The Embedded Linux Quick Start Guide

In the Beginning...

Chris Simmonds

Embedded Linux Conference Europe 2010

Overview

- Genesis of a Linux project
- The four elements
  - Tool chain; boot loader; kernel; user space
- Element 1: Tool chain
- Element 2: Boot loader

“I've just had this great idea...”

- “…our next product will run Linux”

- This workshop will take a look at
  - Board bring-up
  - Development environment
  - Deployment

The four elements

- 🜁 Toolchain (air)
- 🜃 Boot loader (earth)
- 🜂 Kernel (fire)
- 🜄 User space (water)

First element: the toolchain

- You can't do anything until you can produce code for your platform
- A tool chain consists of at least
  - binutils: GNU assembler, linker, etc.
  - gcc: GNU C compiler
  - C library (libc): the interface to the operating system
  - gdb: debugger

Types of toolchain

- Native: run compiler on target board
  - If your target board is not fast enough or doesn't have enough memory or storage, use an emulator e.g. qemu
- Cross: compile on one machine, run on another
  - Most common option

The C library

- Gcc is built along side the C library
  - Hence, the C library is part of the tool chain
- Main options are
  - GNU glibc
    - big but fully functional
  - GNU eglibc
    - glibc but more configurable; embedded-friendly
  - uClibc
    - small, lacking up-to-date threads library and other POSIX functions

Criteria for selecting a toolchain

- Good support for your processor
  - e.g. for ARM A-8 core, armv4 compilers work OK but armv7t works better
- Appropriate C library
- Up-to-date
- Good support (community or commercial)
- Other goodies, e.g.
  - Cross-compiled libraries and programs
  - Development tools for tracing, profiling, etc.

Toolchain examples

Free, minimal

	URL	Architectures
Codesourcery G++ Lite	www.codesourcery.com	ARM, MIPS, PPC, SH

Free, binary

	URL	Architectures
Angstrom	www.angstrom-distribution.org	ARM, PPC, AVR32, SH
Debian	www.debian.org	ARM, PPC
Ubuntu	www.ubuntu.com	ARM
Denx ELDK	www.denx.de	PPC (ARM, MIPS)

Toolchain examples

Free, integrated build environment

	URL	Architectures
Buildroot	www.buildroot.org	ARM, PPC, MIPS
OpenEmbedded	www.openembedded.org	ARM, PPC, AVR32, SH
LTIB	www.bitshrine.org	ARM, PPC

Commercial

	URL	Architectures
MontaVista Linux	www.mvista.com	
Timesys LinuxLink	linuxlink.timesys.com	
Windriver Linux	www.windriver.com	
LynuxWorks BlueCat Linux	www.lynuxworks.com	
Sysgo ElinOS	www.sysgo.com	

“I got a toolchain with my board”

- This is often a trap!
- Most board vendors don't have in-depth embedded Linux expertise
  - Toolchain often out of date
  - Wrong libc
  - Poor selection of other development libraries
  - No update policy
- Consider using a generic toolchain instead

Installing a toolchain

- Usually everything is in a single directory tree
  - typically in `/usr/local` or `/opt`
- In which you will find...
  - cross-compiler and debugger binaries
    - cross tools have a prefix, such as **arm-angstrom-linux-gnueabi-**gcc
  - header files and libraries for the target
- To use it, do something like:

```
PATH=/usr/local/some_tool_chain/bin:$PATH
arm-angstrom-linux-gnueabi-gcc my_prog.c -o my_prog
```

Adding libraries

- A minimal tool chain only has libc
- Example: we have structured data and want to use sqlite3. What to do?
- Worst case: cross compile it yourself
  - libsqlite3 is not difficult; others are much worse
- You need
  - Header files → toolchain usr/include directory
  - Library .a and .la files → toolchain usr/lib directory
  - Library .so files → target usr/lib directory

Tip

- Choose a toolchain that comes with all (or most) of the libraries you will need for the project

Support for debugging

- For remote debugging of the target make sure your toolchain includes cross-development gdb and cross-compiled gdbserver
- Ideally it should include debug symbols in all the libraries
- Ideally it should include source code for the libraries

Other goodies

- Graphical IDE
  - Eclipse with C/C++ Development Toolkit (CDT)
- Profilers
  - Oprofile
  - Memory patrol
- Tracers
  - Linux Trace Toolkit

Second element: bootloader

- Initialise the hardware
  - Set up SDRAM controller
  - Map memory
  - Set processor mode and features
- Load a kernel
- Optional (but very useful)
  - Load images via Ethernet, serial, SD card
  - Erase and program flash memory
  - Display splash screen

