Embedded Systems
a pratical introduction

Ing. Patrick Bellasi

Dipartimento di Elettronica ed Informazione
Politecnico di Milano
bellasi@elet.polimi.it

Last review Oct, 1 2009

Agenda

- Introduction
- Embedded Hardware
  - platforms (micro, SoC, reconfigurable)
  - storage
- Embedded Development Tools
  - building systems
  - version control
  - debugging
- Embedded Software
  - booting and Operating Systems
  - user-space
- References

Introduction
Let's start from a definition

An **embedded system** is a *special-purpose computer* system designed to perform one or a few dedicated functions, often with real-time computing constraints

- It is usually embedded as part of a complete device including hardware and mechanical parts
- In contrast, a general-purpose computer, such as a PC, can do many different tasks depending on programming
- Embedded systems control many of the common devices in use today

Wikipedia
http://en.wikipedia.org/wiki/Embedded_system

Introduction
Many different systems...

- A very generic definition
  - covers very different types of systems
  - fuzzy border with “standard” systems

- *Consumer electronics (CE) products*
  - home routers, DVD players, TV sets, digital cameras, GPS, camcorders, mobile phones, microwave ovens...
- *Industrial products*
  - *machine control, alarms, surveillance systems, automotive, rail, aircraft, satellite...*

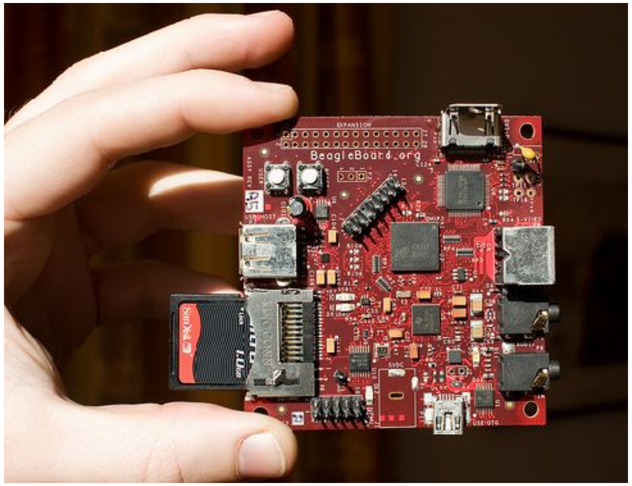
User-space applications

Middleware Library | Middleware Library | Middleware Library

Standard C Library

#include <stdlib.h>

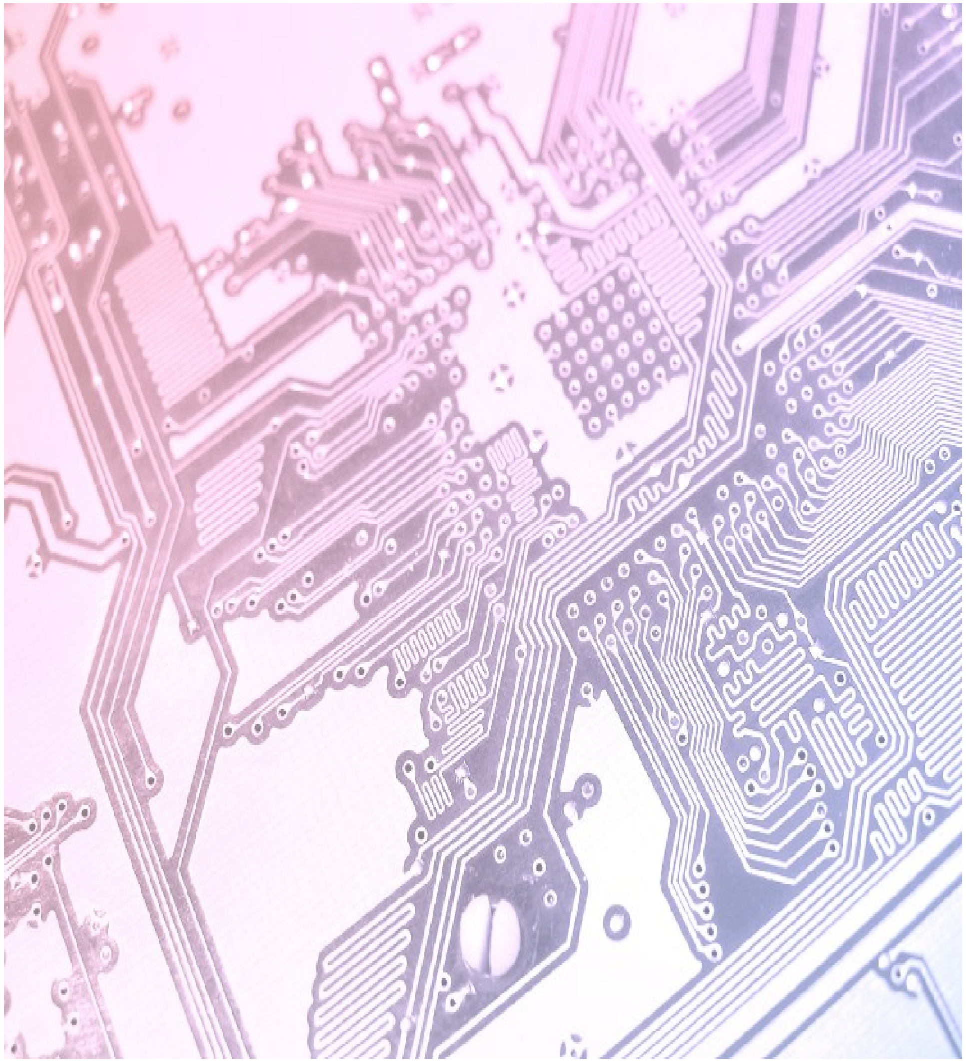
Operating System



Bootloader

Hardware

Tools



Embedded Hardware

Embedded Hardware
Characteristics

- Different from hardware for classical systems
  - CPU architecture
    - *ARM, MIPS or PowerPC; x86 is also used (PXA, Atom)*
  - flash memory for storage
    - *NOR or NAND type*
    - *limited capacity (from a few to hundreds of MB)*
    - *different access mode (erase page based)*
  - limited RAM capacity
    - *from a few MB to several tens of MB*
  - many interconnect bus not often found on the desktop
    - *I2C/TWI, SPI, SSP, CAN, etc.*
- Development boards
  - starting from a few hundreds of EURO
  - often used as a basis for the final board design

Embedded Hardware
Examples 1/4 – Ready to use embedded systems

- Picotux 100

  35mm × 19 mm × 19 mm

  55 MHz 32-bit ARM7

  Netsilicon NS7520 uP

  2 MB of Flash Memory

  8 MB SDRAM Memory

  1 Eth, 5 GPIO, 1 TTY

  µClinux 2.4.27 (750 KB)

