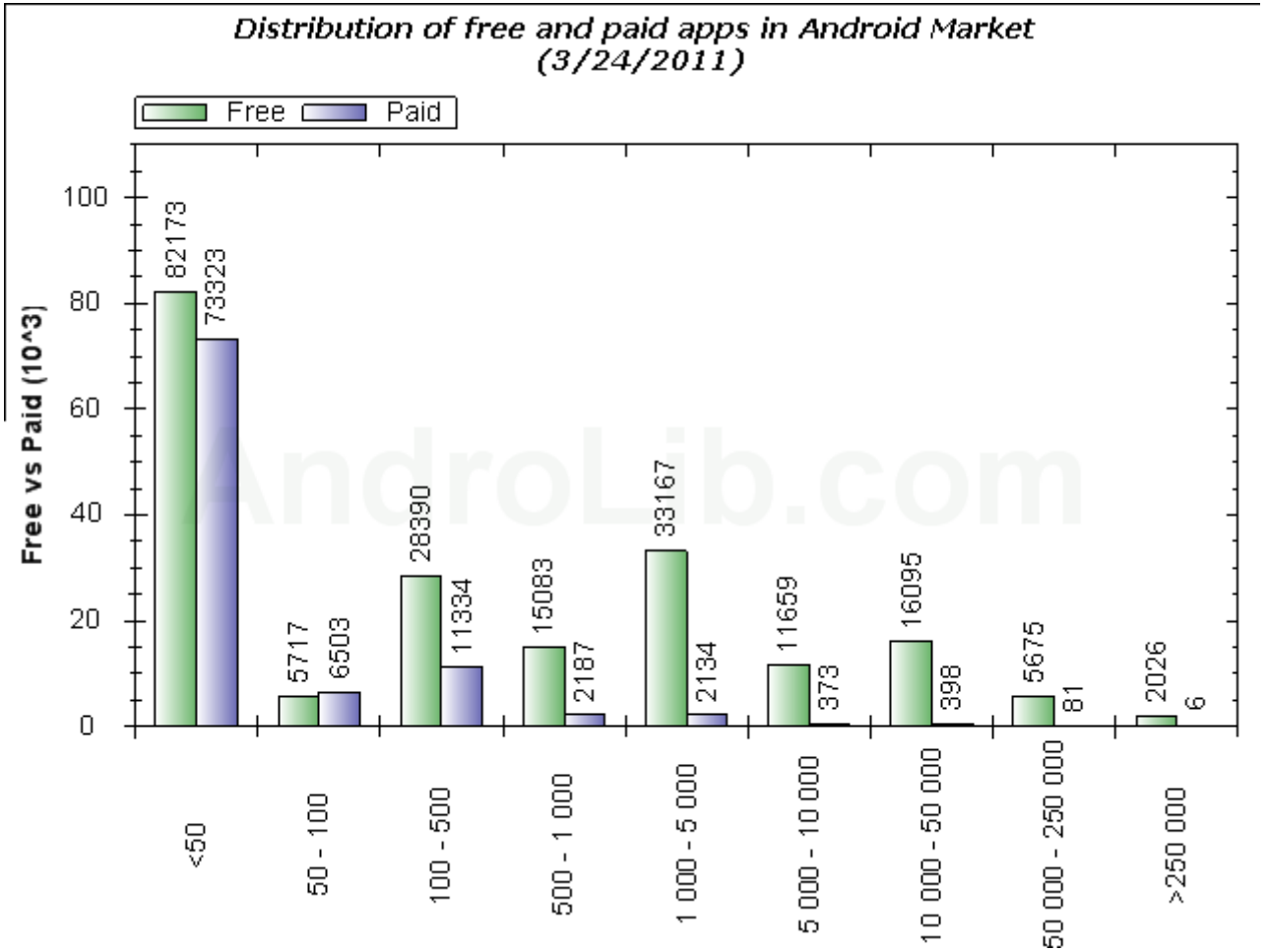
Estimated number of Applications downloaded in the Android Market

Accumulated number of Application and Games in the Android Market



Android Market

- In 2011 (source: http://www.androlib.com/appstats.aspx)

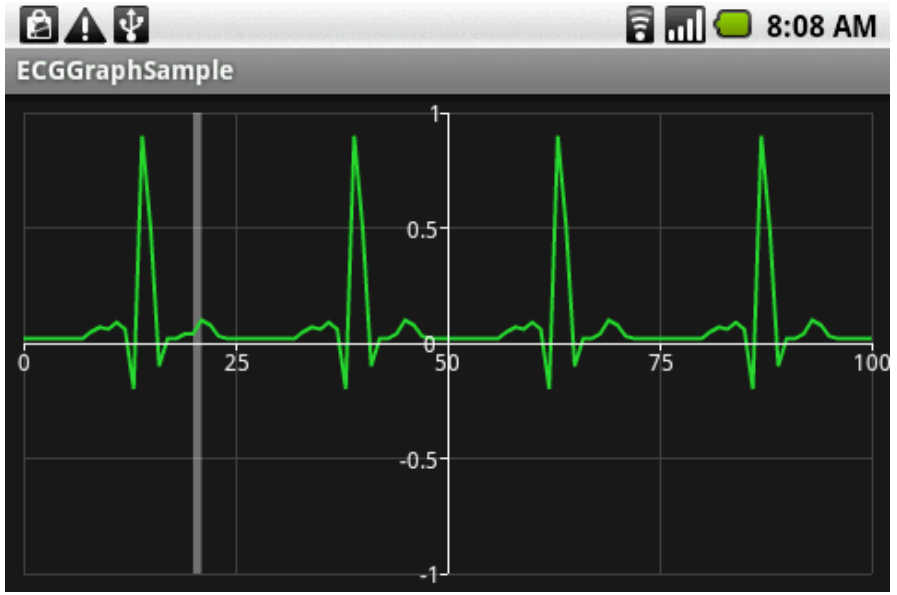


Interesting applications

- Games and emulators

Interesting applications

- Sleep Monitor Application
  - http://www.artfulbits.com/Products/

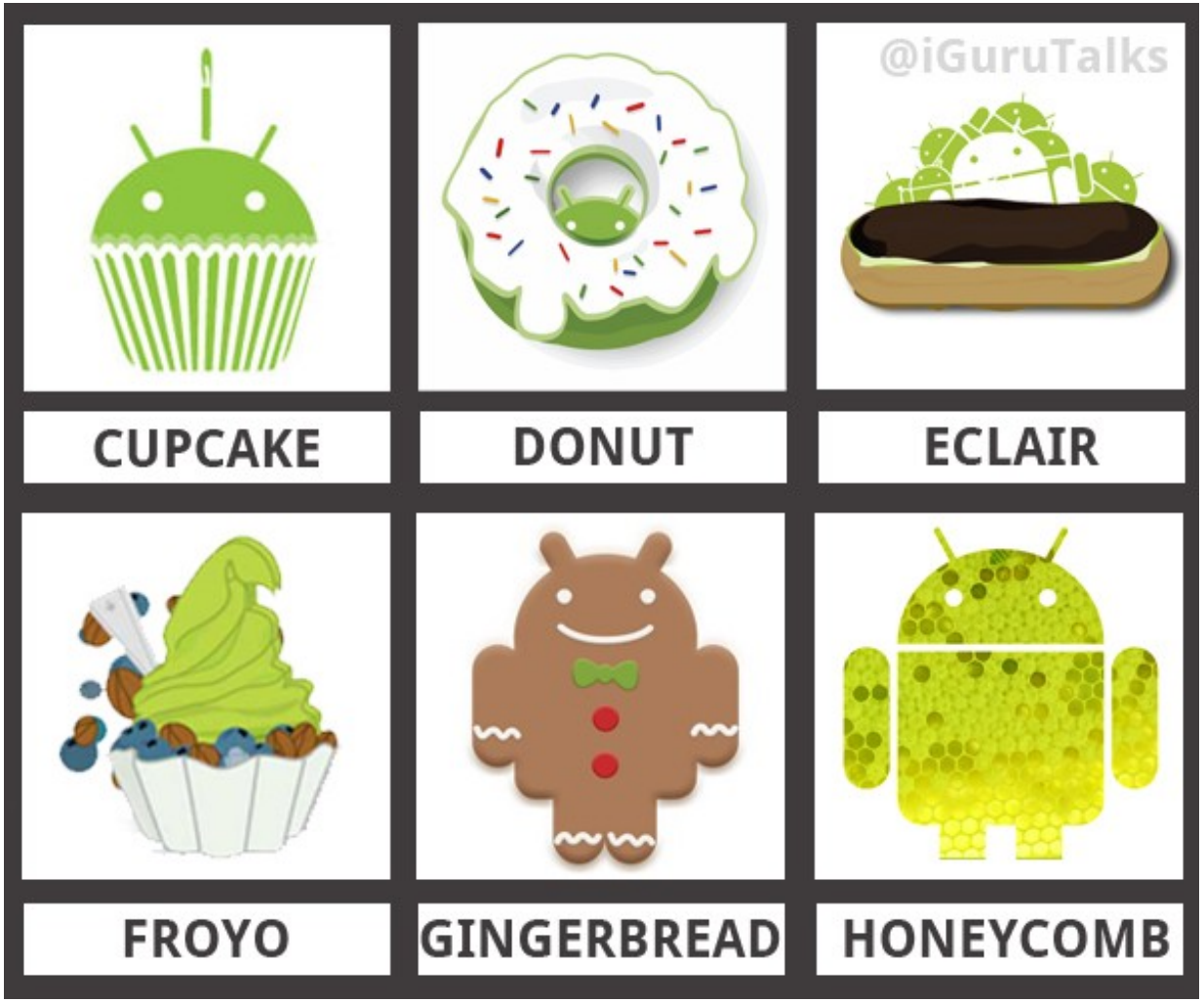


Interesting applications

- **CyanogenMod**
  - Aftermarket firmware for a number of cell phones
  - It offers features not found in the official Android based firmwares of vendors of these cell phones
  - Git project available online

Android naming

Android naming



Source: http://igurutalks.com/internet/historyofandroid/

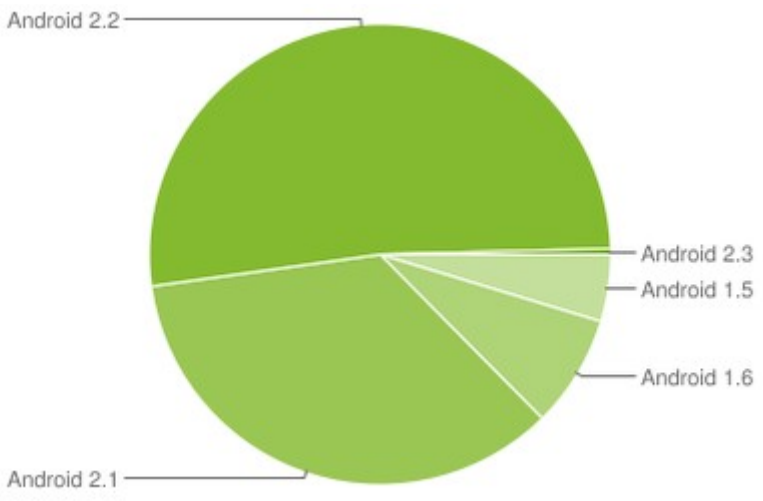
Android History

- Android Inc. founded in 2003 (software for mobile phones)
- Android Inc. acquired by Google in 2005
  - Open source mobile platform built on top of the Linux Kernel

- **Android 1.0**
  - Released in September 2008
  - The first mobile to run Android 1.0 was HTC Dream (or) HTC G1.This mobile was officially marketed by Google.
- **Android 1.1**
  - Released in February 2009
  - Android Dev Phones (ADP)

Android History

- **Android** 1.5 (Cupcake)
  - Released in April 2009
- **Android** 1.6 (Donut)
  - Released in September 2009
- **Android** 2.0, 2.1 (Eclair)
  - 2.0 released in October 2009
  - 2.1 released in January 2010
- **Android** 2.2 (Froyo)
- **Android** 2.3 (Gingerbread)
- **Android** 3.0 (Honeycomb)



Usage share as of March 2011 (source: Wikipedia)

Android Eclair (2.0 / 2.1)

- Android 2.0
  - October 26, 2009
  - API level 5
- Android 2.1
  - January 12, 2010
  - API level 7 (this is correct)
- Revamped the user interface
- Introduced HTML5 and Exchange ActiveSync 2.5 support

Android Froyo – FROzen YOgurt (2.2)

- Android 2.2
  - May 2010
  - Updated to 2.2.2 July 2010
  - API level 8

- Big performance increase due to JAVA Just In Time Compiling.

  Over 4x application performance increase
- Chrome V8 JavaScript
- Wi-Fi hotspot tethering
- Adobe Flash support

Android Gingerbread (2.3)

- Android 2.3
  - December 2010
  - API level 9

- Multiple camera
- Support for larger screens
- Improved the soft keyboard
- Copy/paste features
- Support for Near Field Communication
- USB Accessory (2.3.4)

Android Honeycomb (3.x)

- Android 3.0 and 3.1
  - API levels 11 and 12
- Tablet-oriented
- Larger screen
- Many new interface features
- USB Host (3.1)

Android is Open Source

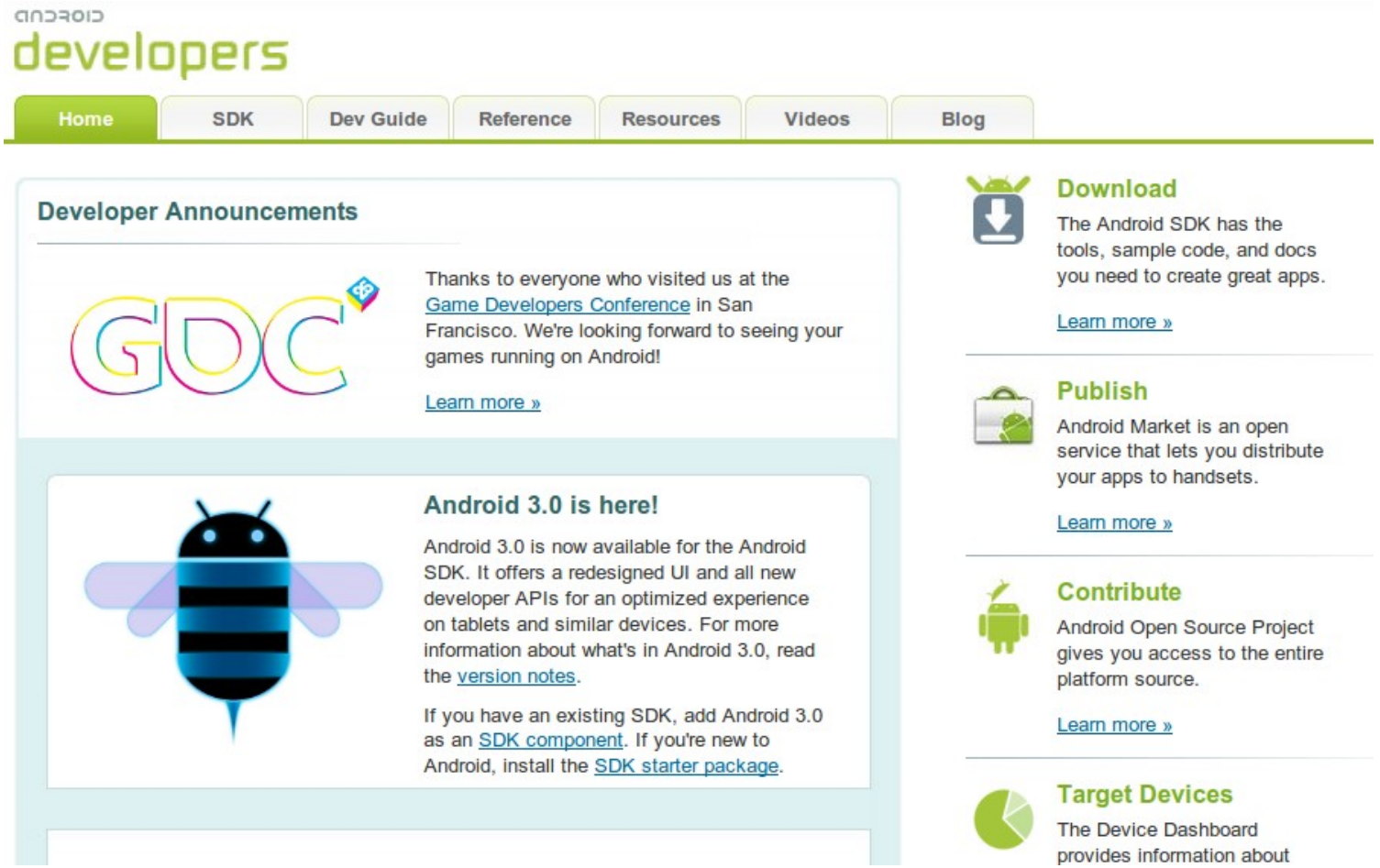
- The Android OS is opens source, available for anyone to download
  - Most of the Android code is released under the Apache License
- Android runs on top of Linux, which of-course is also Open Source
- This makes Android based phones and devices completely open, and EASY to modify/Hack

www.android.com

- Central entry point

developers.android.com

- For application developers (SDK)