Pre-boot loader

- Usually stored in flash memory
  - Old days: NOR flash mapped to processor restart vector so whole boot loader stored as single image
  - These days: first stage boot loader is stored in first page of NAND flash which is loaded by on-chip microcode
- Sequence:
  - Pre-boot loader → main boot loader → kernel

Loading the kernel

- Primary task of boot loader is to
  - Generate a description of the hardware
    - e.g. size and location of RAM, flash, ...
  - Load a kernel image into memory
  - (Optional) load a ramdisk image into memory
  - Set the kernel command line (see later)
  - Jump to kernel start vector, passing pointers to
    - information about hardware
    - kernel command line

Bootloader-kernel ABI: ATAGS

ARM (and some others) the kernel is passed values in two registers

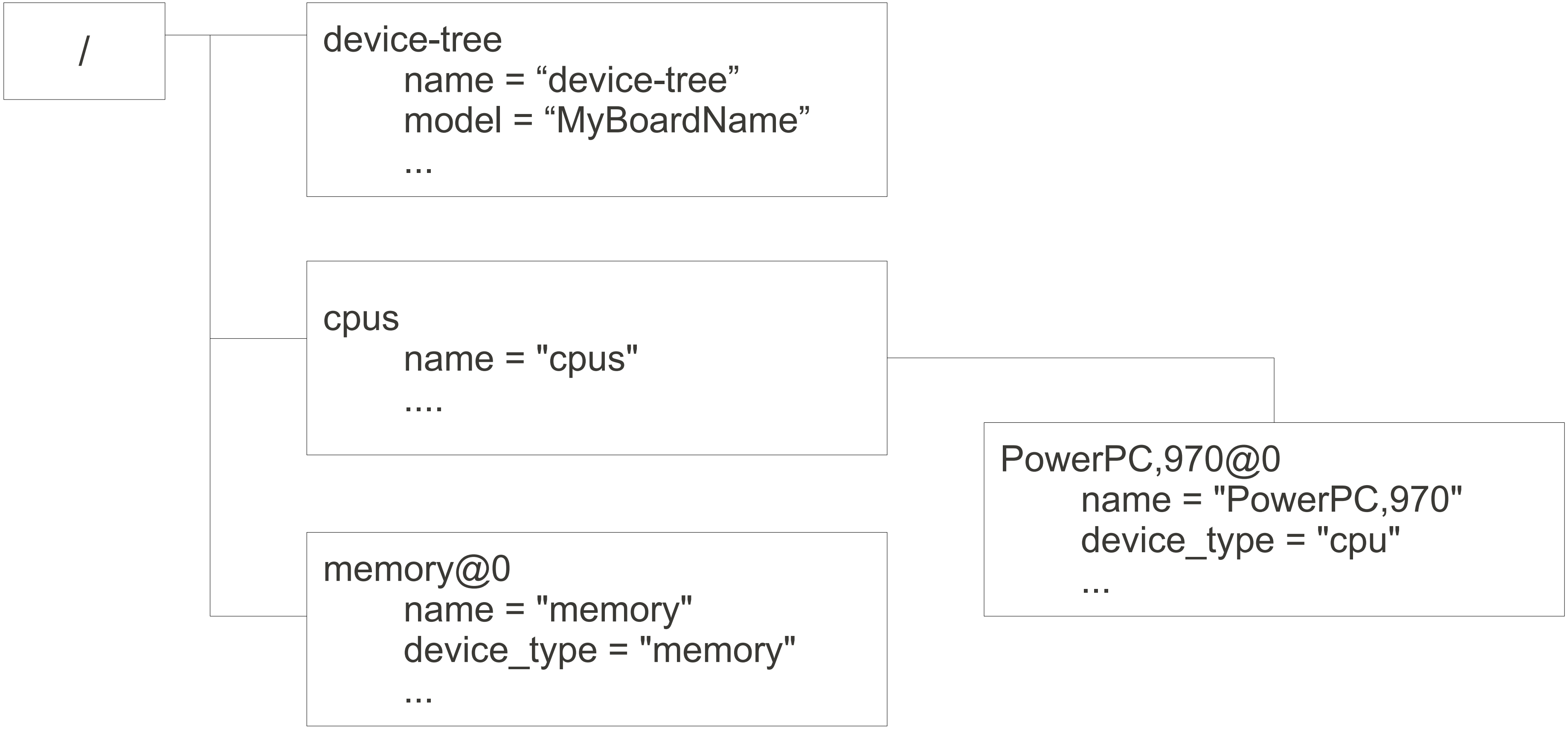
R1 = machine number
R2 = Pointer to ATAGS list

The ATAGS are a linked list of tagged values. For example

```
ATAG_CORE       ; mandatory (pagesize, rootdev)
ATAG_MEM        ; size, start physical addr
ATAG_CMDLINE    ; Kernel cmdline
ATAG_NONE       ; end of list
```

Bootloader-kernel ABI: flattened Device Tree

PPC (and others) use Flattened Device Tree (FDT)



/

device-tree
name = “device-tree”
model = “MyBoardName”
...

cpus
name = "cpus"
....