  BusyBox 1.0 (shell)

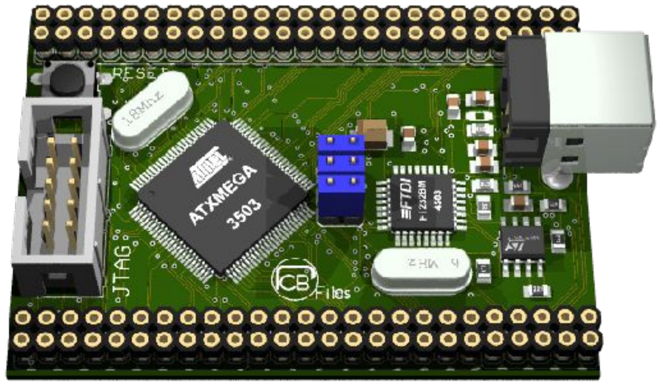
  250 mA only and 3.3 V

http://www.picotux.com

- ATxmega256
  - AVR XMEGA
    - *8/16-bit uP*
    - *32 MIPS @ 32 MHz*
  - 256K Flash
  - 16K SRAM
  - 50 GPIO
  - 7x16bit timers
  - 4 SPI, 2 TWI, 7 USART
  - 8 ADC 12bit, 2 DAC 12bit

  - 4 DMA channels
  - 8 event system channels
  - crypto engine (AES, DES)

ATSTK600 - Atmel's starter kit

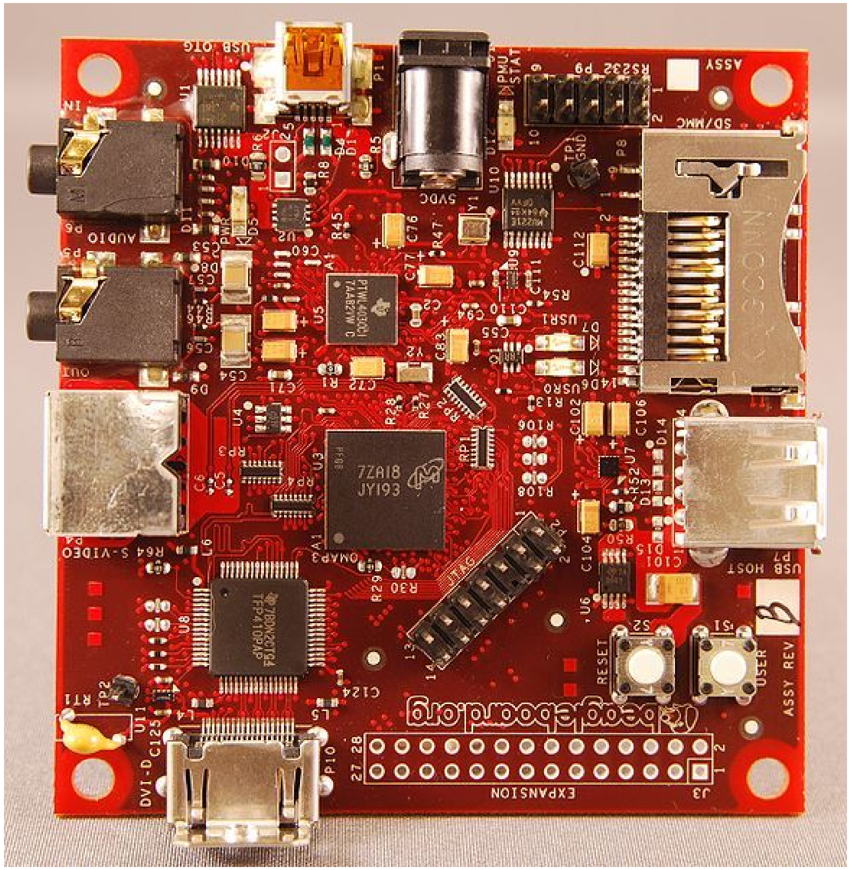


Example of custom board

Embedded Hardware
Examples 3/4 – SoC based

- Beagleboard
  - OMAP3530 SoC
    - *600 MHz ARM Cortex-A8*
    - *PowerVR SGX530 GPU*
      - *OpenGL ES 2.0*
    - *430MHz MS320C64x+ DSP*
      - *HD capable*
    - *IVA2 Accellerator*
  - POP CPU/Memory chip
    - *256MB of NAND*
    - *256MB of RAM*
  - Rich set of peripherals
    - *S-Video, HDMI, SD/MMC, USB OTG, RS-232, JTAG*