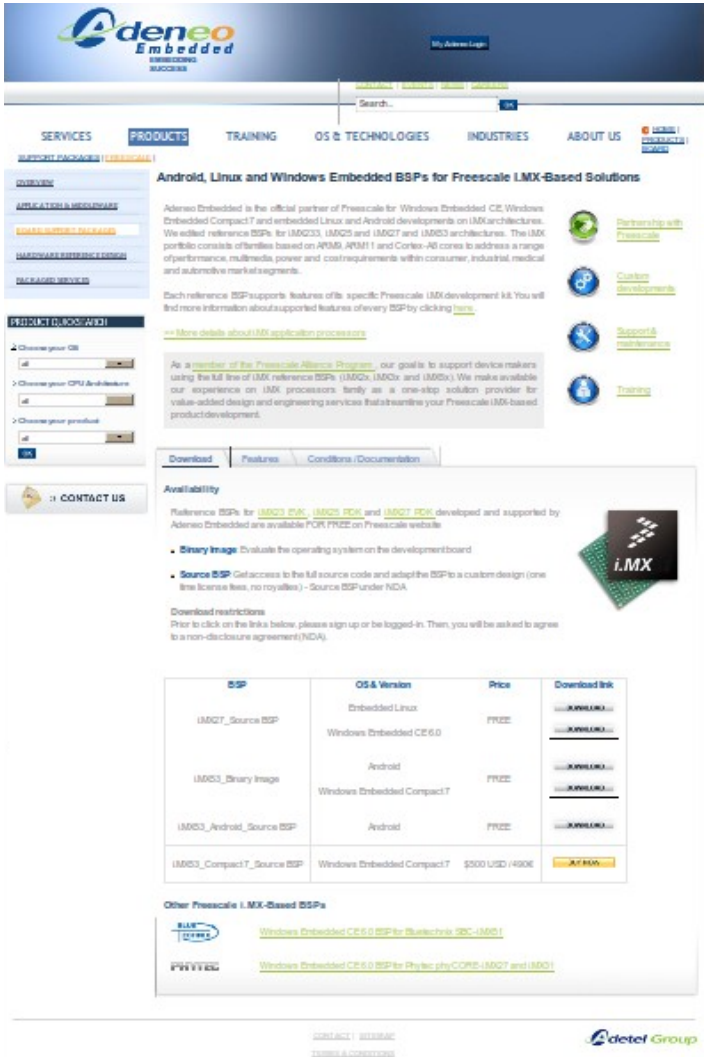


source.android.com

- For platform developers (BSP)

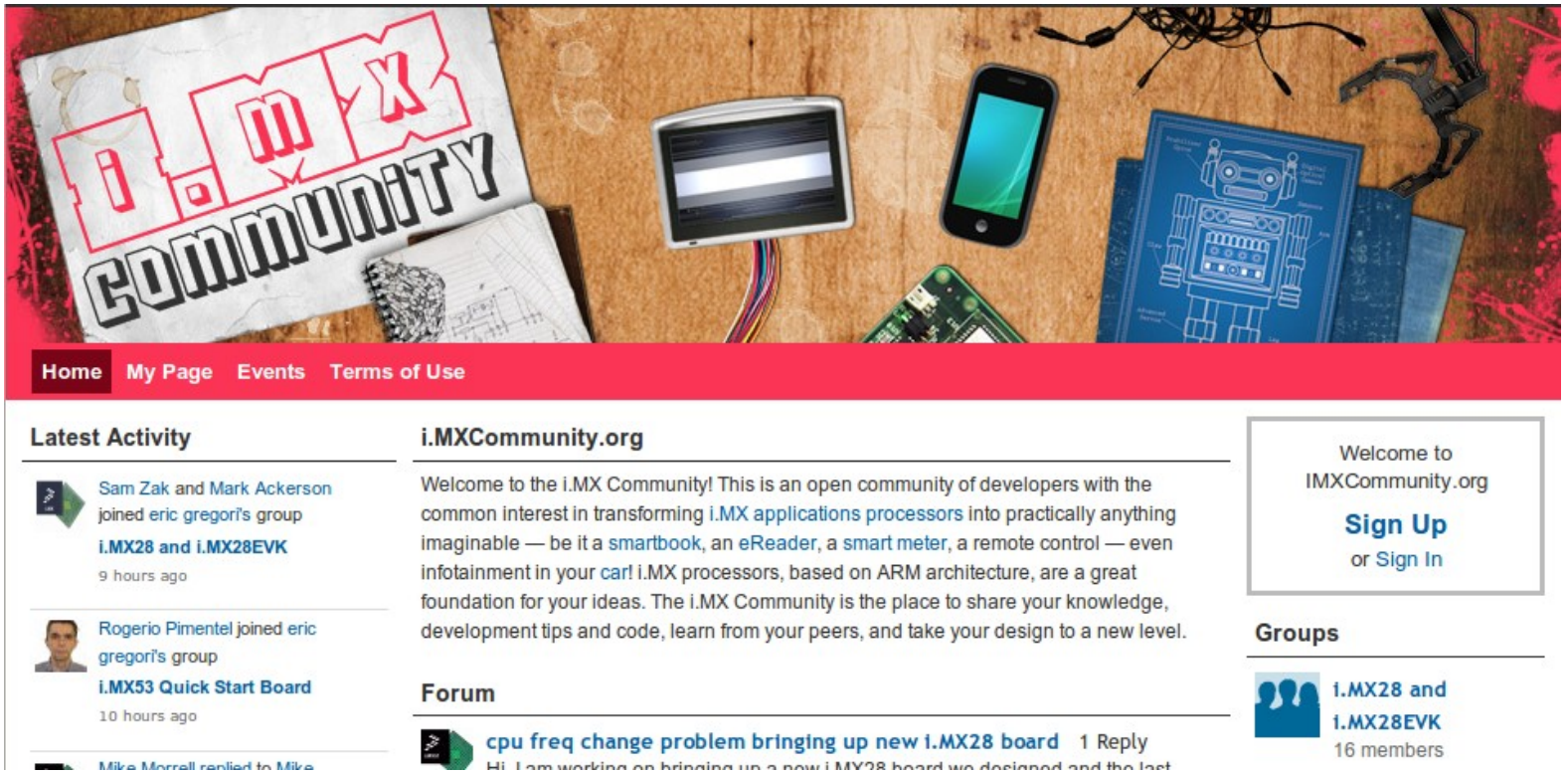
Getting the Android sources for the i.MX53 QSB

- Get the sources and documentation from Adeneo's website
- The Android sources for the other i.MX products can be downloaded on Freescale's website
- You need the sources if you want to make changes to the system (typically not if you are writing applications)



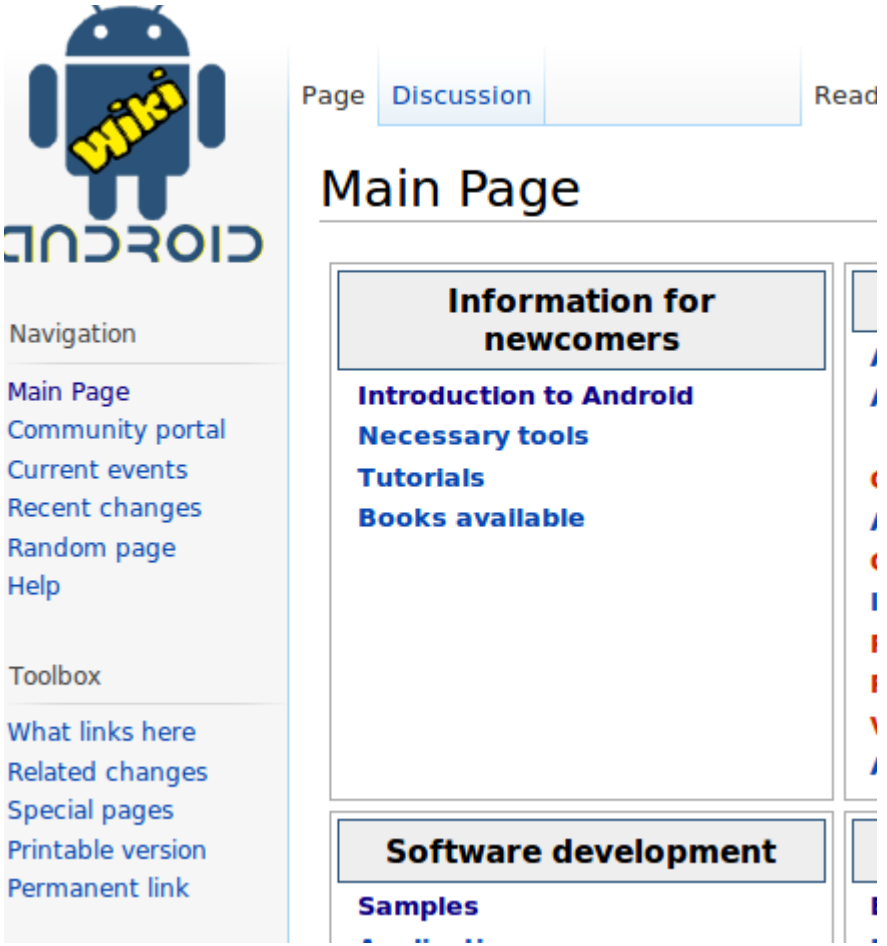
imxcommunity.com

- Open community of developers
- Get help
- Get access to online trainings and events

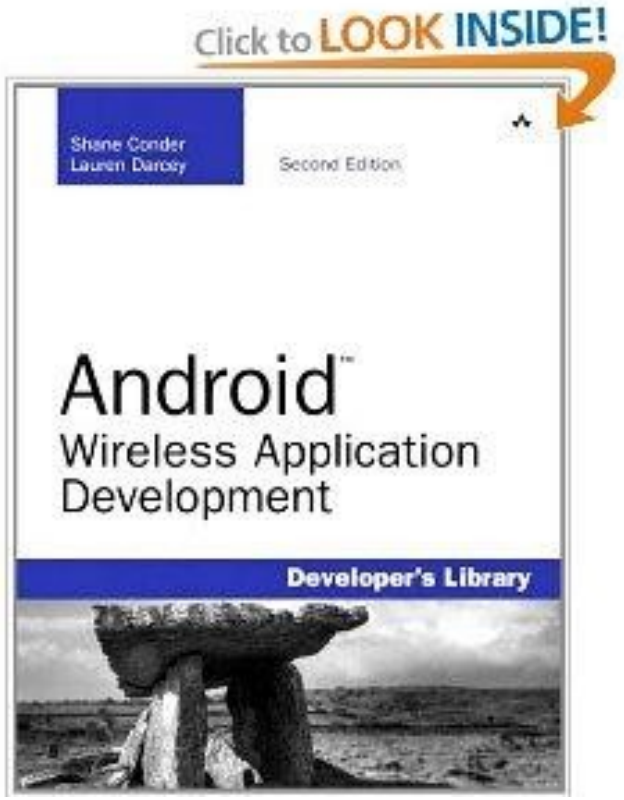


Other websites

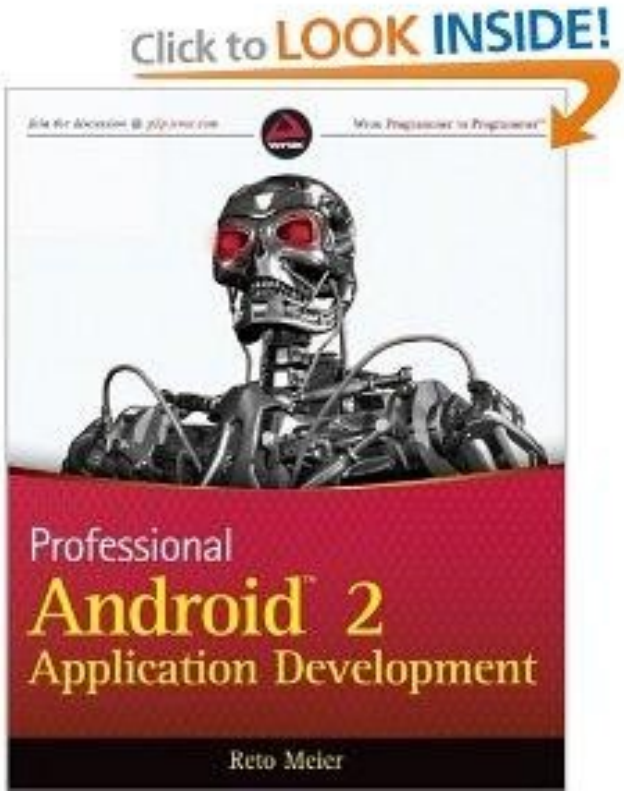
- http://en.androidwiki.com
- … many other websites (use Google!)



Books



ISBN-10: 0321743016



ISBN-10: 0470565527

Lab: Using Android on the i.MX53 Quick Start Board

freescale™ semiconductor

Adeneo Embedded

LAB: Flashing the Android images

- Reminder: these labs will be run from the VM

- Insert the SD Card into your PC's card reader
- Open a terminal and issue the following command:

  > *dmesg*

  - This will display the kernel messages of your host and tell you what entry in */dev* corresponds to your SD Card reader, e.g.

```
[15762.076079] tifm_core: MMC/SD card detected in socket 0:1
[15762.305755] mmc1: new SDHC card at address e624
[15762.305984] mmcblk0: mmc1:e624 SU04G 3.69 GiB
[15762.306198]  mmcblk0: p1
```

  => The device is */dev/mmcblk0* in this case

  - Alternatively, you can use

    > *cat /proc/partitions*

    Before and after you insert the SD Card. This way, you can see what has changed.

- **CAUTION: make sure you are using the right device node! Any mistake could cause you to erase your hard drive! Ask your instructor if you have any doubt.**

LAB: Flashing the Android images

- Flashing scripts have been prepared for the lab environment. They will prepare the whole Android SD Card for you

- Use the following commands:

  > *cd ~/training_mx53/android/*

  > *./flash_android.sh [**NAME OF YOUR SD CARD DEVICE (e.g. /dev/mmcblk0)**]*

  - You will be prompted for the superuser password ("*freescale*") and the SD Card will be prepared (takes a few minutes).

- Have a look at the script if you are curious!