PowerPC,970@0
name = "PowerPC,970"
device_type = "cpu"
...

memory@0
name = "memory"
device_type = "memory"
...

Examples of boot loaders

- (Das) U-Boot
  - PPC, ARM, MIPS, SH4
  - http://www.denx.de/wiki/U-Boot/WebHome
- Redboot
  - PPC, ARM, MIPS, SH4
  - http://sources.redhat.com/redboot/
- For PC hardware use
  - BIOS together with GRUB or LILO

U-Boot command line

Load a kernel image into memory from...

- NAND flash

```
nand read 80100000 1000000 200000
```

- SD card

```
mmc rescan 1
fatload mmc 1:1 80100000 uimage
```

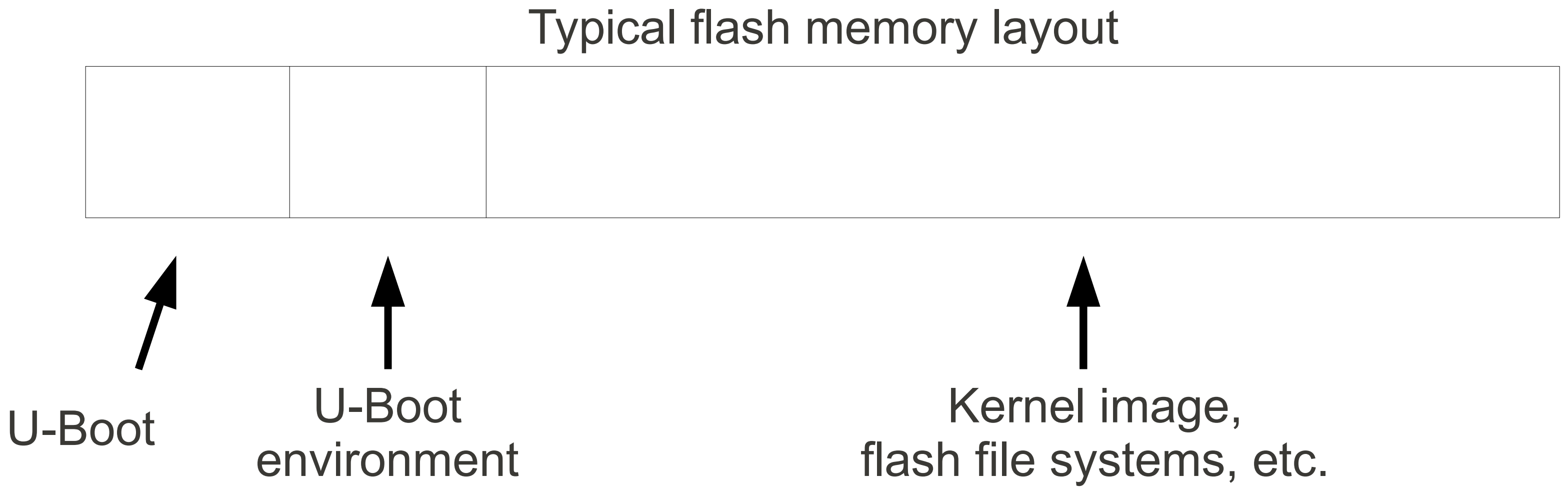
- TFTP server

```
setenv ipaddr 192.168.1.2
setenv serverip 192.168.1.1
tftp 80100000 uImage
```

Boot a kernel image in memory

```
bootm 80100000
```

U-Boot environment


Typical flash memory layout
U-Boot
U-Boot environment
Kernel image, flash file systems, etc.

U-Boot commands for environment

```
setenv ipaddr 192.168.1.101

printenv ipaddr

savvenv
```

Automating boot: bootcmd

Set command to run when U-Boot starts

```
setenv bootcmd tftp 80100000 uImage\;bootm 80100000
```

Set delay before bootcmd is execcuted

```
setelv bootdelay 3
```

Summary

- Tool chain
  - Cross or native
  - Choice of C library: glibc, eglibc or uClibc
  - Plus development libraries as needed
- Boot loader
  - Initialises the hardware and loads a kernel
  - Passes hardware description to kernel