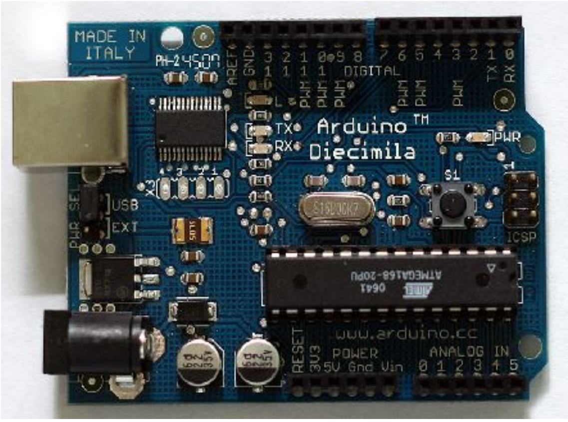
  2W @ 5V



beagleboard.org

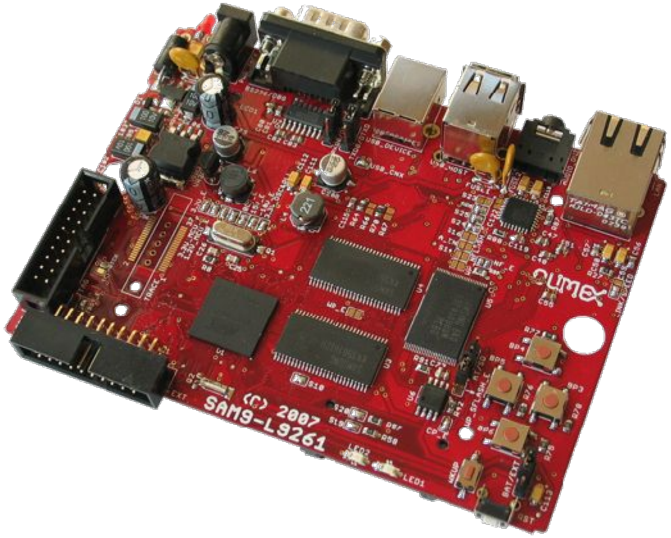
Embedded Hardware

Examples 4/4 – Many other development platforms

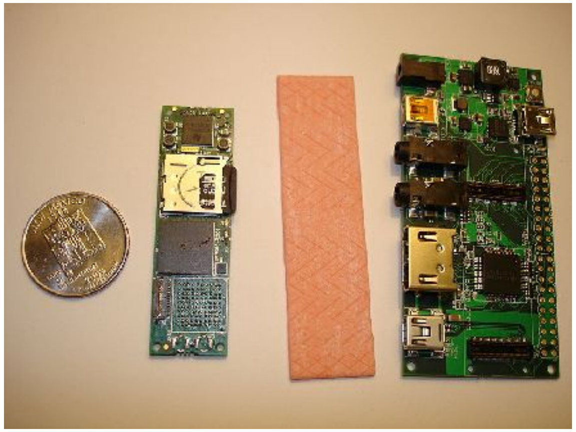


Arduino Diecimila

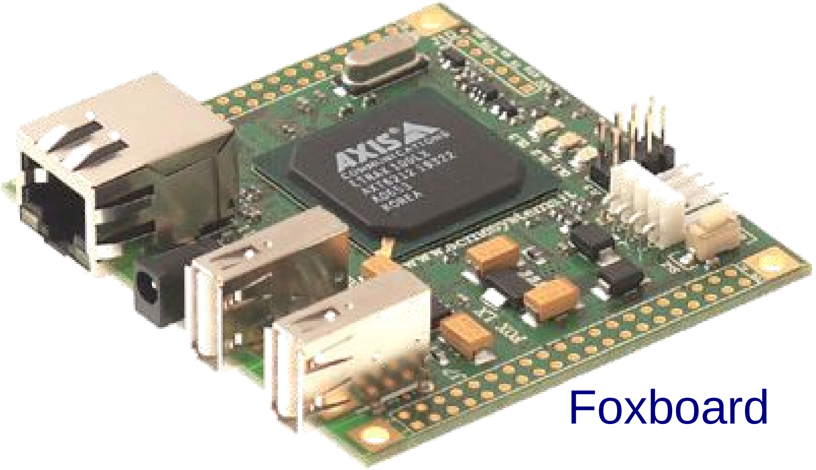
Micaz



Olimex SAM9-L9261



Gumstix Overo



Foxboard

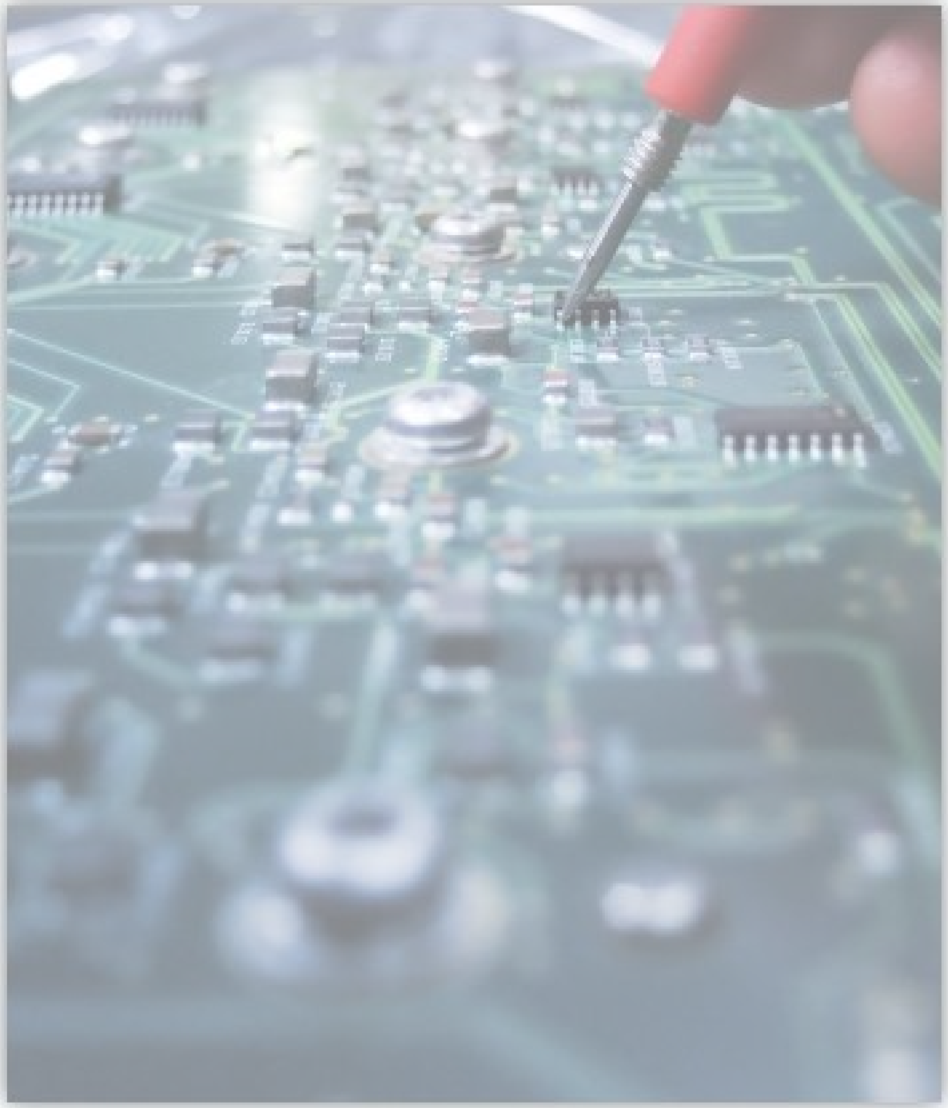
- Raw flash storage
  - no hardware takes care of “wear leveling”
  - writes across flash sectors (erase pages)
- Memory Technology Devices (MTD)
  - specific filesystems must be used
    - *JFFS2, UBIFS, LogFS, AXFS*
  - some FS have specific purpose
    - *read-only parts: SquashFS, cramfs*
    - *temporary data: tmpfs*

Flash			RAM
kernel	**read-only compressed data** *squashFS, cramFS*	**read-write data** *jffs2, ubifs*	**temporary data** *tmpfs*

- Support off-hardware development and testing

- QEMU allows to emulate many CPU architectures
  - X86, PowerPC, ARM, MIPS, SPARC, etc.
  - command: `qemu-system-ARCH`
- Full system emulation
  - CPU, RAM and devices
  - for each architecture, several platforms are proposed
    - *ARM: Integrator, Versatile, PDA Sharp, Nokia N8x0, Gumstix, etc.*
    - *`qemu-system-arm -M ?`*