LAB: Running Android on the device

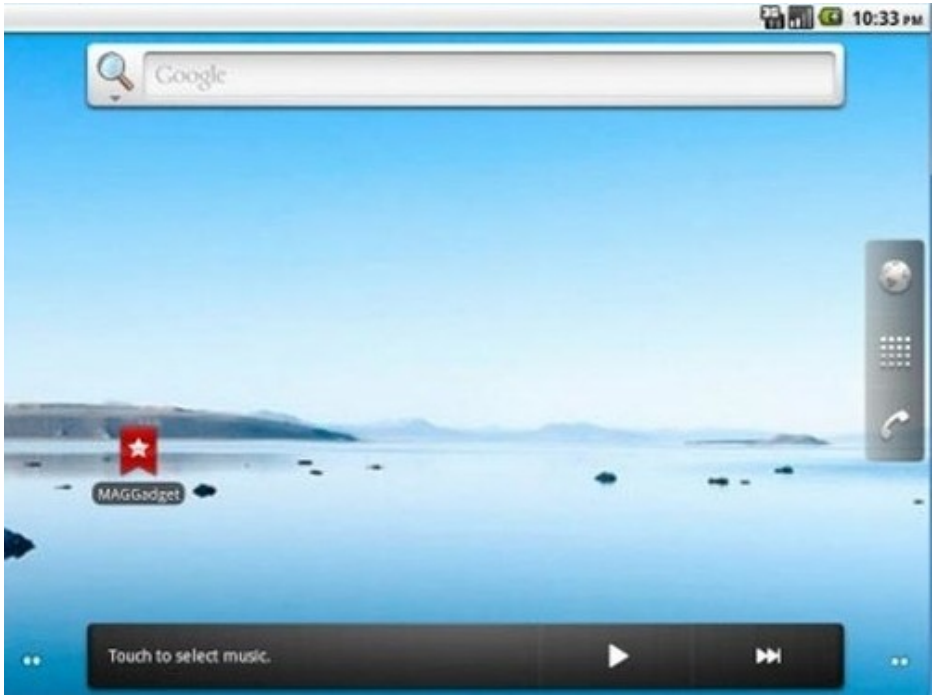
- You can remove the SD Card once the script is done flashing it
- Insert the SD Card in the micro SD slot (the smallest one)
- Turn on your board (do not forget to use the power button – you should see the LEDs turn blue and green)

- You should successively see:
  - The Linux penguin logo
  - The calibration screen (only done once, so make sure you are doing it right – one light touch per cross is enough)
  - The Android logo
  - A prompt saying “Complete action using” => select the value at the top

Note: If the screen remains black after a few minutes, just restart the board

LAB: Running Android on the device

- Click on the launcher (2nd button in the icon bar on the right of the screen)
- Choose *Settings* | *Display*
  - Set the brightness to the maximum
  - Disable *Auto-rotate screen*
  - Set the *Screen timeout* to 30 minutes
- Go back to the Home screen using the two user buttons on the board (between the Ethernet and SD Card connectors)
- Play around!



Android architecture

freescale™ semiconductor

Adeneo Embedded

Android key features

- **Connectivity**
  - Supports connectivity technologies including GSM/EDGE, CDMA, EV-DO, UMTS, Bluetooth, and Wi-Fi
- **Web browser**
  - Web browser available in Android is based on the open-source WebKit application framework
- **Media**
  - Supports the following audio/video/still media formats: H.263, H.264 (in 3GP or MP4 container), MPEG-4 SP, AMR, AMR-WB (in 3GP container), AAC, HE-AAC (in MP4 or 3GP container), MP3, MIDI, OGG Vorbis, WAV, JPEG, PNG, GIF, BMP
- **Hardware and graphics**
  - Can use video/still cameras, touchscreens, GPS, accelerometers, magnetometers, accelerated 2D bit blits (with hardware orientation, scaling, pixel format conversion) and accelerated 3D graphics

Android key features

- **Multi-touch**
  - Has native support for multi-touch which is available in newer handsets
- **Android Market place**
  - Catalog of applications that can be downloaded and installed to target hardware over-the-air, without the use of a PC
- **Dev environment**
  - Includes a device emulator, tools for debugging, memory and performance profiling, a plugin for the Eclipse IDE

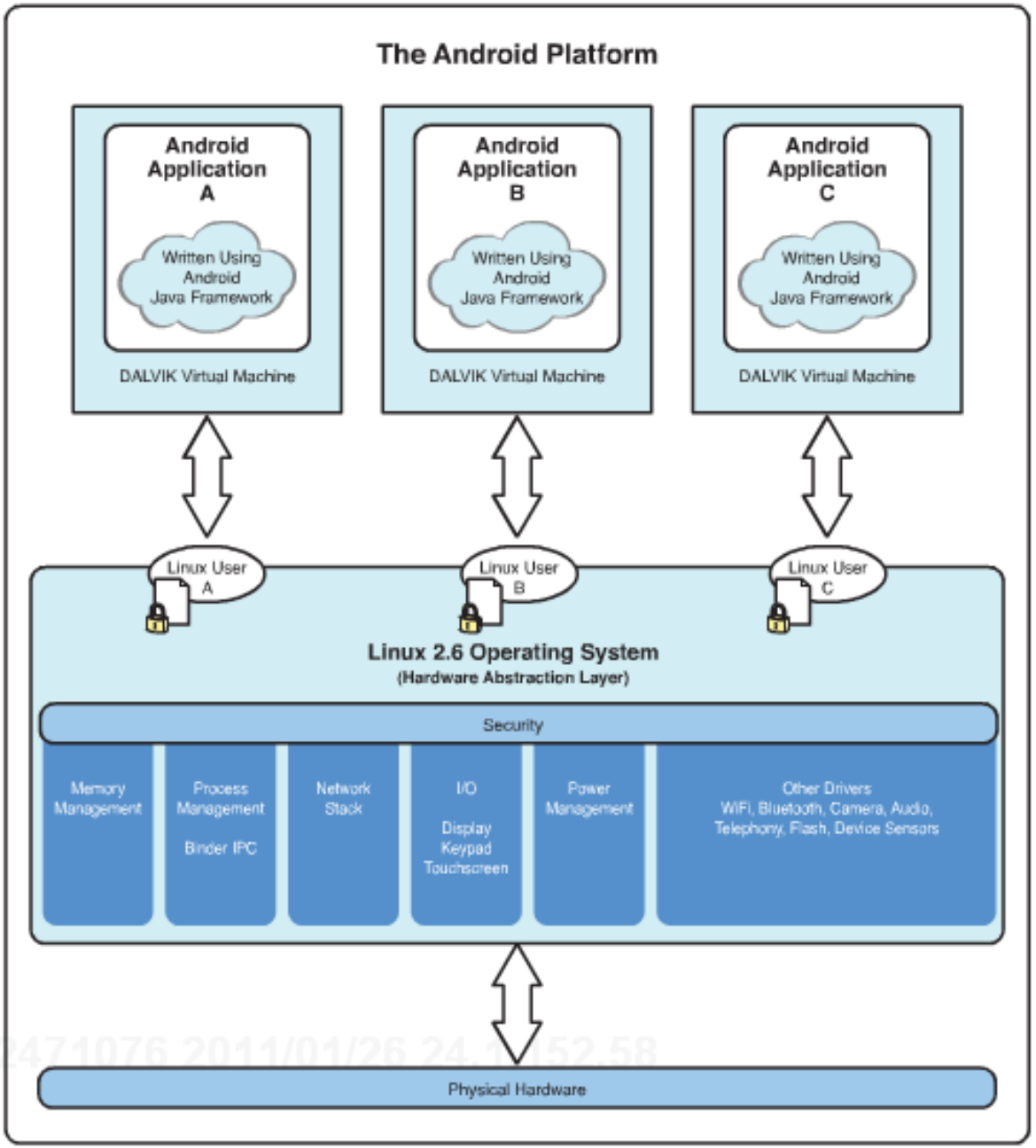
Android platform details

- Android uses Linux for its device drivers, memory management, process management, and networking

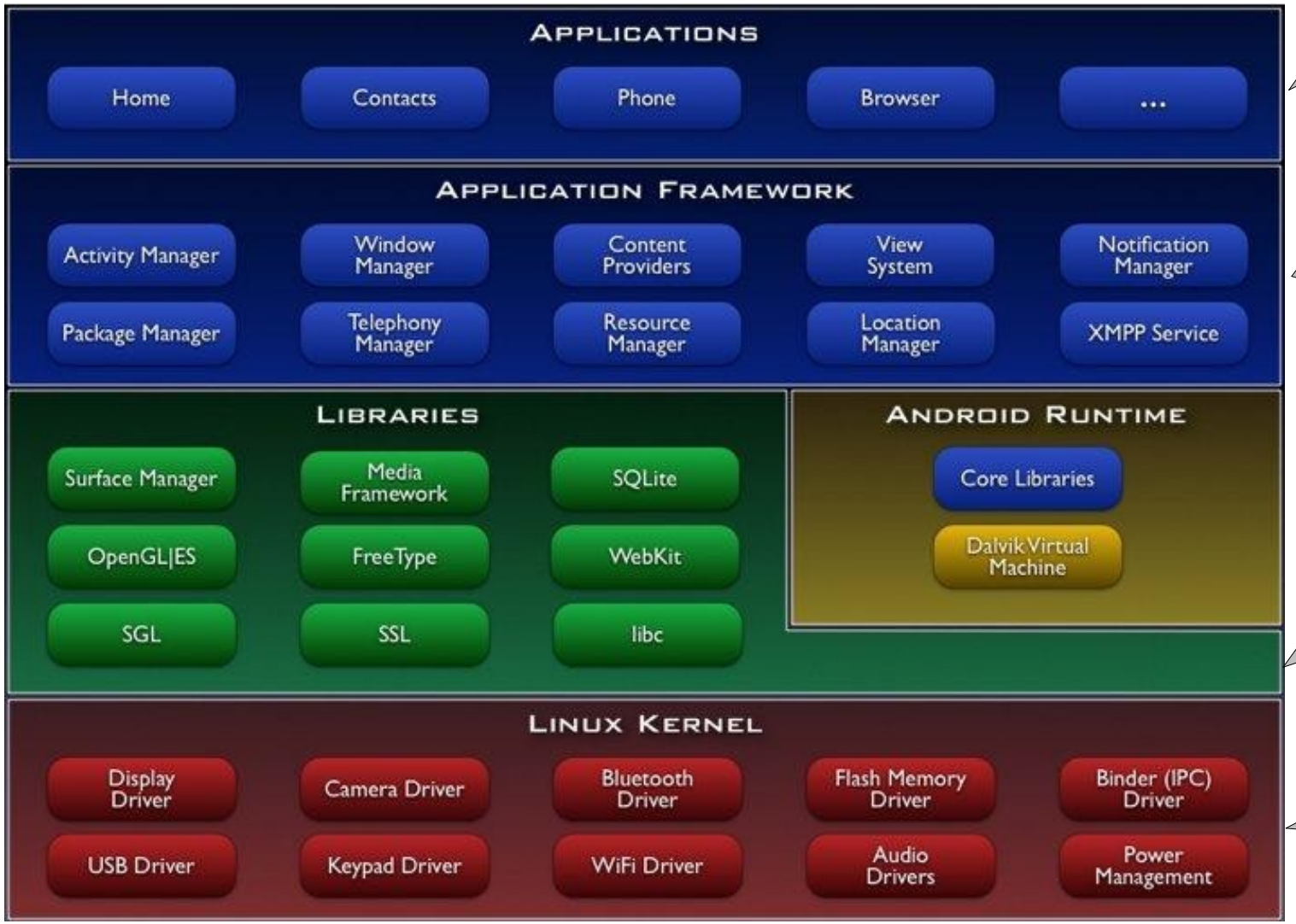
- The next level up contains the Android native libraries
  - Written in C/C++ internally
  - Called through Java interfaces (in most cases).
  - In this layer you can find the Surface Manager, 2D and 3D graphics, Media codecs, the SQL database (SQLite), and a native web browser engine (WebKit)

- Dalvik Virtual Machine
  - Runs Java programs

Android platform details



Android software stack



Apps (Java) – Everyone can create his/her own application based on Android API

Android "Program" API

Middleware (Java) – App framework including window/focus management, inter-app communication, event notification, etc.

Middleware (C/C++) – system libraries for media, graphic, database, font, web engine, etc.

Android "Porting" IF

2.6 based Linux kernel with Android patch

From “Android Anatomy and Physiology”

Linux Kernel

- Android is built on the Linux kernel, but Android is not Linux
- No native windowing system
- No glibc support
- Does not include the full set of standard Linux utilities

From “Android Anatomy and Physiology”

Why Linux Kernel?

- Great memory and process management
- Permissions-based security model
- Proven driver model
- Support for shared libraries
- It’s already open source!

From “Android Anatomy and Physiology”

Kernel Enhancements

- Alarm
- Ashmem
- Binder
- Power Management
- Low Memory Killer
- Kernel Debugger
- Logger

From “Android Anatomy and Physiology”

Hardware Abstraction Libraries

- User space C/C++ library layer
- Defines the interface that Android requires hardware “drivers” to implement
- Separates the Android platform logic from the hardware interface

From “Android Anatomy and Physiology”

Bionic libc

- BSD License
- Small size and fast code paths
- Very fast and small custom pthread implementation

From “Android Anatomy and Physiology”

Bionic libc

- Doesn't support certain POSIX features
- Not compatible with Gnu Libc (glibc)
- All native code must be compiled against bionic

From “Android Anatomy and Physiology”

WebKit

- Based on open source WebKit browser: http://webkit.org
- Renders pages in full (desktop) view
- Full CSS, Javascript, DOM, AJAX support
- Support for single-column and adaptive view rendering

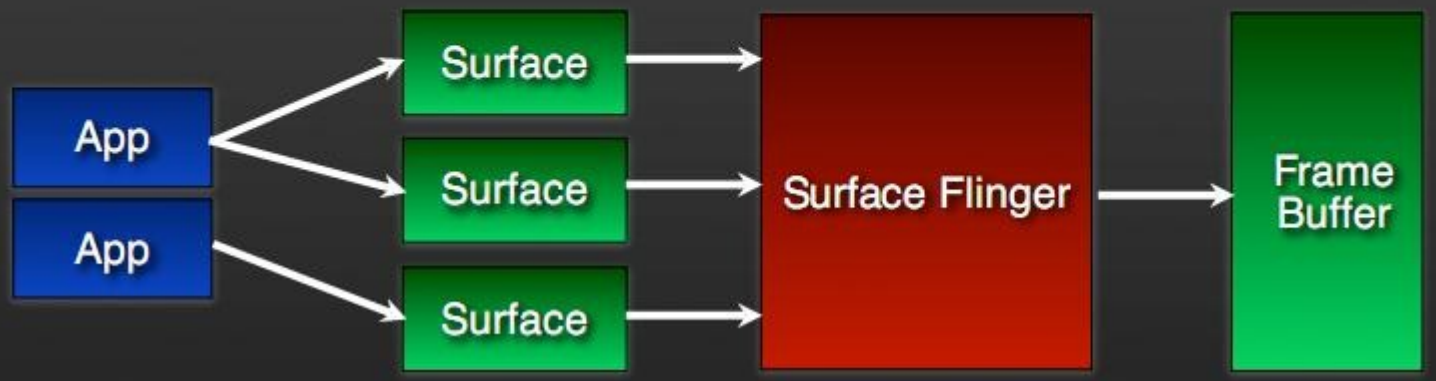
From “Android Anatomy and Physiology”

Media Framework

- Based on PacketVideo OpenCORE platform
- Supports standard video, audio, still-frame formats
- Support for hardware / software codec plug-ins

From “Android Anatomy and Physiology”

Surface Flinger



- Provides system-wide surface “composer”, handling all surface rendering to frame buffer device
- Can combine 2D and 3D surfaces and surfaces from multiple applications

From “Android Anatomy and Physiology”

From “Android Anatomy and Physiology”

Dalvik Virtual Machine

- Designed for embedded environment
  - Supports multiple virtual machine processes per device
  - Highly CPU-optimized bytecode interpreter
  - Uses runtime memory very efficiently

From “Android Anatomy and Physiology”

Dalvik Virtual Machine

- Android’s custom clean-room implementation virtual machine
  - Provides application portability and runtime consistency
  - Runs optimized file format (.dex) and Dalvik bytecode
  - Java .class / .jar files converted to .dex at build time

From “Android Anatomy and Physiology”

Core Libraries

- Core APIs for Java language provide a powerful, yet simple and familiar development platform
  - Data structures
  - Utilities
  - File access
  - Network Access
  - Graphics
  - ...

From “Android Anatomy and Physiology”

Core Platform Services

- Services that are essential to the Android platform
- Behind the scenes - applications typically don’t access them directly

From “Android Anatomy and Physiology”

Core Platform Services

- Activity Manager
- Package Manager
- Window Manager
- Resource Manager
- Content Providers
- View System

From “Android Anatomy and Physiology”

Hardware Services

- Provide access to lower-level hardware APIs
- Typically accessed through local *Manager* object

```
LocationManager lm = (LocationManager)
   Context.getSystemService(Context.LOCATION_SERVICE);
```

From “Android Anatomy and Physiology”

Hardware Services

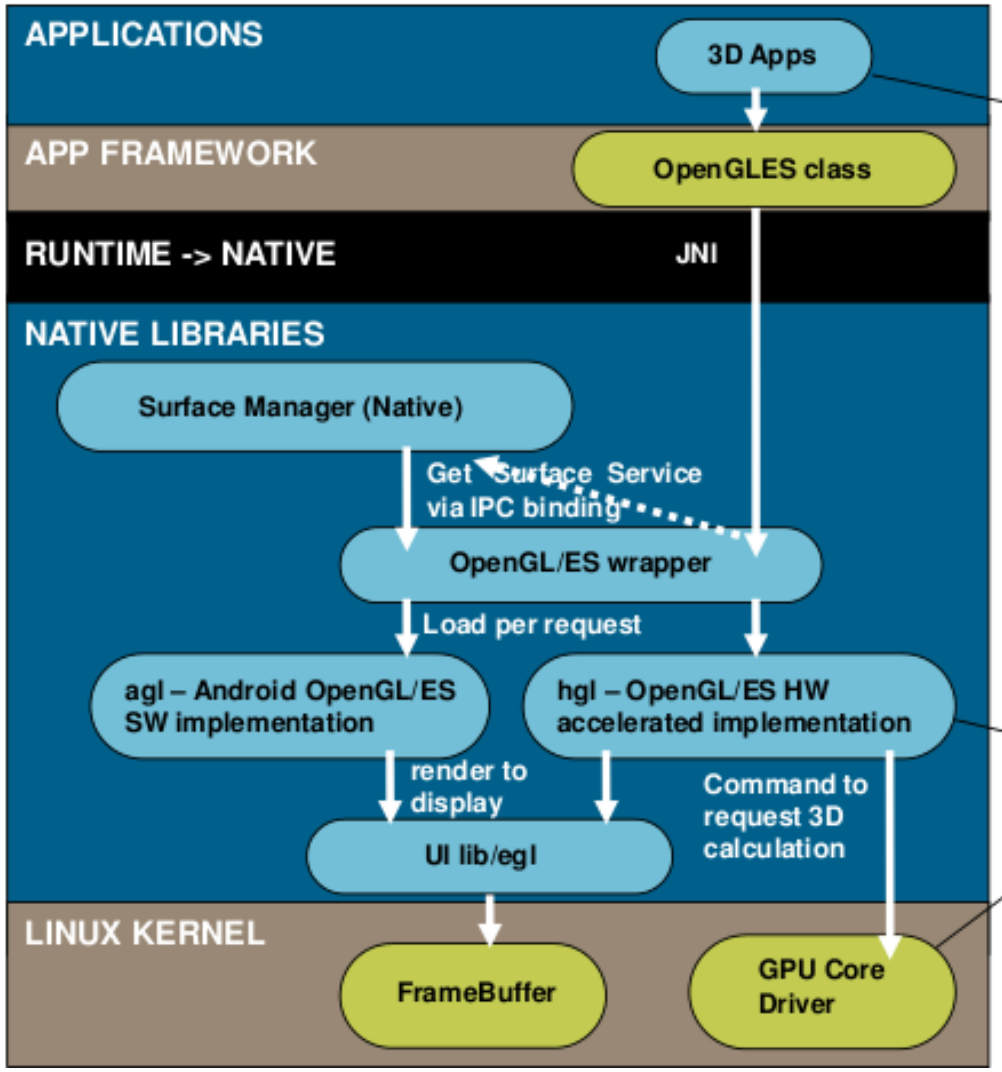
- Telephony Service
- Location Service
- Bluetooth Service
- WiFi Service
- USB Service
- Sensor Service

Graphics acceleration on Android

freescale™ semiconductor

Adeneo Embedded

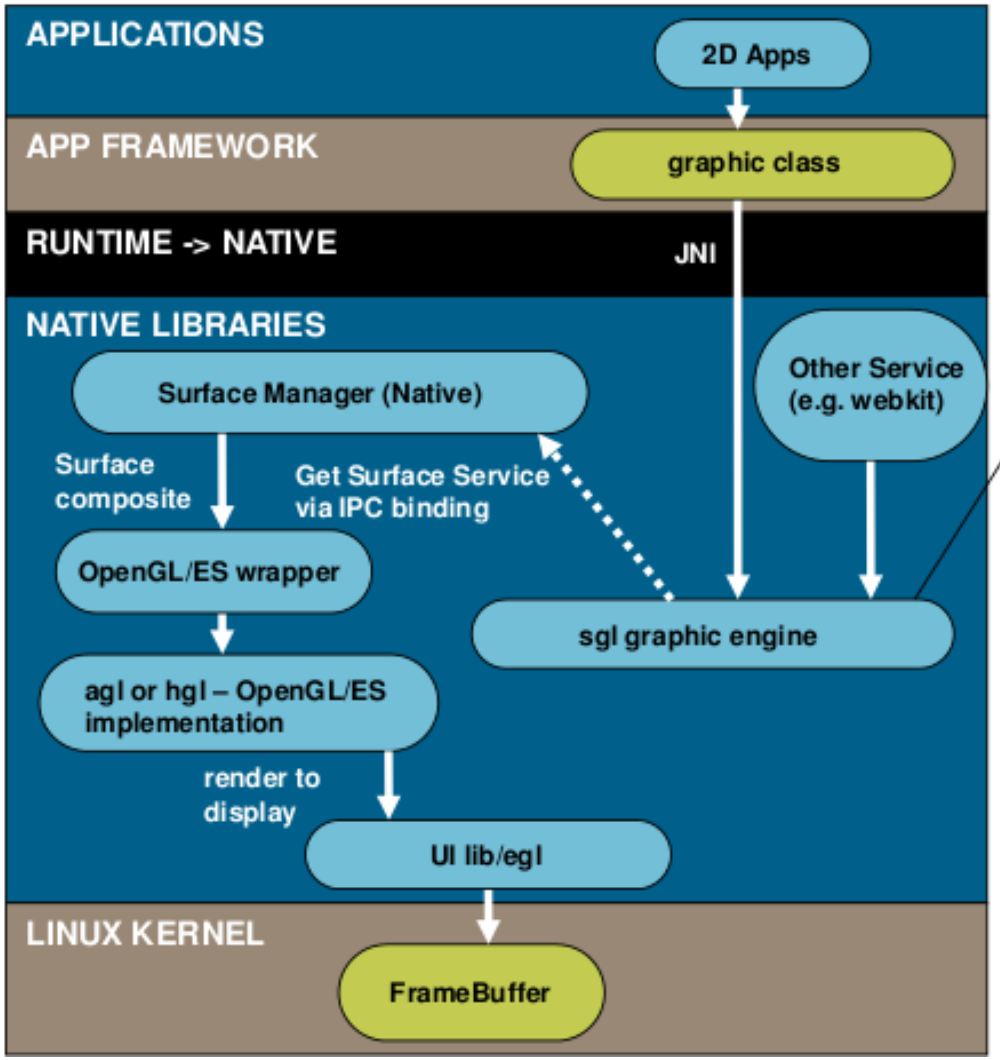
Multimedia – 3D graphics



Verify OpenGL/ES based 3D operation is available in APP level as expected (i.e. either SW or HW OpenGL/ES is properly integrated into Android).

Create a "hgl" and integrate in GPU accelerated OpenGL/ES implementation.

Multimedia – 2D graphics



Use SGL.

Graphics

- More information about accelerated graphics in the GPU SDK can be found in the downloads section for the i.MX5 family (remember the Ubuntu demo).

Writing applications for Android

freescale™ semiconductor

Adeneo Embedded

Android development

- Android development is done in Java (using the SDK), C, and C++ (using the NDK)
- The tools are provided for Windows, Linux, and Mac OSX hosts
- The tools include:
  - The Eclipse IDE (not mandatory for Android development)
  - An emulator to test your code on your computer
  - Complete debugger, to test your code directly on your Android device
- The tools are free and well supported

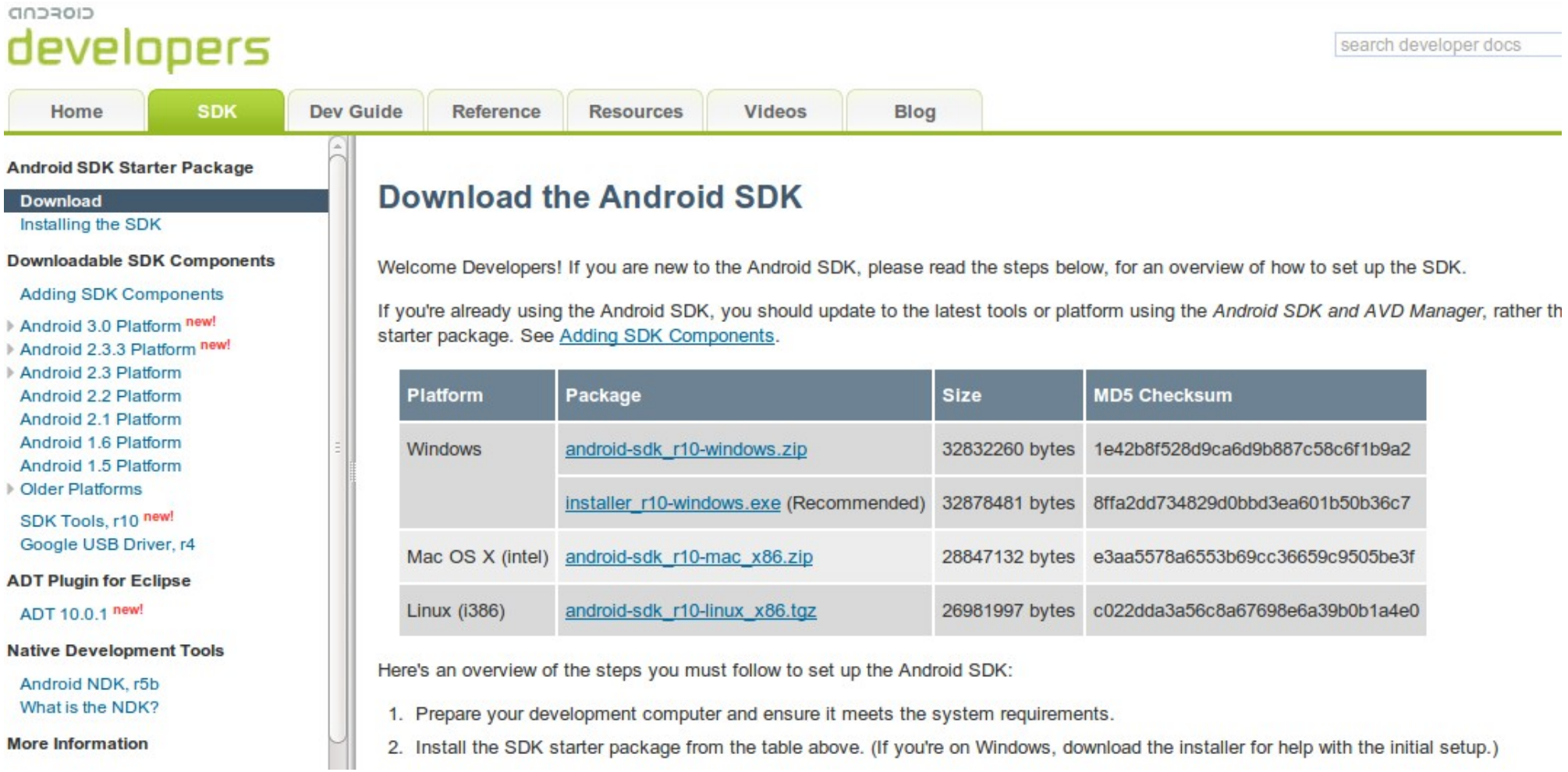
Android SDK and tools

- The Android SDK contains:
  - Class Library
  - Developer Tools
    - *dx* – Dalvik Cross-Assembler
    - *aapt* – Android Asset Packaging Tool
    - *adb* – Android Debug Bridge
    - *ddms* – Dalvik Debug Monitor Service
  - Emulator (AVD) and System Images
  - Documentation and Sample Code

- JAVA JDK + Eclipse IDE + ADT (Android Development Tools)
  - Reduces Development and Testing Time
  - Makes User Interface-Creation easier
  - Makes Application Description Easier

Downloading the Android SDK

- http://developer.android.com/sdk



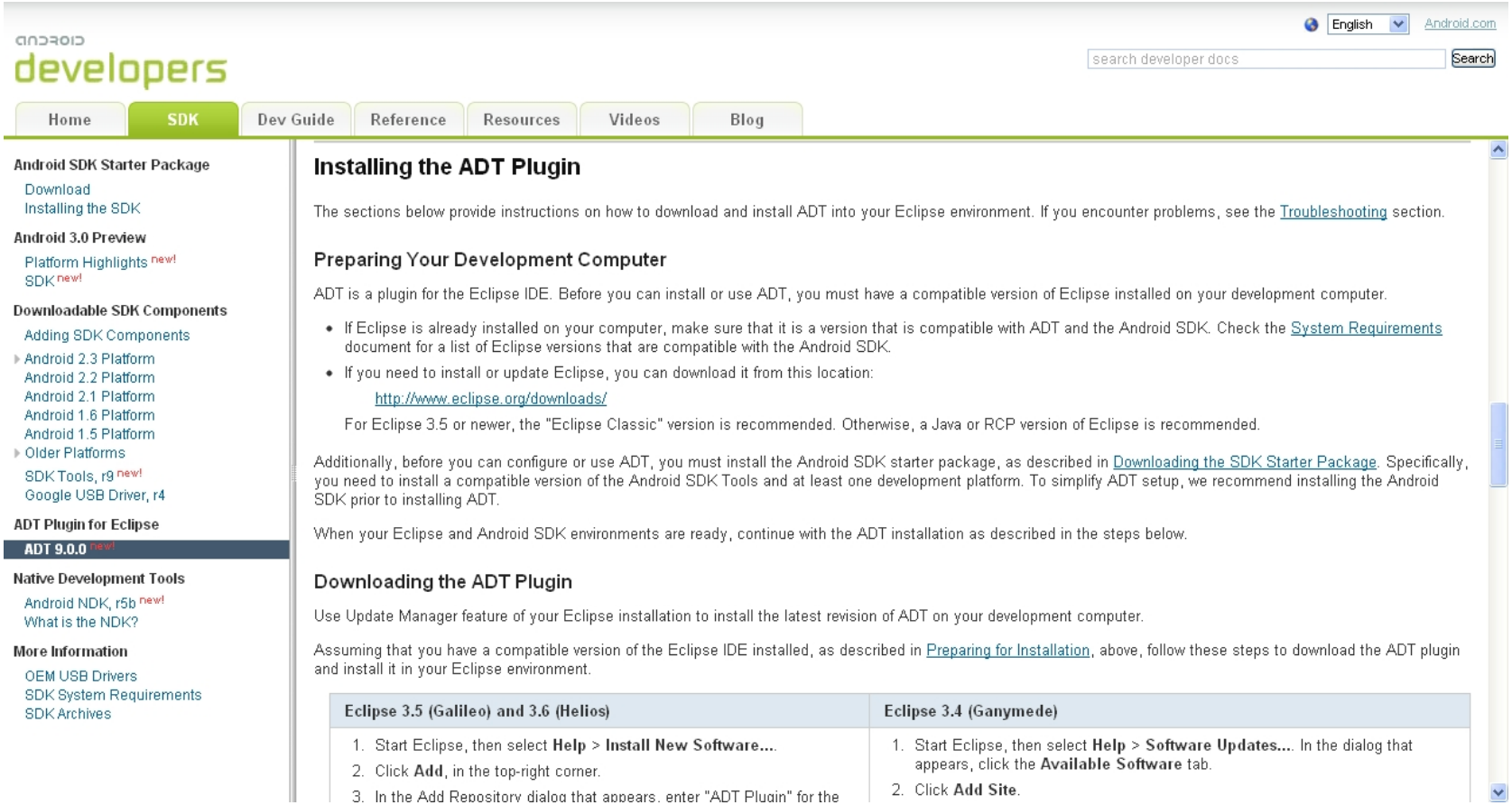
ADT Eclipse plugin

- Android Development Tools (ADT) is a plugin for the Eclipse IDE that is designed to give you a powerful, integrated environment in which to build Android applications.
- ADT extends the capabilities of Eclipse:
  - Set up new Android projects
  - Create an application UI
  - Add components based on the Android Framework API
  - Debug your applications using the Android SDK tools
  - Export signed (or unsigned) APKs in order to distribute your application.