- http://bellard.org/qemu/



Development Tools

- Essential tool for embedded development
- Tools running on **host**, handling code for **target**
  - binutils: ld, as, nm, readelf, objdump, etc.
  - standard C library: glibc, uClibc or eglibc
  - C/C++ compiler & debugger: gcc & gdb
  - math libraries: gmp, mpfr
- How to get one?
  - Hand made
    - *configure and compile all the components in the right order*
    - *gcc and binutils are not always bug free: need to apply patches*
  - Pre-compiled
    - *Code Sourcery is a renowned supplier*
  - Generated by scripts
    - *Crosstool-ng, Buildroot, Openembedded*

- Cross-compilation can sometimes be tedious
  - different compiling tools (arm-linux-gcc instead of gcc)
  - files are not installed in usual paths
    - *e.g. binaries, libraries, pkgconfig configuration files, includes, etc.*
  - cross-compiled code cannot bu run on the host machine
- Some knowledge on major build systems is useful
  - how they handle the cross-compilation
    - *useful to fix build issues*
- Autotools
  - autoconf, automake, pkgconfig, and libtool
  - is still the most used one today
- CMake
  - a new generation build system

- Simplified usage description

  autoconf - handle the configuration of the software package

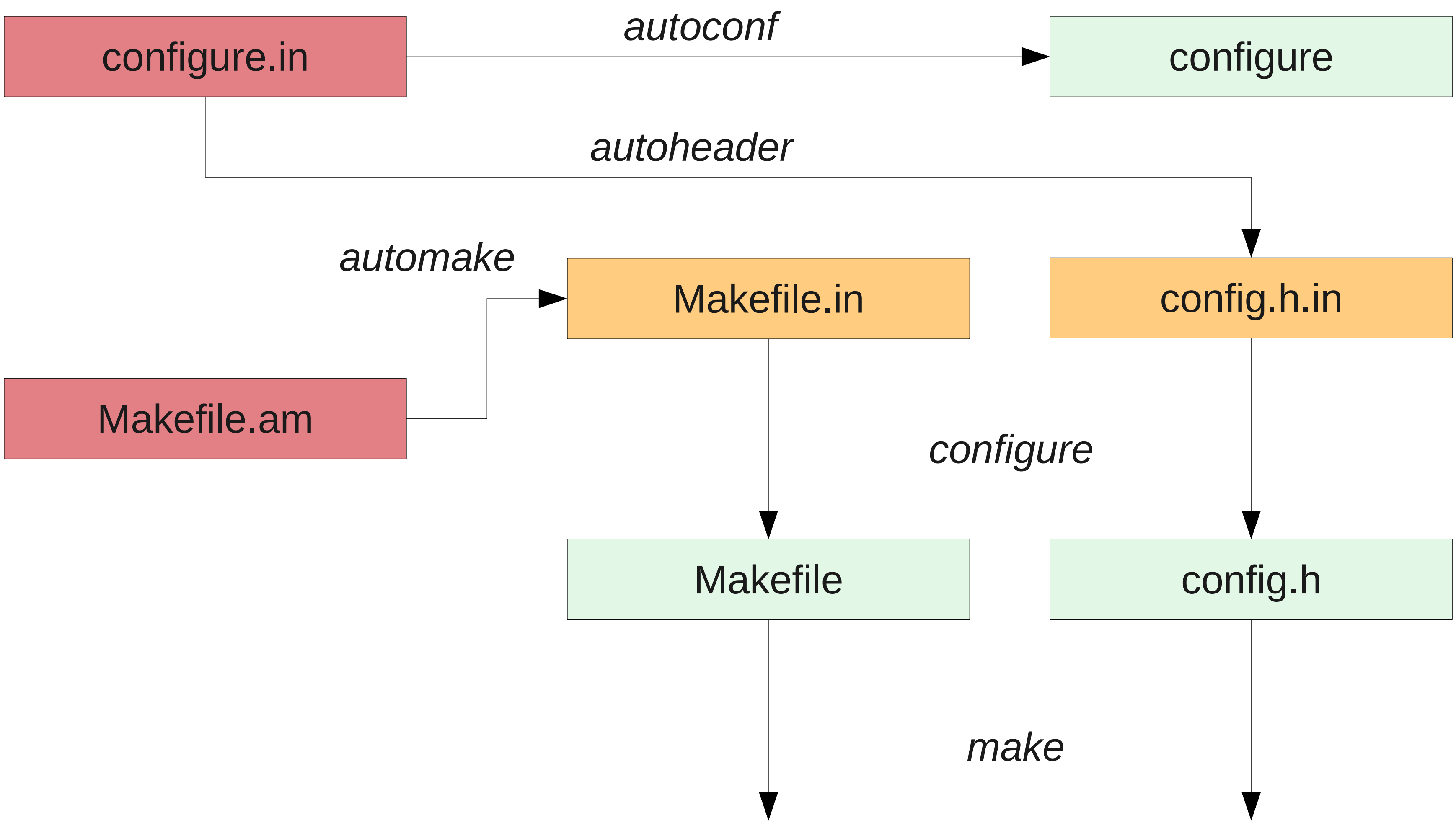
  automake - generate the Makefiles needed to build the software package

  pkgconfig - ease compilation against already installed shared libraries

  libtool - handle the generation of shared libraries in a system-independent way

- Most of these tools are old and relatively complicated to use, but they are used by a majority of free software packages today
- One must have a basic understanding of what they do and how they work

Embedded Development Tools
Building tools – Autotools 2/2



- Cmake, Cross Platform Make
  - used by large projects such as KDE or Second Life
  - much newer and simpler than the Autotools
  - supports cross-compilation

- Many others
  - Waf
  - Scons

- Scratchbox solves some “operative” cross-compilation issues
    - supported platforms: arm, x86 (ppc, mips and cris)
- Benefits
    - chrooted environment
        - *you are still on the host, but you only see the target files*
    - transparent cross-compiling
        - *the cross-compiler looks like a native one*
    - transparent execution
        - *either through remote execution on the target or through CPU code emulation (qemu)*
- Drawbacks
    - no infrastructure for build reproduction
    - requires modified toolchains, only old ones released

Embedded Development Tools
Building systems – Automatic building 1/2

- Automate the process of building a target system
  - kernel, applications and sometimes the toolchain too
- Automatically download, configure, compile and install all components
  - satisfy dependencies: using the right order
  - fix cross-compiling issues: applying patches
- Support a large number of packages
  - should fit main requirements, are easily extensible
- Builds become reproducible
  - allows to easily change the configuration of some components, upgrade them, fix bugs, etc.