- http://developer.android.com/sdk/eclipse-adt.html#installing

Installing the ADT Eclipse plugin

- Eclipse Classic is recommended for Android development
- http://developer.android.com/sdk/eclipse-adt.html#installing



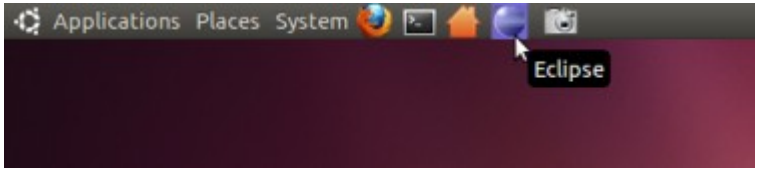
LAB: Building your first Android application

freescale™ semiconductor

Adeneo Embedded

LAB: Hello World

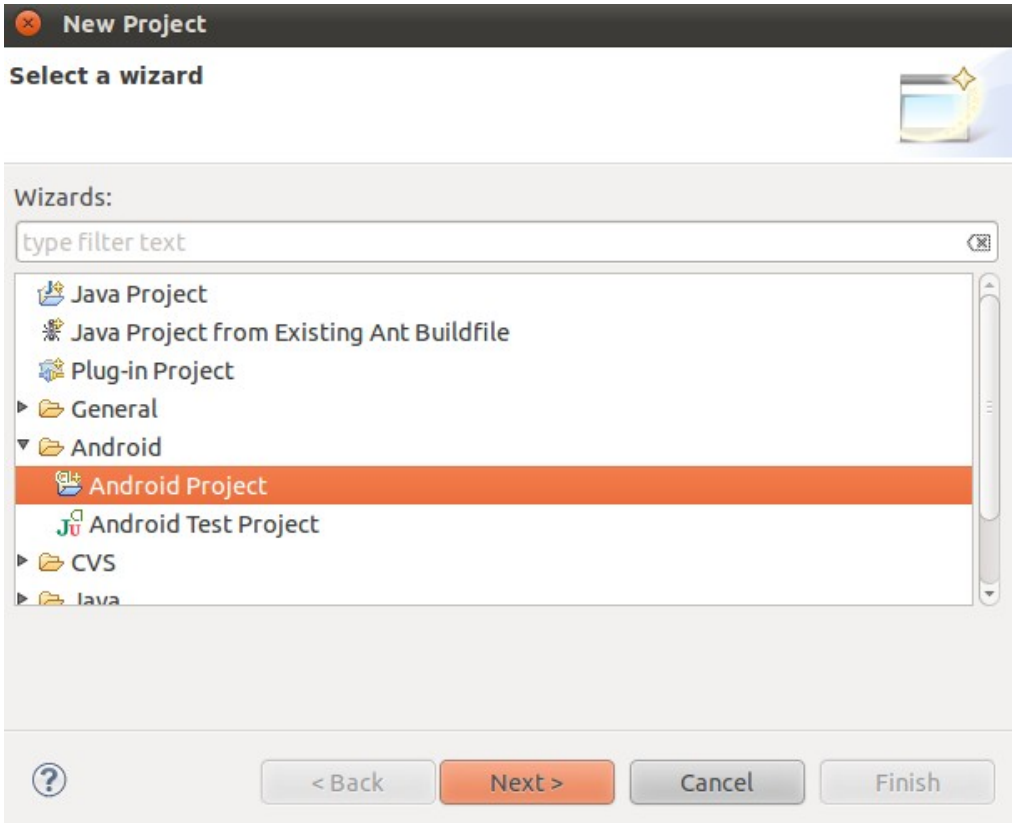
- Start Eclipse Classic (the shortcut can be found in the quicklaunch panel at the top of the screen)



- If prompted for the workspace, go on with the default option

LAB: Hello World

- Create a new Android project:
  - Choose *File* | *New* | *Project...*
  - Choose *Android* → *Android Project*
  - Click *Next*



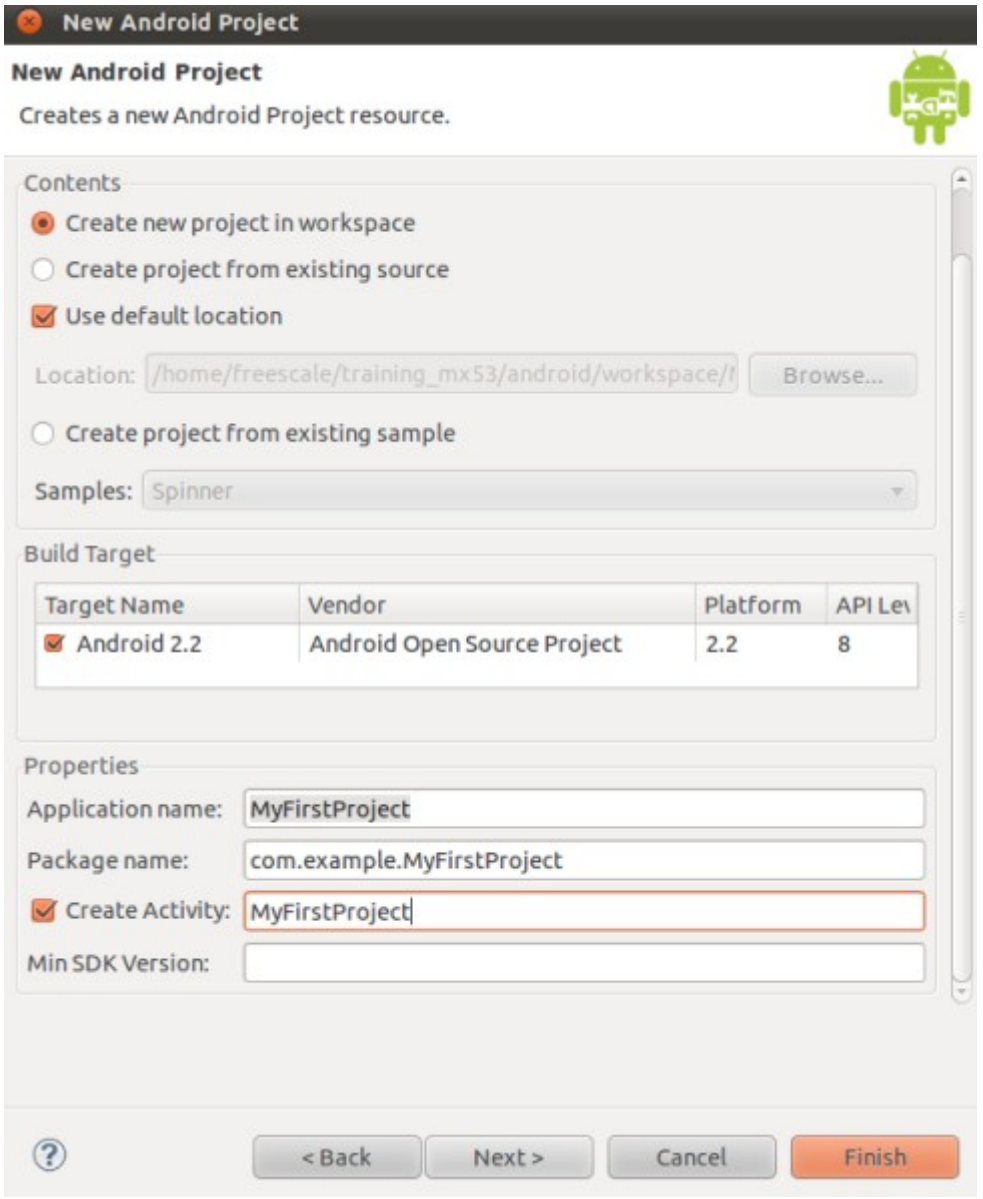
LAB: Hello World

- The New Android Project window will open. Fill in the fields.

- **Project Name:** *MyFirstProject*
- **Contents:**
  - *Create new project in workspace*
  - Check *"Use default location"*
- **Build Target:** *"Android 2.2"* should be checked
- **Application name:** *MyFirstProject*
- **Package name:** *com.example.MyFirstProject*
- **Create Activity:** *MyFirstProject*

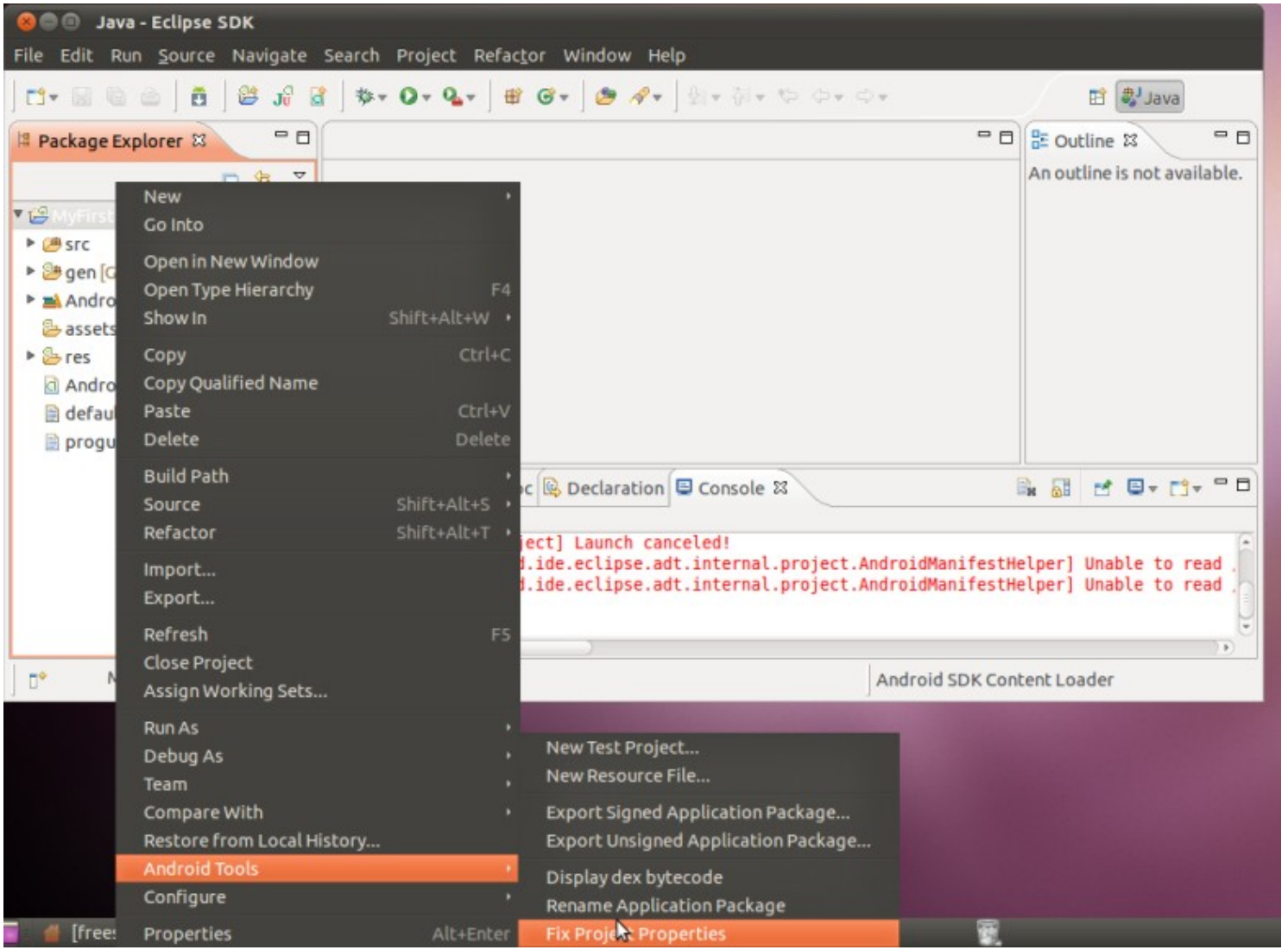
- Click on *Finish*

  The project will be created



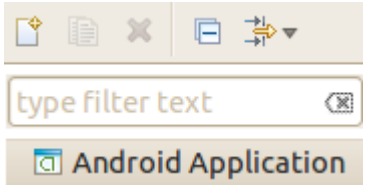
LAB: Hello World

- If you see errors in the console, right-click on the project (in the Package explorer pane) and choose *Android Tools* | *Fix Project properties*

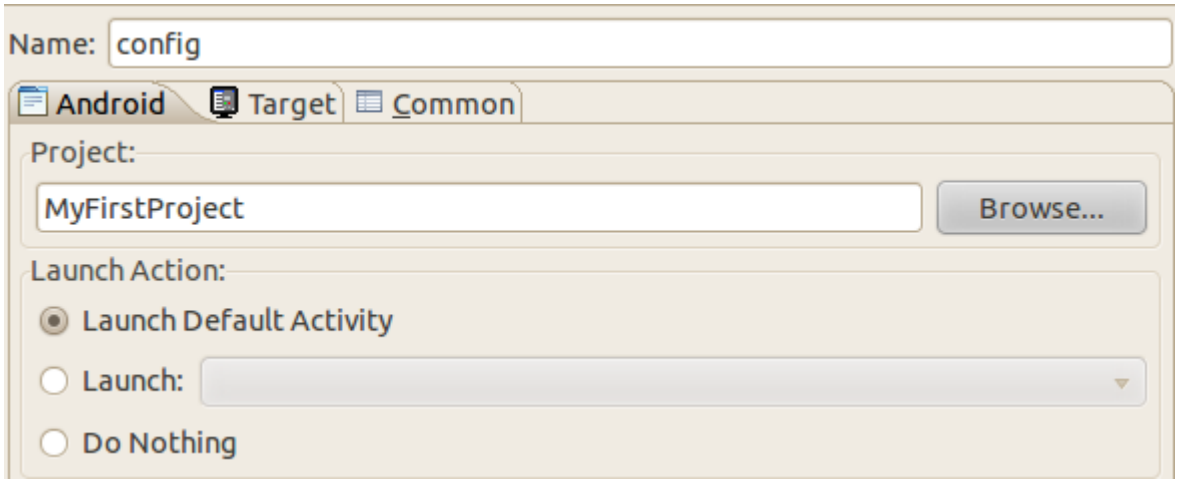


LAB: Hello World

- From the *Run* menu, select *Configurations*
- Select *Android Application* and press the *New* button 



- Choose a name for your configuration (anything works)
- Select your project by clicking *Browse* and choosing *MyFirstProject*



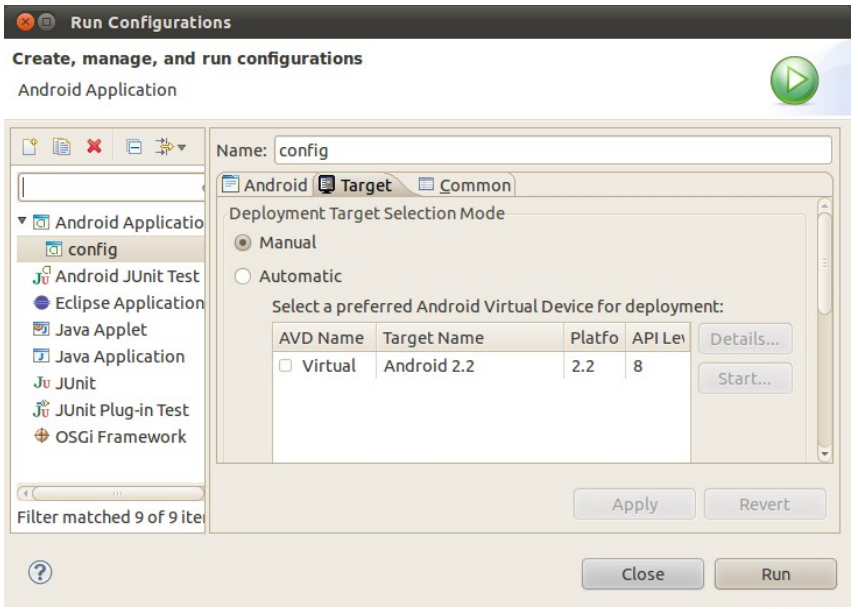
LAB: Hello World

- Switch to the *Target* tab and select *Manual*

  → Now, when you run your program, you will be prompted for which target you want to run it on (i.e. the emulator or a physical device)

  Note: You can also choose *Automatic* and assign the Run configuration for one specific device

- Click on *Run* button (in the bottom right corner)



LAB: Hello World

- If you have not already created a virtual device, you will be asked to create one.