Embedded Development Tools
Building systems – Automatic building 2/2

Buildroot - Making Embedded Linux easy
community developed

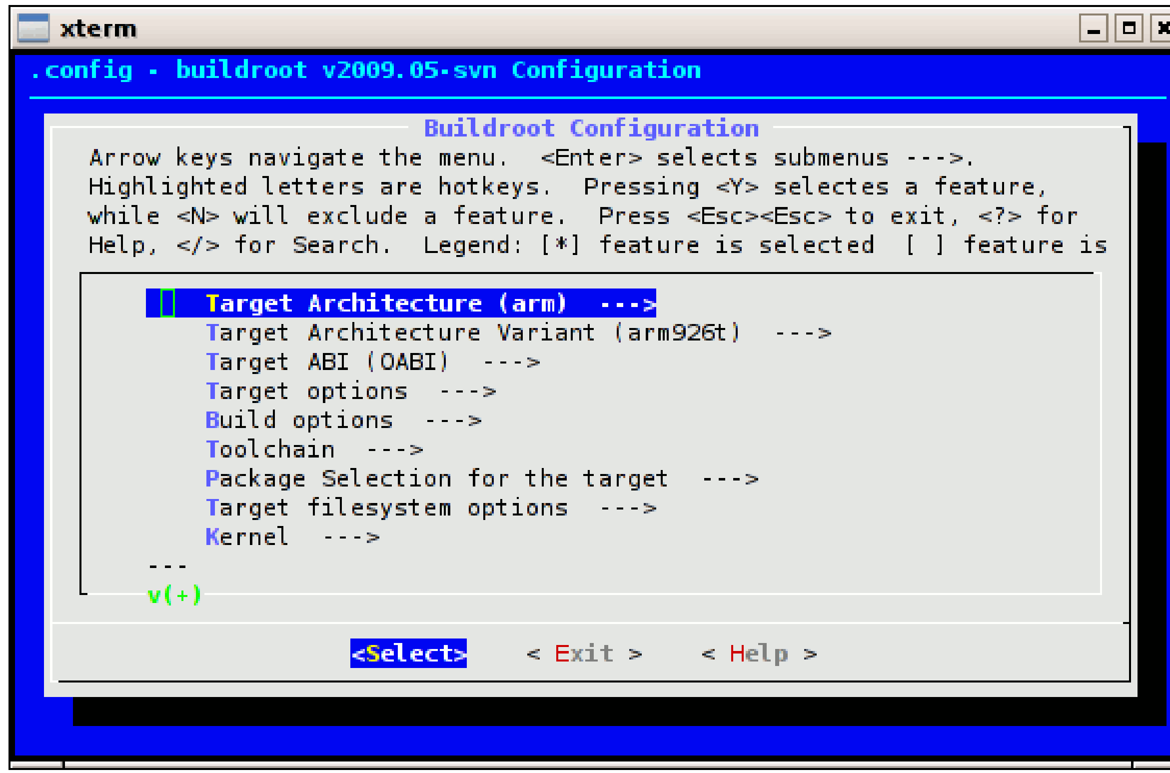
PTXdist - Reproducable Embedded Linux Systems
developed by Pengutronix

LTIB - Linux Target Image Builder
developed mainly by Freescale

OpenEmbedded - the build framework for embedded Linux
more flexible but also far more complicated

Gentoo Embedded

- Simple menuconfig interface to create the configuration
  - target architecture
  - toolchain
  - packages
- make menuconfig
- make



- See free-electrons' presentation for details

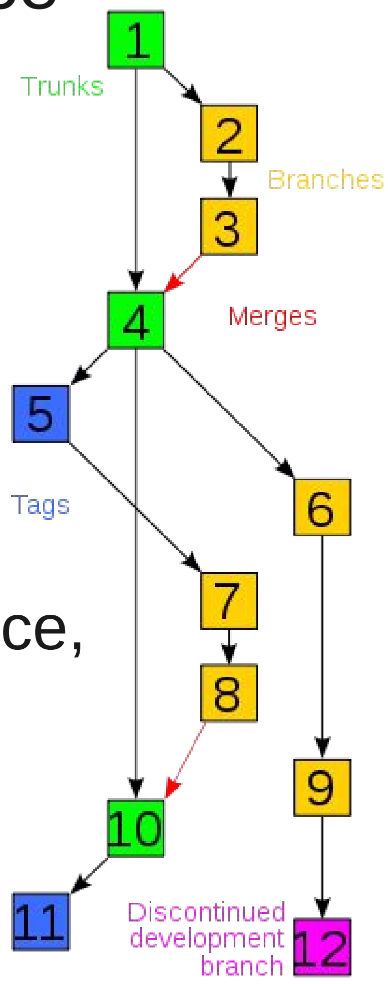
Embedded Development Tools

Building systems – Openembedded

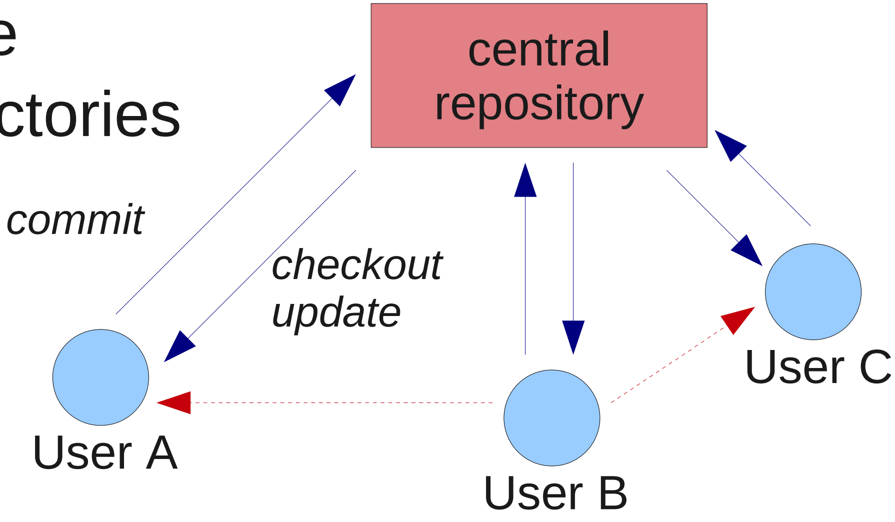
- Use a self contained cross-compiling environment
  - bitbake, python coded
- Generates everything from scratch
- Use package descriptions
  - bitbake recipes (metadata)
- Describing how to build
  - packages (applications, libraries, kernels, bootloaders...)
    - *~1900 packages in recipes/<tool>/*
    - *~7400 package versions in packages/<tool>/*.bb*
  - target machines
    - *248 machines defined in conf/machine/*
  - distributions: machine and package configurations
    - *39 machines defined in conf/distro*

statistics updated to October 2009

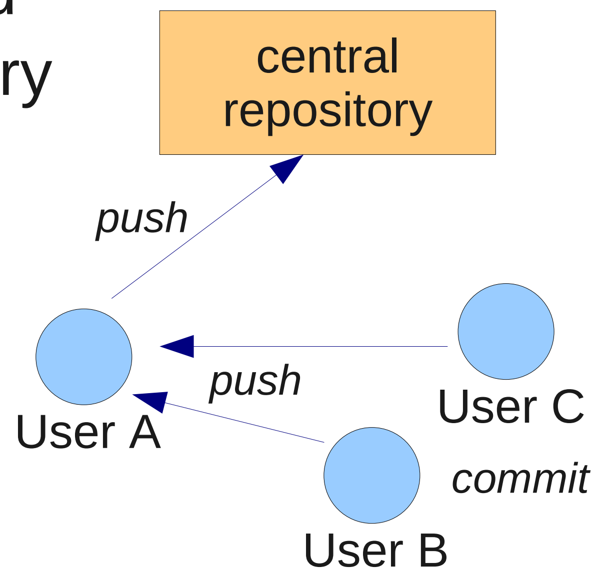
- Fundamental tools to support Open Source development
  - collect contributions from different coders
    - *who does what?... for merits and blaming! ;-)*
  - track code revisions
  - fork projects
    - *make experiments*
- Many alternatives, differences in:
  - repository models, cuncorrency models, licence, supported platforms, costs
    - *Bazaar, BitKeeper, ClearCase, CVS, GIT, Mercurial, Monotone, Subversion, …*
- Opensource is moving thowards
  - *Distributed Version Control Systems*



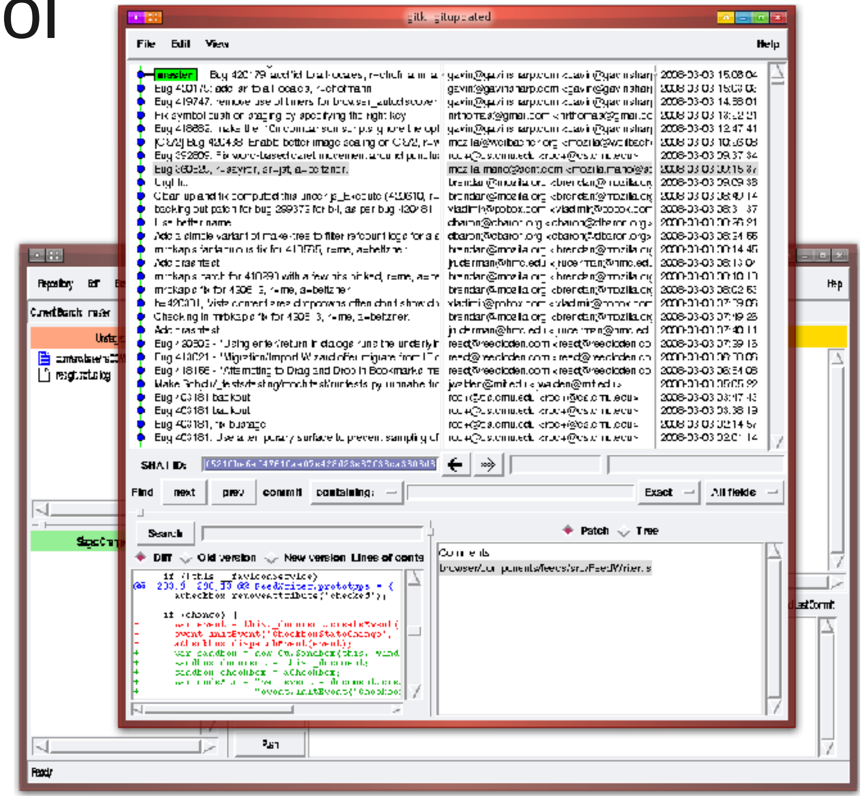
- Why Central VCS are not satisfying?
  - branching is easy but merging is a pain
    - *Subversion has no history-aware merge capability*
    - *forcing its users to manually track exactly which revisions have been merged between branches making it error-prone*
  - no way to push changes to another user
    - *without submitting to the central server*
  - subversion fails to merge changes on renamed files
  - offline commits are not possible
  - .svn files pollute your local directories
  - poor performance



- Why distributed VCS are satisfying?
  - no canonical, reference copy of the codebase
    - *only working copies*
  - disconnected operations
    - *fast common operations, e.g. commits, history, diff and reverting changes*
    - *a central server can exist for stable, reference or backup version*
  - each working copy is effectively a remoted backup of the codebase and change history
    - *providing natural security against data loss*
  - experimental branches
    - *reating and destroying branches are simple operations and fast*
  - collaboration between peers made easy

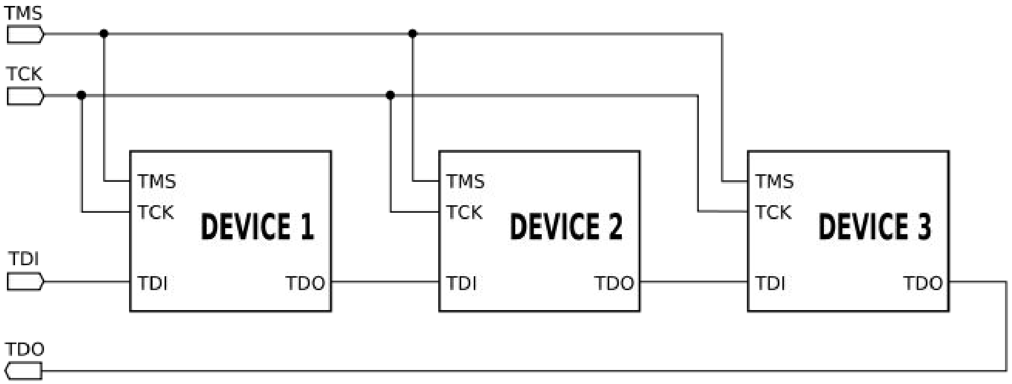


- free distributed revision control
- developed by Linus Torvalds
  - Linux kernel development
- basic design principles
  - *take CVS as an example of what not to do*
  - *support a distributed BitKeeper-like workflow*
  - *strong safeguards against corruption*
  - *very high performance*