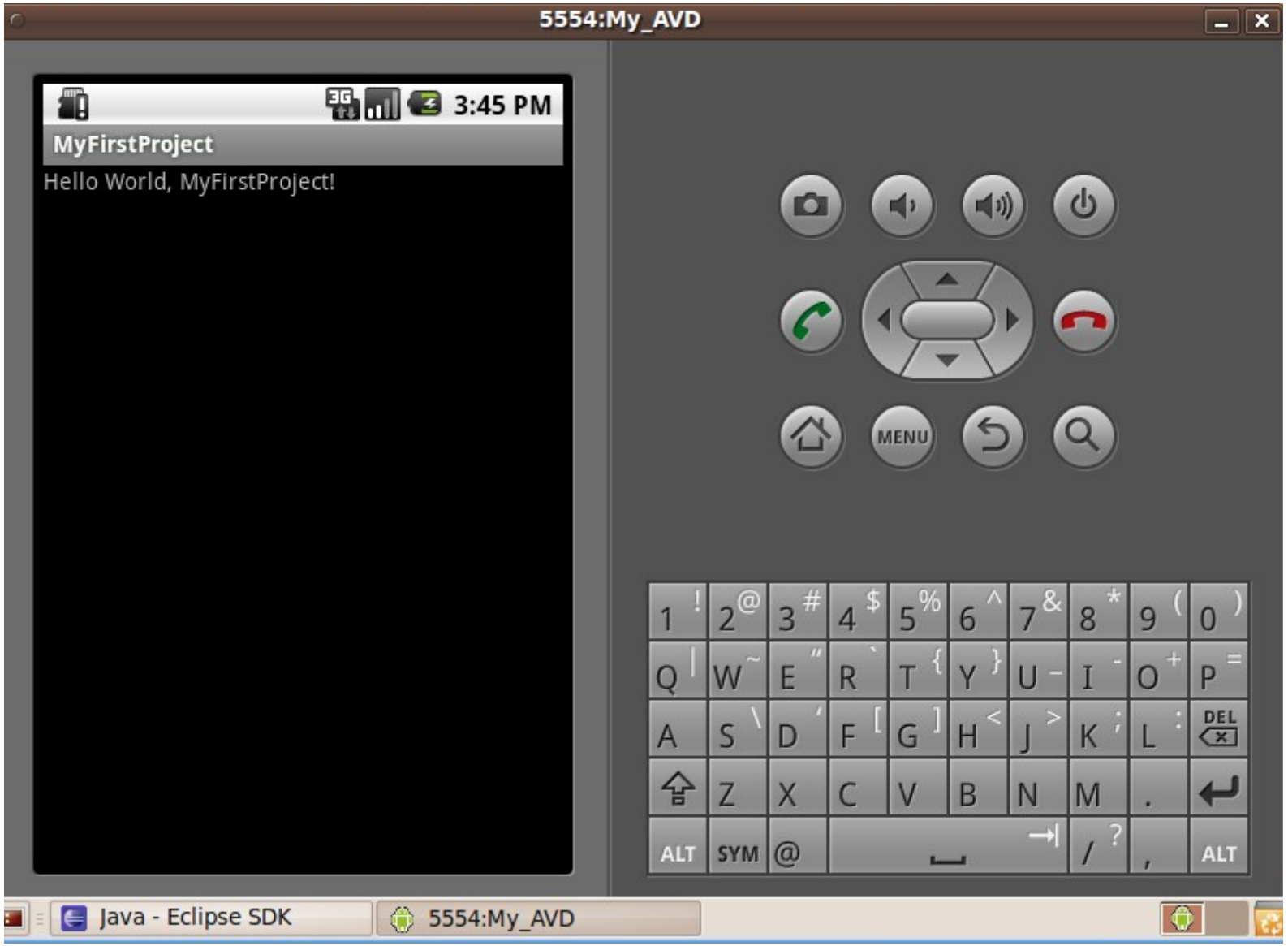
  Answer "*Yes*" in that case.

- In the *Virtual devices* tab, click on *New...*
- Create the device using the following information:
  - **Name**: *mx53_emulator*
  - **Target**: *Android 2.2*
  - **Skin**: *Built-in (WVGA800)*
- If the emulator takes too much space, you can increase the value of the "*Abstracted LCD density*" in the *Hardware* box.
- Click on *Create AVD*

- Click on *Start...*

  The Emulator will start (this takes a few minutes). You can then run your application in the emulator (using the *Run* menu).

LAB: Hello World



Waiting for the emulator to start

- The emulator takes about 5 minutes to start.
- The emulator is painfully slow in a VMWare environment.
- Running directly on Linux, or in Windows, it works fantastically and is a very powerful tool for Android application development.
- To speed things up, do not close it.
- Run the application again if the deploying process seems stuck

Android application files

- The following list describes the structure and files of an Android application. Many of these files can be built for you (or stubbed out) by the android tool shipped in the tools/ menu of the SDK.
- MyApp/
  - AndroidManifest.xml (required)
    - Advertises the screens that this application provides, where they can be launched (from the main program menu or elsewhere), any content providers it implements and what kind of data they handle, where the implementation classes are, and other application-wide information. Syntax details for this file are described in the AndroidManifest.xml File.
- src/
  - /myPackagePath/.../MyClass.java (required) This folder holds all the source code files for your application, inside the appropriate package subfolders.
- res/ (required)
  - This folder holds all the resources for your application. Resources are external data files or description files that are compiled into your code at build time. Files in different folders are compiled differently, so you must put the proper resource into the proper folder. (See Resources for details.)
- anim/
  - animation1.xml(optional) Holds any animation XML description files that the application uses.
- drawable/
  - some_picture.png
  - some_stretchable.9.png
  - some_background.xml

    ... (optional) Zero or more files that will be compiled to android.graphics.drawable resources. Files can be image files (png, gif, or other) or XML files describing other graphics such as bitmaps, stretchable bitmaps, or gradients. Supported bitmap file formats are PNG (preferred), JPG, and GIF (discouraged), as well as the custom 9-patch stretchable bitmap format.

Android application files

- layout/
  - screen_1_layout.xml
  - …

  (optional) Holds all the XML files describing screens or parts of screens. Although you could create a screen in Java, defining them in XML files is typically easier. A layout file is similar in concept to an HTML file that describes the screen layout and components. See User Interface for more information about designing screens, and Available Resource Types for the syntax of these files.
- values/
  - arrays
  - classes.xml
  - colors.xml
  - dimens.xml
  - strings.xml
  - styles.xml
  - values.xml
- (optional) XML files describing additional resources such as strings, colors, and styles. The naming, quantity, and number of these files are not enforced--any XML file is compiled, but these are the standard names given to these files. However, the syntax of these files is prescribed by Android, and described in Resources.
- xml/ (optional) XML files that can be read at run time on the device.
- raw/ (optional) Any files to be copied directly to the device.

Android Manifest

- The information in this file is used by the Android system to:
  - Install and upgrade the application package
  - Display the application details such as the application name, description, and icon to users
  - Specify application system requirements, including which Android SDKs are supported, what hardware configurations are required (for example, d-pad navigation),and which platform features the application relies upon (for example, uses multitouch capabilities).
  - Launch application activities
  - Manage application permissions
  - Configure other advanced application configuration details, including acting as a service, broadcast receiver, or content provider.
  - Enable application settings such as debugging and configuring instrumentation for application testing.

DDMS

- Android DDMS (Dalvik Debug Monitor Server)
- DDMS is represented as an Eclipse Perspective.

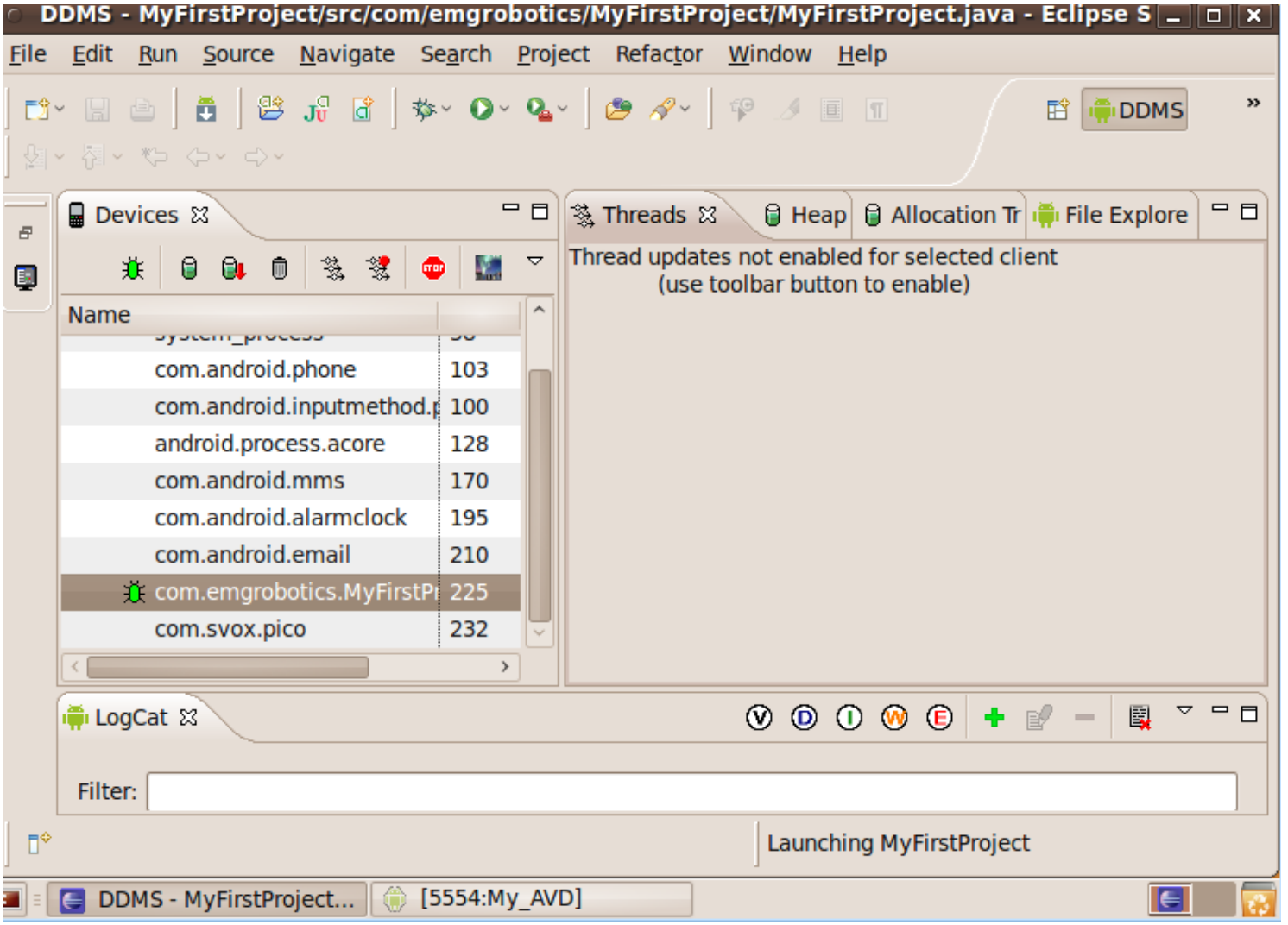
- With Eclipse, you can think of a perspective as a building containing a bunch of windows. In Eclipse, a window is called a view.

  So a perspective contains a bunch of views (windows).

- Perspectives are used to group views based on a common task.

- The DDMS perspective is a debugger. You can use it to:
  - View all the running processes
  - Step through your code

DDMS



Android on the i.MX53

freescale™ semiconductor

Adeneo Embedded

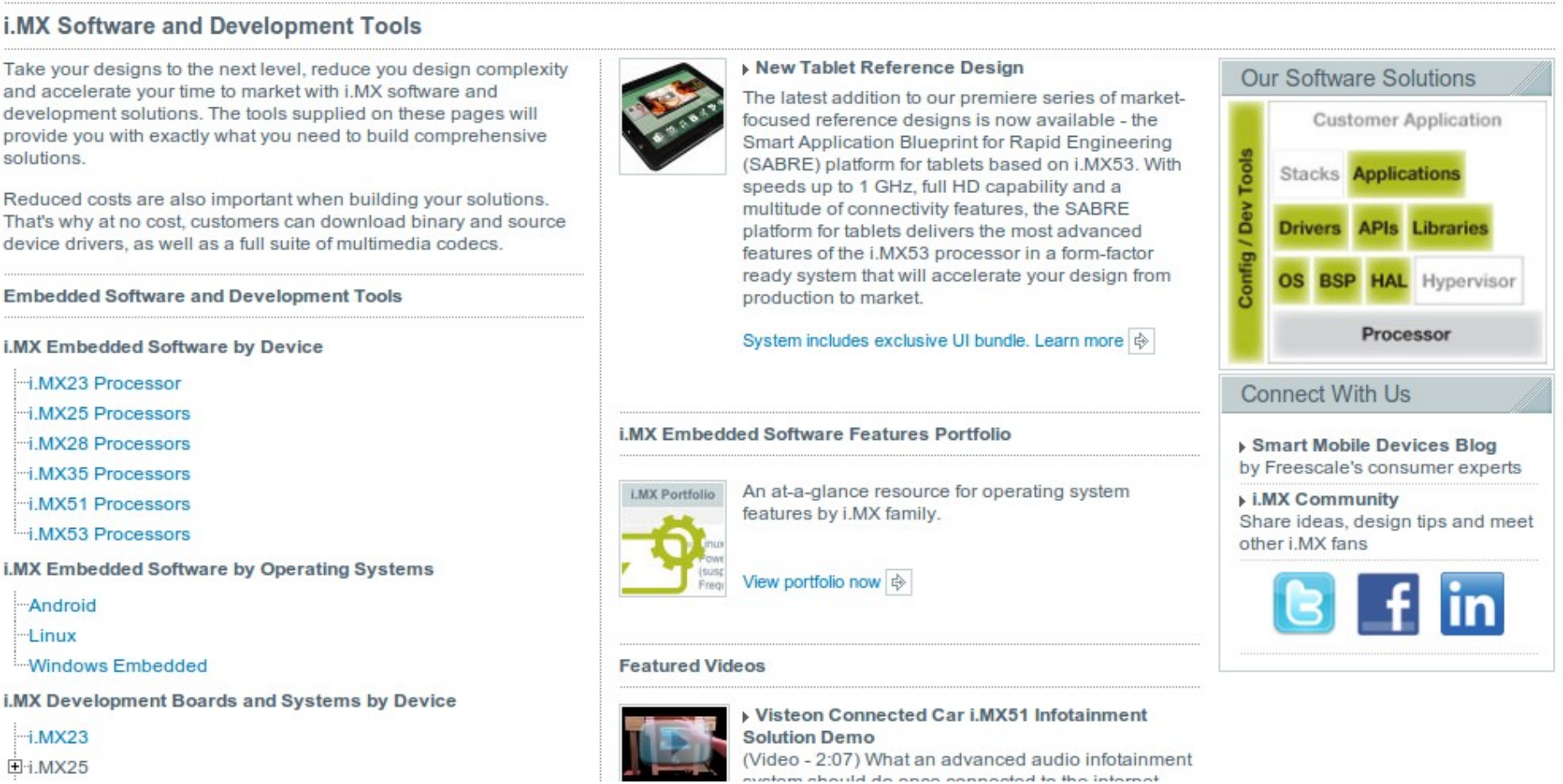
Android i.MX53 BSP features

- Support for the i.MX53 Quick Start Board hardware
- Supports Froyo (Android 2.2)

  Gingerbread (Android 2.3) support coming soon
- Hardware acceleration
  - Graphics acceleration: OpenGL, 2D graphics
  - Multimedia codecs
- Very easy to connect with the Android tools

Resources for Android on the i.MX

- Android BSP sources and images
  - www.freescale.com/imxtools



i.MX Software and Development Tools

Take your designs to the next level, reduce you design complexity and accelerate your time to market with i.MX software and development solutions. The tools supplied on these pages will provide you with exactly what you need to build comprehensive solutions.

Reduced costs are also important when building your solutions. That's why at no cost, customers can download binary and source device drivers, as well as a full suite of multimedia codecs.

Embedded Software and Development Tools

i.MX Embedded Software by Device

- i.MX23 Processor
- i.MX25 Processors
- i.MX28 Processors
- i.MX35 Processors
- i.MX51 Processors
- i.MX53 Processors

i.MX Embedded Software by Operating Systems

- Android
- Linux
- Windows Embedded

i.MX Development Boards and Systems by Device

- i.MX23
- i.MX25

New Tablet Reference Design

The latest addition to our premiere series of market-focused reference designs is now available - the Smart Application Blueprint for Rapid Engineering (SABRE) platform for tablets based on i.MX53. With speeds up to 1 GHz, full HD capability and a multitude of connectivity features, the SABRE platform for tablets delivers the most advanced features of the i.MX53 processor in a form-factor ready system that will accelerate your design from production to market.

System includes exclusive UI bundle. Learn more

i.MX Embedded Software Features Portfolio

i.MX Portfolio

An at-a-glance resource for operating system features by i.MX family.

View portfolio now

Featured Videos

Visteon Connected Car i.MX51 Infotainment Solution Demo

(Video - 2:07) What an advanced audio infotainment

Our Software Solutions

Config / Dev Tools

Customer Application

Stacks Applications

Drivers APIs Libraries

OS BSP HAL Hypervisor

Processor

Connect With Us

Smart Mobile Devices Blog

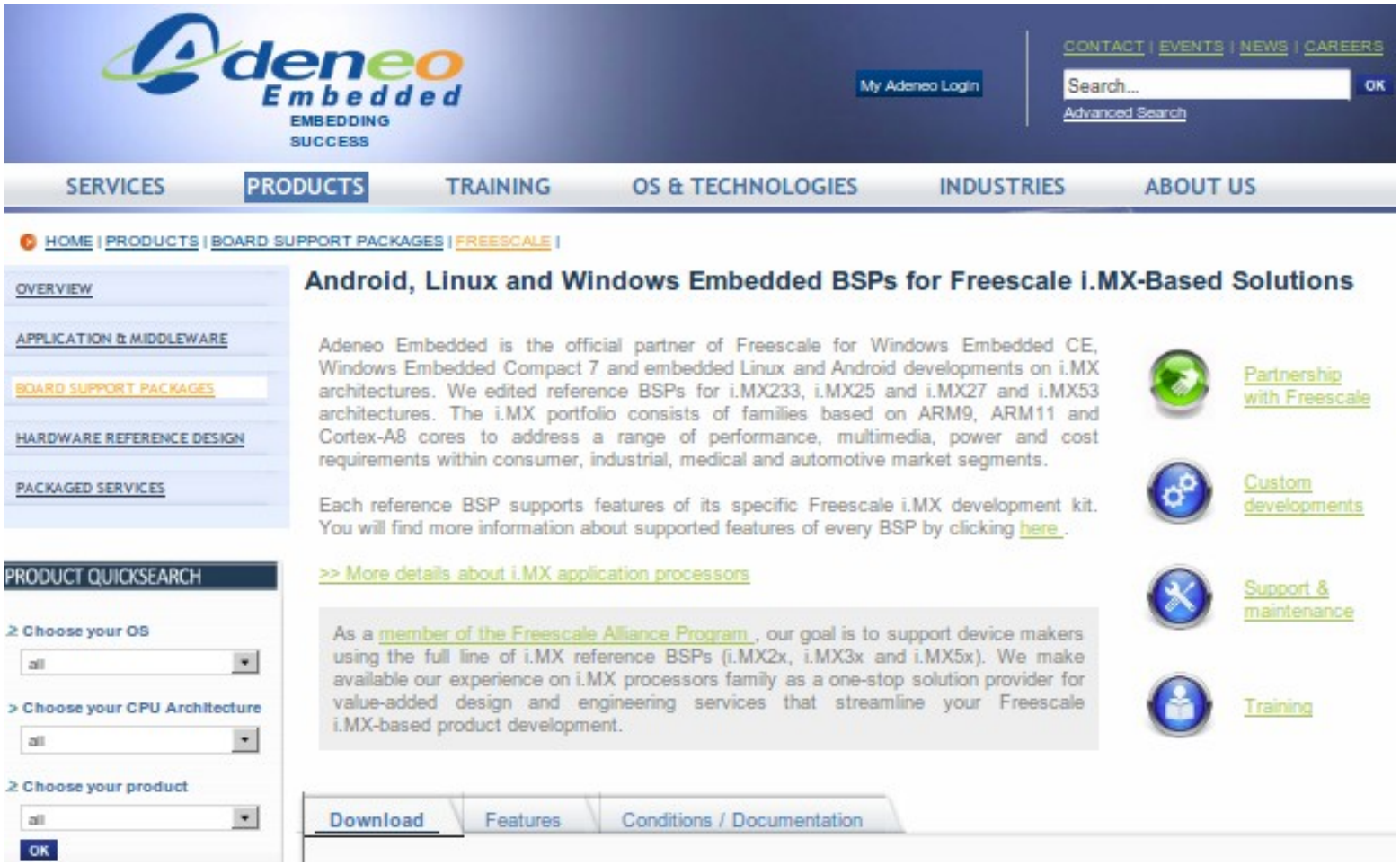
by Freescale's consumer experts

i.MX Community

Share ideas, design tips and meet other i.MX fans

Resources for Android on the i.MX

- Android i.MX53 QSB BSP sources
  - Adeneo Embedded



Resources for Android on the i.MX

- imxcommunity and imxdev.org

ADB

- The Android Debug Bridge (ADB) is a communication channel between the Android tools, and an Android Device (also works with the emulator)
- Connect to your device through:
  - USB
  - Ethernet (for devices with an ethernet connection)
- Using ADB, you can connect to the device for:
  - Debugging
  - Transferring files
  - Installing applications
  - Executing commands via a shell
  - Configuring the device

ADB commands

- adb push
- adb pull
- adb logcat
- adb sync
- adb install
- adb shell
- adb devices

Checking the logs with logcat

- The *logcat* command can be entered at the shell prompt, or can be accessed from DDMS
- Very powerful debug tool, similar to *dmesg* in Linux

- **Usage**:
  - *adb logcat <option> <filter>*
  - Or at the shell prompt:
    - *logcat <option> <filter>*

- Filters are formatted tags and event priority pairs. The format for each filter is:

  <Tag Name>:<Lowest Event Priority to Print>

Checking the logs with logcat

- The severity types(from lowest priority or most verbose to highest priority or least verbose) follow:
- <filter>
  - 1. Verbose (V)
  - 2. Debug (D)
  - 3. Info (I)
  - 4. Warning (W)
  - 5. Error (E)
  - 6. Fatal (F)
  - 7. Silent (S)

Checking the logs with logcat

- The following shell command displays all AppLog-tagged logging information and suppresses all other tags:

  *# logcat –v time AppLog:V *:S*

- You can clear the emulator log using the –c flag:

  # adb –e logcat –c

- You can redirect log output to a file on the device using the –f flag. For example, to direct all informational logging messages (and those of higher priority) from the emulator to the file mylog.txt in the sdcard directory, you can use the following ADB shell command:

  *# logcat -f /sdcard/mylog. txt *: I*

LAB: Using ADB to connect to the Quick Start Board

freescale™ semiconductor

Adeneo Embedded

LAB: ADB

- Boot your i.MX53 Quick Start Board
- On the Android device, go to *Settings* | *Applications* | *Development* and make sure that *USB debugging* is enabled.
- Plug the micro USB cable (right next to the user buttons) between the device and your PC.
- VMware will tell you that a device has been connected. Follow the instructions to make it available in the VM.

LAB: ADB

- In the VM, open a shell:

  > *cd ~/training_mx53/android/sdk/android-sdk-linux_x86/platform-tools/*

  > *./adb devices*

```
List of devices attached
0123456789ABCDEF        device
```

  Note: you can restart the *adb* server by issuing the following commands:

  > *./adb kill-server*

  > *sudo ./adb start-server*

- If you do not see your device, you may need to toggle "*High android phone*" in VMware (look in lower left corner of VMware window)

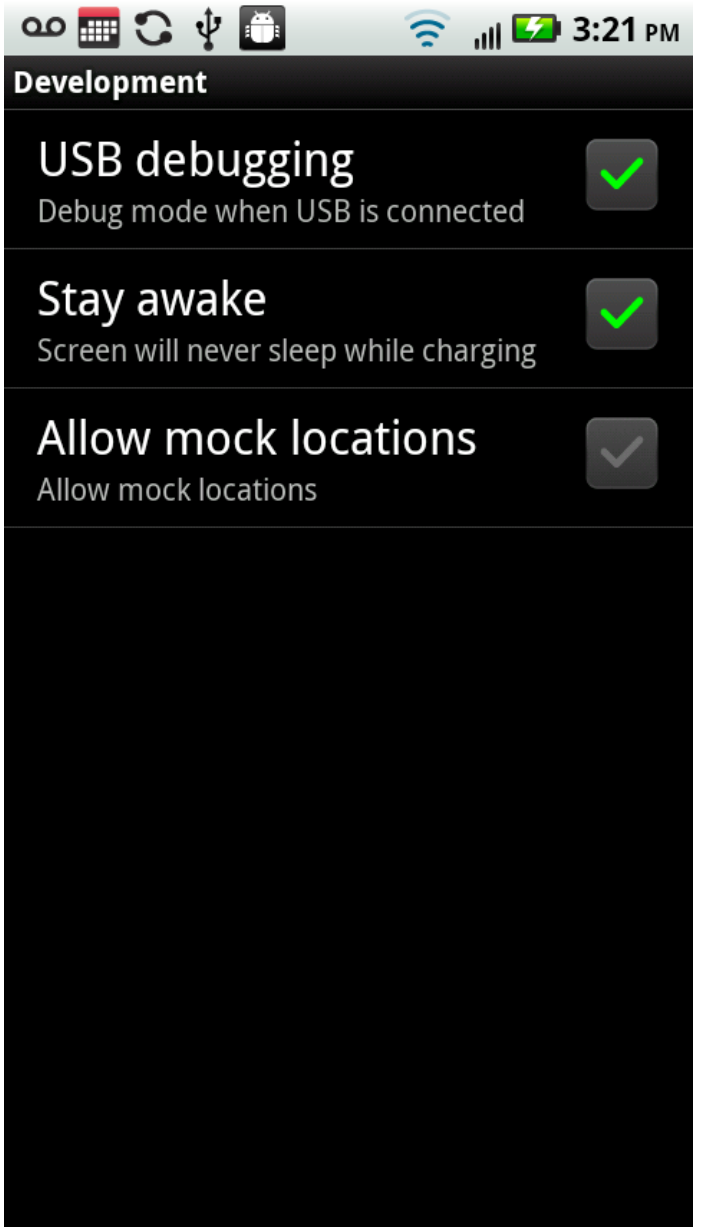
- If the AVD (i.e. the device emulator) is still running, you will see it too

Troubleshooting ADB issues

- Like all debugging connections, it can be difficult to establish ADB communications. Here are some tips to help you get connected.
- Verify your Android device is configured properly.
  - Menu -> settings -> Applications -> development →
    - Check *USB Debugging*
    - Check *Stay awake*
- When you connect the device to the host, YOU MUST ACCEPT THE CONNECTION ON THE DEVICE!

  If the device is locked, you must unlock it.
- If using VMWare, make sure you have connected the device to VMWare.

Troubleshooting ADB issues



LAB: ADB

- We now want to see the logs that were generated on the device.
- From the shell of your development machine, issue the following command:

  > *./adb -d logcat*

  The logs will be printed to the screen.
- You can also open a remote shell on the device:

  > *./adb -d shell*

  Exit by issuing the "*exit*" command.
- Note: you add "-d" to the command line to instruct ADB to direct the requests to the USB-connected device (rather than the emulator)

LAB: Deploy and debug an application on the device

freescale™ semiconductor

Adeneo Embedded

LAB: Deploy the application on the device

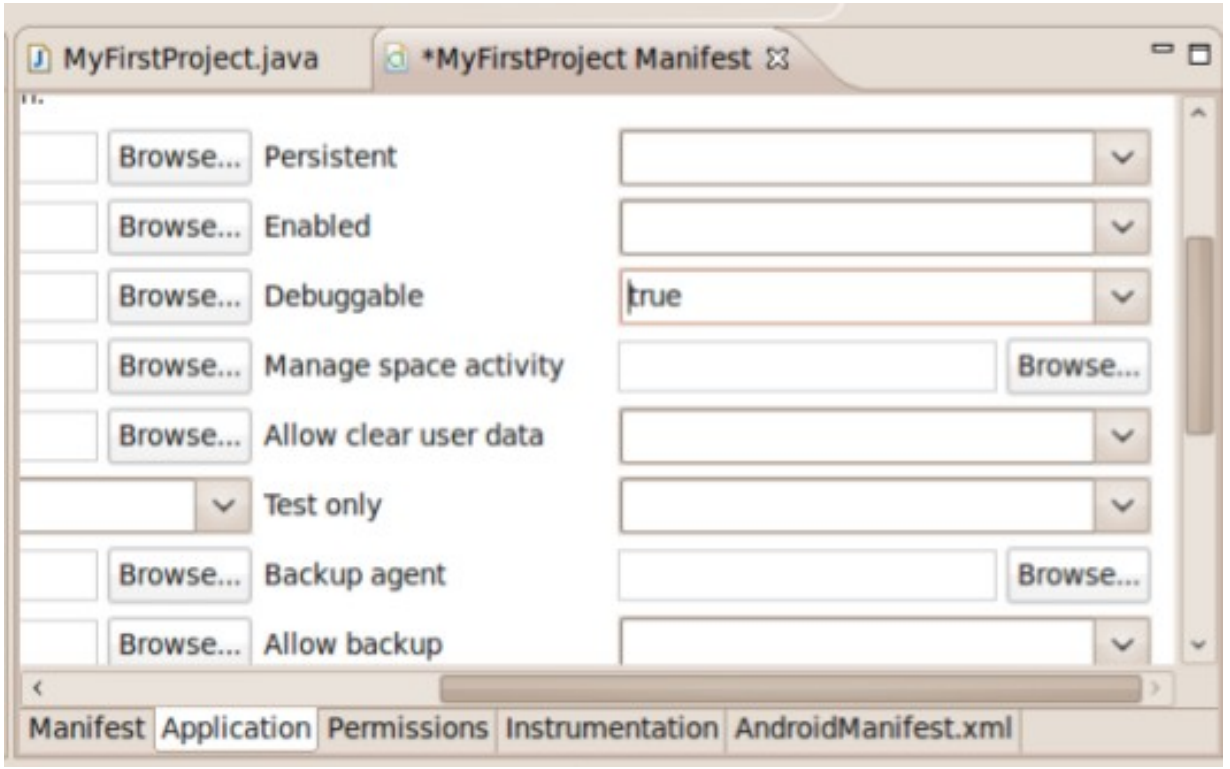
- We will reuse the application that we have tested on the emulator

- To debug on hardware, you have to register the application as debuggable in the manifest file.
- To do this, perform the following steps:
  - Open the *Java* perspective
  - Double-click on the *AndroidManifest.xml* file (in the *Package Explorer* pane)
  - Switch to the *Application* tab
  - Set the *Debuggable* Application Attribute to *true* (right side).
  - Save the file.