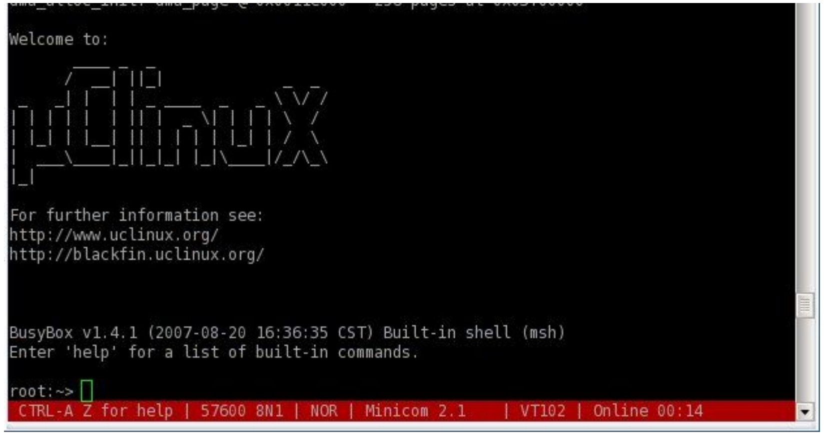


- How to verify an HW circuit is working properly?
  - JTAG - Standard Test Access Port and Boundary-Scan Architecture
    - *for low level parts, bootloader and kernel*
  - bus allowing to directly control the CPU
    - *a probe connects the board to the host machine, generally used through gdb*
  - typical usage scenarios
    - *step-by-step debugging (instruction level)*
    - *chip external connections test*
    - *flash programming*

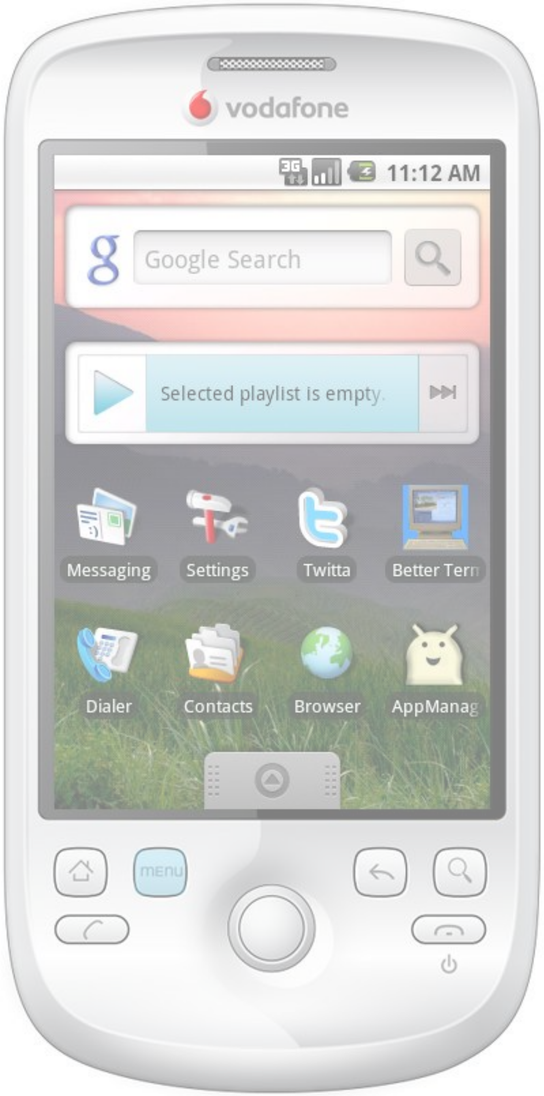
- OpenOCD
  - opensource tool
- Olimex
  - chip jtags probes



- More simpler solution: using the console
  - always accessible via serial connection, immediately after system boot
  - file transfer support
  - need a terminal emulator
    - *e.g. screen, minicom, cutecom,...*



- Better: gdbserver on the target board
  - gdb runs on the development host
    - *compiled to support the target CPU*
  - it controls the application execution remotely



Embedded Software

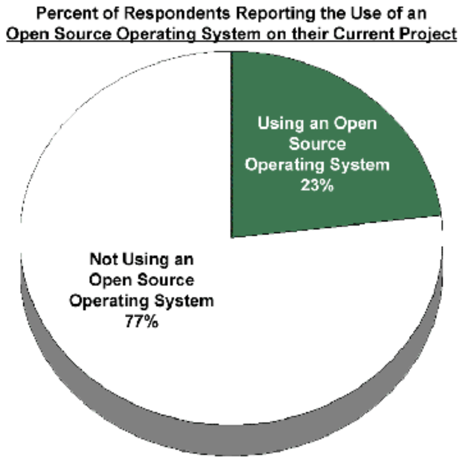
Embedded Software
Bootloader

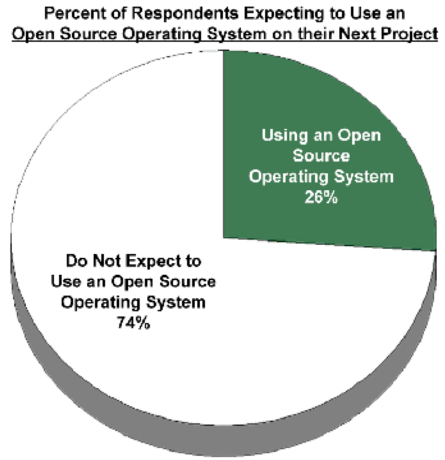
- No BIOS on embedded architectures
  - the bootloader must properly initialize the HW
    - *required to boot an OS: e.g. RAM controller*
- At power-up: CPU starts to execute at a fixed address
  - hardware design: part of flash is mapped at this address
    - *the bootloader entry point is stored at that address*
- Takes control on the hardware right from power-up

- Das U-Boot: most popular free software bootloader
  - wide number of architectures support
  - easy to configure and modify
  - it is a powerful boot monitor
    - *interactive prompt: kernel and FS load via network, flash handling, HW diagnostic, start execution*

- Not all embedded system have a kernel
    - most microcontroller based application run a single binary
- Many embedded application require an ad-hoc kernels
    - wireless sensor networks (WSNs): TinyOS
        - *open source component-based operating system*
    - real-time embedded system market
        - *QNX, commercial, unix-like microkernel*
        - *RTLinux, hard realtime RTOS microkernel, runs Linux a fully preemptable process*
        - *VxWorks, by Wind River Systems (now Intel, since July 2009)*
        - *Windows CE, component-based, deterministic interrupt latency*
- Many and many embedded applications use Linux

- Open source OSs usage by embedded engineers
  - Linux: 18% (15% in 2004)
  - Others: 5%
    - *eCos, BSD, FreeRTOS, and TinyOS*
- Reasons:
  - Licensing cost advantages
  - Flexibility of source code access
  - General familiarity
  - Ecosystem of applications and tools
  - Growing developer experience
- Study predicts strong growth for mobile Linux





Source: VDC's annual 2008 Embedded Software Market Intelligence report

Embedded Software
Linux – Basic characteristics

- Basic system components
  - managing processes, memory, filesystems, protocols, networking, etc.
- Contains drivers for most devices
- Different levels for embedded hardware support
  - architecture, e.g. ARM
  - machine, e.g. Atmel AT91
  - board, e.g. sam9261ek
- Virtual memory: virtual addresses, no relocation
- Memory protection: safety and HW abstraction
- On-demand paging (using MMU): optimized memory usage
- Many many others...

- Usually ported on a board by its manufacturer
- Default configuration for each machine
- Cross-compilation require few basic steps
  1. get kernel sources
  2. apply patches if needed
  3. configure the architecture and cross-compiling toolchain
     export ARCH=<target architecture>
     export CROSS_COMPILE=<toolchain-prefix>
  4. use a ready-made configuration
     make <targetmachine>_defconfig
  5. compile
     make
- Result: compressed kernel image
  e.g. on ARM: `arch/arm/boot/zImage`

- The base library above the kernel
  - it is part of the cross-compiling toolchain
  - feature-rich API to program non-graphical applications
- Different solutions
  - glibc (GNU Libc)
    - *standard, used in all desktop and server systems*
    - *full features => big memory footprint (~400K)*
  - uClibc
    - *complete rewrite of a simpler libc*
    - *optimized for size (stripped C++ support), configurable features ()*
  - EGLIBC
    - *“variant” of glibc, more configuration flexibility (e.g. Debian, ArcLinux)*
  - Bionic
    - *BSD licensed, small footprint (~200K), fast code paths (e.g. pthread)*
    - *developend by Google for Android*

- Provide a basic set of utilities for the target system
  - e.g. cp, ls, mv, mkdir, rm, tar, mknod, wget, grep, sed
- Standard GNU tools *not* designed for embedded
  - too many utilities: fileutils, coreutils, tar, wget, etc.
  - full featured => big memory footprint
- BusyBox provide a better solution
- All the utilities (+200) in a *single* binary program
  - symbolic links to use them as usual
  - single binary + reduced features => small footprint
    - *reduces the executable file format (e.g. ELF) overheads*
  - extremely configurable

"The Swiss Army Knife of Embedded Linux"

- Low-level graphical solutions
  - framebuffer, managed by the Linux kernel
  - DirectFB, more convenient programming interface
  - X.org Kdrive, simplified X server
  - Nano-X

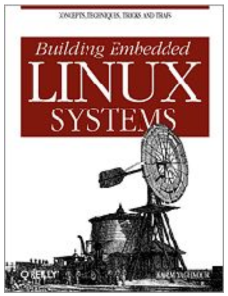
- Higher-level graphical solutions
  - Qt, on top of the kernel framebuffer, or using an X server
  - GTK, on top of DirectFB or using an X server
  - WxEmbedded, on top of X, DirectFB or Nano-X

Embedded Software
Applications and tools

- In theory, all the free software tools and libraries can be cross-compiled and used on an embedded platform.
  - once the system is in place, it's just Linux
- In practice, cross-compiling is often difficult
  - because not anticipated by original developers
- Properly used autotools are the best way to make software cross-compiling aware
  - though they have many shortcomings
- Dedicated tools for platforms with limited resources
  - OpenSSH as ssh server and client => Dropbear replacing
  - Apache => several reduced HTTP servers

- Customizations of desktop distributions
  - emdebian (available for ARM, MIPS and PowerPC)
  - Embedded Gentoo
- Distributions designed for specific devices
  - Ångström
    - *targets PDAs and webpads (OpenZaurus, OpenSimpad...)*
    - *Nokia: “easy start for Beagleboard”*
  - Rockbox
    - *portable media players (e.g. iPods)*
  - Poky by OpenedHand
    - *GNOME-based Linux distribution*
    - *Sato (GTK based) graphical application framework*
- Meta-distribution: Openembedded
  - *the* software framework to create Linux distributions

References
Useful readings

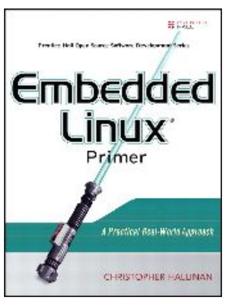


- *Building Embedded Linux Systems*

  Karim Yaghmour at al.

  O'Reilly Media, Aug. 2008

  a good starting point for embedded engineers

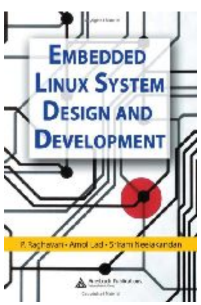


- *Embedded Linux Primer*

  Christopher Hallinan

  Prentice Hall, Sep. 2006

  covers a very wide range of interesting topics



- *Embedded Linux System Design and Development*

  P. Raghavan, A. Lad, S. Neelakandan

  Dec. 2005

  useful book covering most aspects of embedded Linux system development (kernel and tools)

References
Useful website

- Embedded Linux Wiki

  present information about the development and use of Linux in embeddedsystems

- LinuxDevices.com

  weekly newsletter with news and announcements about embedded devices running Linux

- LWN.net

  weekly newsletter presenting kernel developments

- The DENX U-Boot and Linux Guide

  generic help and advice for embedded Linux systems

- Linux Kernel Newbies

  the starting point for aspiring Linux kernel developers

References
International conferences

- Embedded Linux Conference
  - organized by the CE Linux Forum
    - *in California (San Francisco, April)*
    - *in Europe (October-November)*
  - Very interesting kernel and userspace topics for embedded systems developers. Presentation slides freely available
- Ottawa Linux Symposium
  - kernel and system development presentations
  - *freely available proceedings*
- Linux Plumbers Conference
  - appointment for all the “kernel ecosystem”'s developers
    - *both invited guests as well as open registration, gathering 300 stakeholders, decision makers and developers*

References

Some more interesing pointeres

- Free Electrons – Embedded Linux Experts. http://free-electrons.com/doc/
- LinuxDevices.com. http://free-electrons.com
- AT91SAM Portal. http://www.at91.com
- Linux Kernel Newbies. http://kernelnewbies.org
- Openembedded. http://wiki.openembedded.net
- DENX Embedded Linux Development Kit. http://www.denx.de/wiki/DULG/ELDK
- CE Linux Forum. http://www.celinuxforum.org
- Embedded Linux Wiki. http://elinux.org
- Android Developers. http://developer.android.com
- Gentoo Embedded Handbook. http://www.gentoo.org/proj/en/base/embedded/handbook/
- emdebian.org. http://www.emdebian.org/
- Ångström Distribution Wiki. http://www.linuxtogo.org/gowiki/Angstrom
- OpenEmbedded User Manual. http://docs.openembedded.org/usermanual/usermanual.html
- (Unofficial) Android Porting Guide. http://www.kandroid.org/android_pdk/index.html
- Autoconf manual. http://www.gnu.org/software/autoconf/manual/
- Automake manual. http://www.gnu.org/software/automake/manual/
- Autotools Tutorial. http://www.seul.org/docs/autotut/
- Scratchbox. http://www.scratchbox.org
- Open Circuits. http://www.opencircuits.com