  Note: You can also modify the *application* element of the *AndroidManifest.xml* file directly with the android *debuggable* attribute (use the *AndroidManifest.xml* tab):

  <application ... android: debuggable="true">

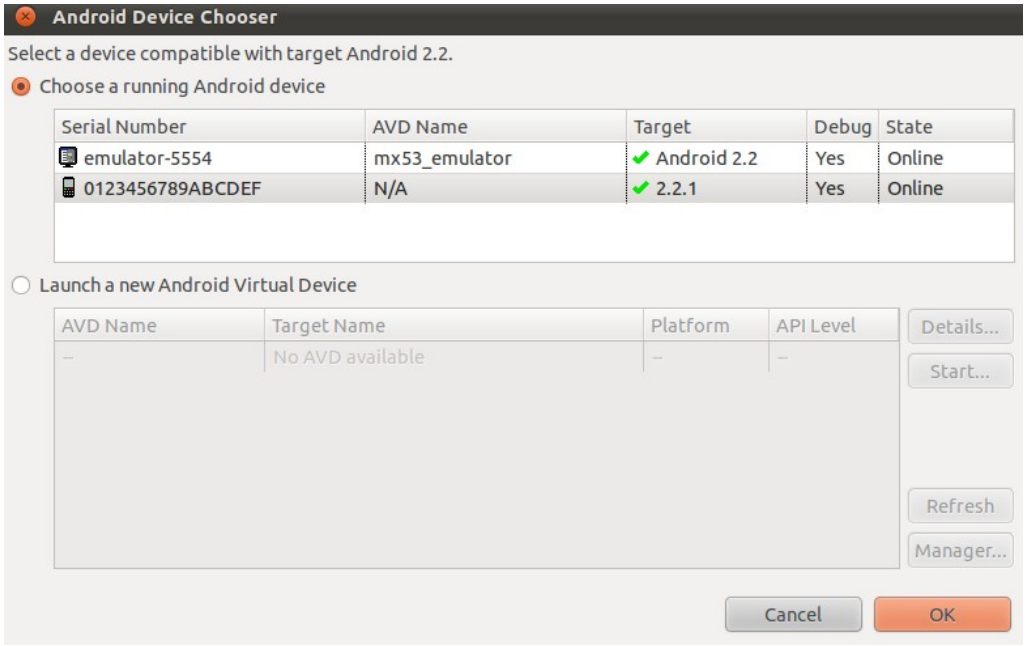
LAB: Deploy the application on the device



LAB: Deploy the application on the device

- From the *Run* menu, choose *Run*
- The *Android Device Chooser* window will appear.

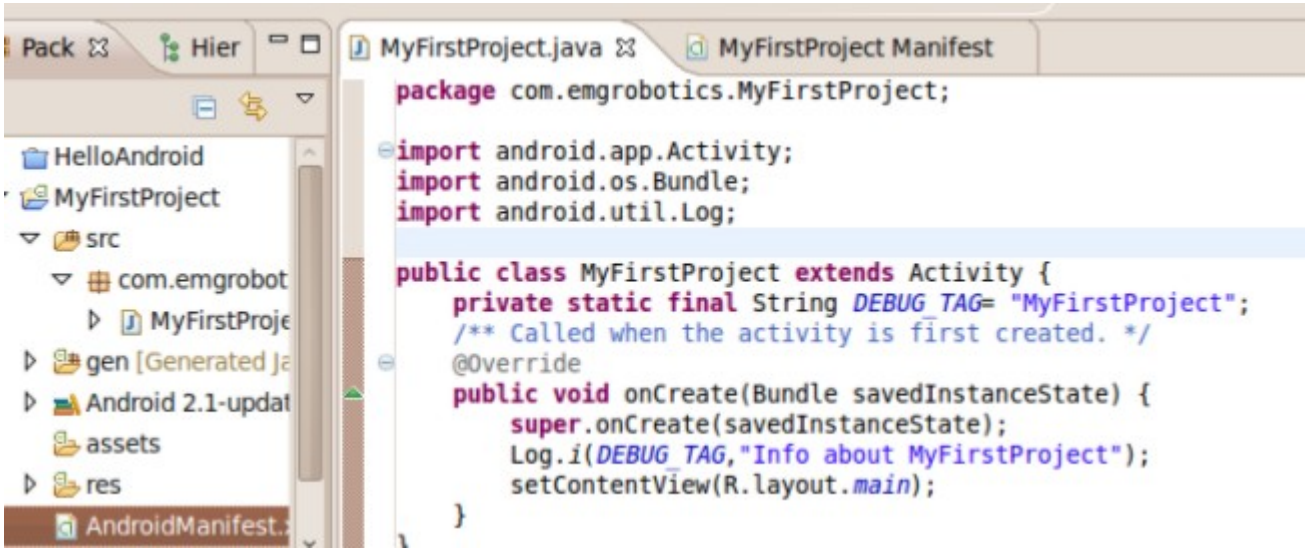
  Choose the *0123456789ABCDEF* device (it is your actual device, not the emulator)



- The application will be deployed and run on the device. Not too hard either!

LAB: Deploy the application on the device

- Add logging code to your application:
  - In the *Package Explorer* pane, expand *src* and *com.example.MyFirstProject*
  - Double-click on *MyFirstProject.java* to edit the file.
  - onCreate() is essentially the entry point of your application (more about activities in the SDK documentation).
  - Replace the file contents (see next page)

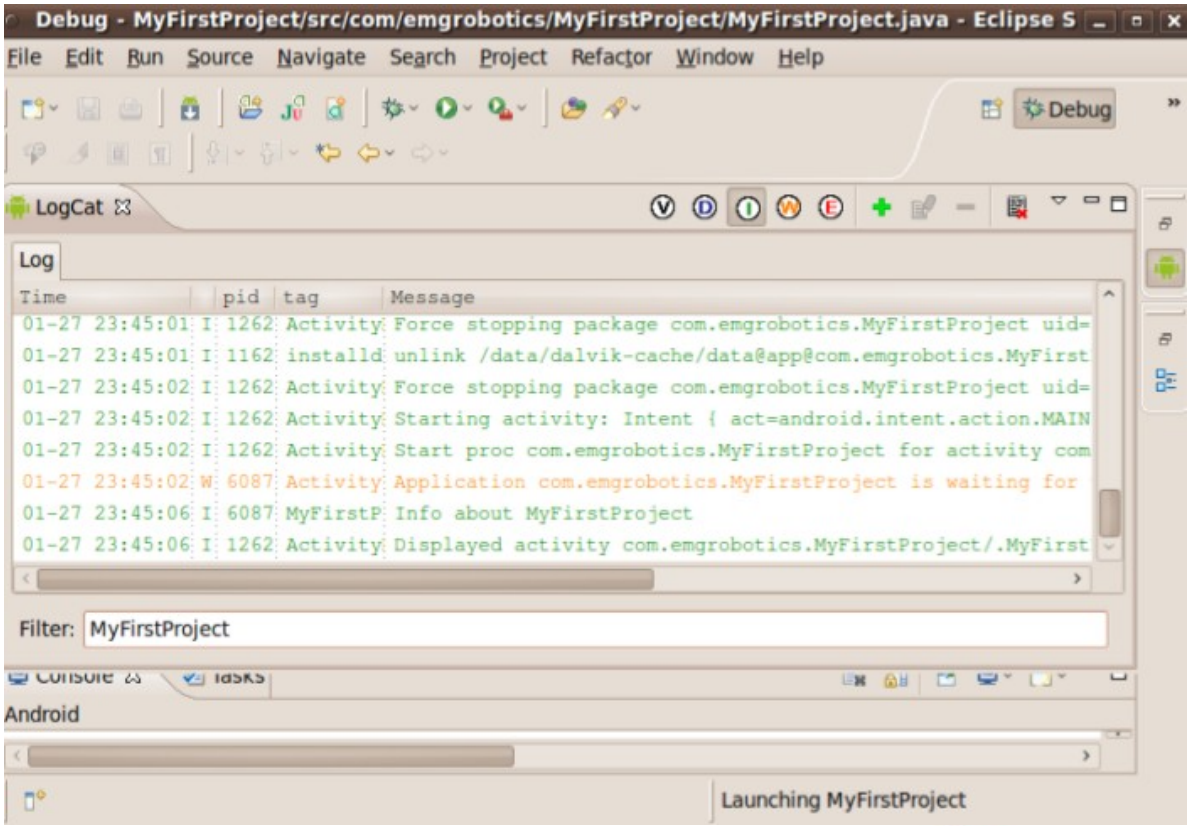


LAB: Deploy the application on the device

```
package com.example.MyFirstProject;

import android.app.Activity;
import android.os.Bundle;
import android.util.Log;

public class MyFirstProject extends Activity {
   private static final String DEBUG_TAG= "MyFirstProject";
   /** Called when the activity is first created. */
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      Log.i(DEBUG_TAG,"Info about MyFirstProject");
      setContentView(R.layout.main);
   }
}
```

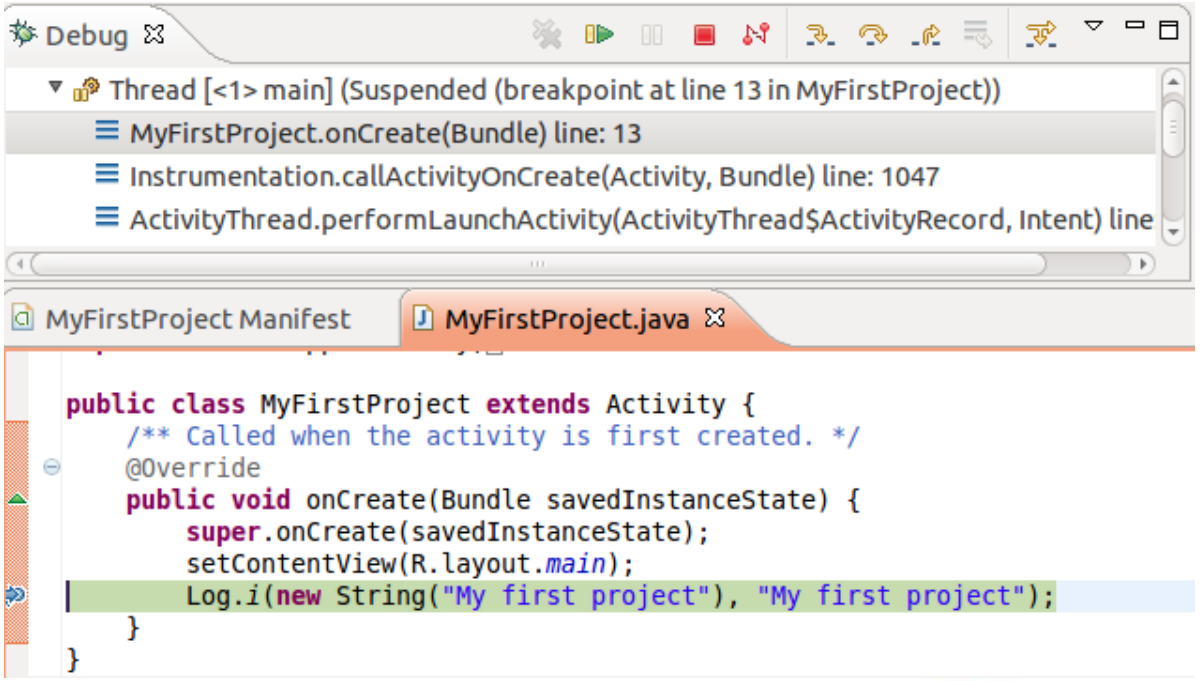
LAB: Deploy the application on the device

- Switch Eclipse to the *Debug* perspective.
- From the *Run* menu select *Debug*.
- The application will be run on the device.
- Check that your message appears in the *LogCat* pane:



LAB: Deploy the application on the device

- Quit the application
- In your code, move the cursor to the line where the log is generated "*Log.i...*"
- From the *Run* menu, select *Toggle Breakpoint*.
- Debug the application again. The application will stop on your breakpoint.



LAB: Building sample applications from the SDK

freescale™ semiconductor

Adeneo Embedded

LAB: Building sample applications

- Create a new Android project
- In the wizard, choose "*Create project from existing source*" and choose *Browse*
- Open */home/freescale/training_mx53/android/sdk/android-sdk-linux_x86/samples/android-8/* ***[PICK A PROJECT]***
- Click on **Finish**
- Build and run the application.

That was easy!

Building native applications with the NDK

freescale™ semiconductor

Adeneo Embedded

Android NDK

- NDK = Native Development Kit

- Allows you to build C and C++ libraries for Android applications
- You can also build C/C++ Executables
  - A bit more difficult to execute (requires a shell)

- Writing native code does not necessarily improve the performance
- The NDK is useful when porting an existing C/C++ codebase

- Uses JNI for binding between Native library and Java apps.

Getting started with the NDK

- Once you have installed the NDK successfully, take a few minutes to read the documentation included in the NDK.
- You can find the documentation in the *<ndk>/docs/ directory*
- In particular, please read the *OVERVIEW.TXT* document completely, so that you understand the intent of the NDK and how to use it.

Working with the NDK

- Place your native sources under *<project>/jni/...*
- Create *<project>/jni/Android.mk* to describe your native sources to the NDK build system
- Optional: Create *<project>/jni/Application.mk*
- Build your native code by running the 'ndk-build' script from your projet's directory. It is located in the top-level NDK directory:

  $ cd <project>

  $ <ndk>/ndk-build

- The build tools copy the stripped, shared libraries needed by your application to the proper location in the application's project directory.
- Finally, compile your application using the SDK tools in the usual way. The SDK build tools will package the shared libraries in the application's deployable .apk file.

- For complete information on all of the steps listed above, please see the documentation included with the NDK package.

NDK samples

- The NDK includes sample applications that illustrate how to use native code in your Android applications:
  - hello-jni — a simple application that loads a string from a native method implemented in a shared library and then displays it in the application UI.
  - two-libs — a simple application that loads a shared library dynamically and calls a native method provided by the library. In this case, the method is implemented in a static library imported by the shared library.
  - san-angeles — a simple application that renders 3D graphics through the native OpenGL ES APIs, while managing activity lifecycle with a GLSurfaceView object.
  - hello-gl2 — a simple application that renders a triangle using OpenGL ES 2.0 vertex and fragment shaders.
  - hello-neon — a simple application that shows how to use the cpufeatures library to check CPU capabilities at runtime, then use NEON intrinsics if supported by the CPU. Specifically, the application implements two versions of a tiny benchmark for a FIR filter loop, a C version and a NEON-optimized version for devices that support it.
  - bitmap-plasma — a simple application that demonstrates how to access the pixel buffers of Android Bitmap objects from native code, and uses this to generate an old-school "plasma" effect.

Conclusion

freescale™ semiconductor

Adeneo Embedded


freescale™
semiconductor


deneo
Embedded