Using the Device Tree to Describe Embedded Hardware

Grant Likely
Secret Lab Technologies Ltd.

Embedded Linux Conference
April 16, 2008

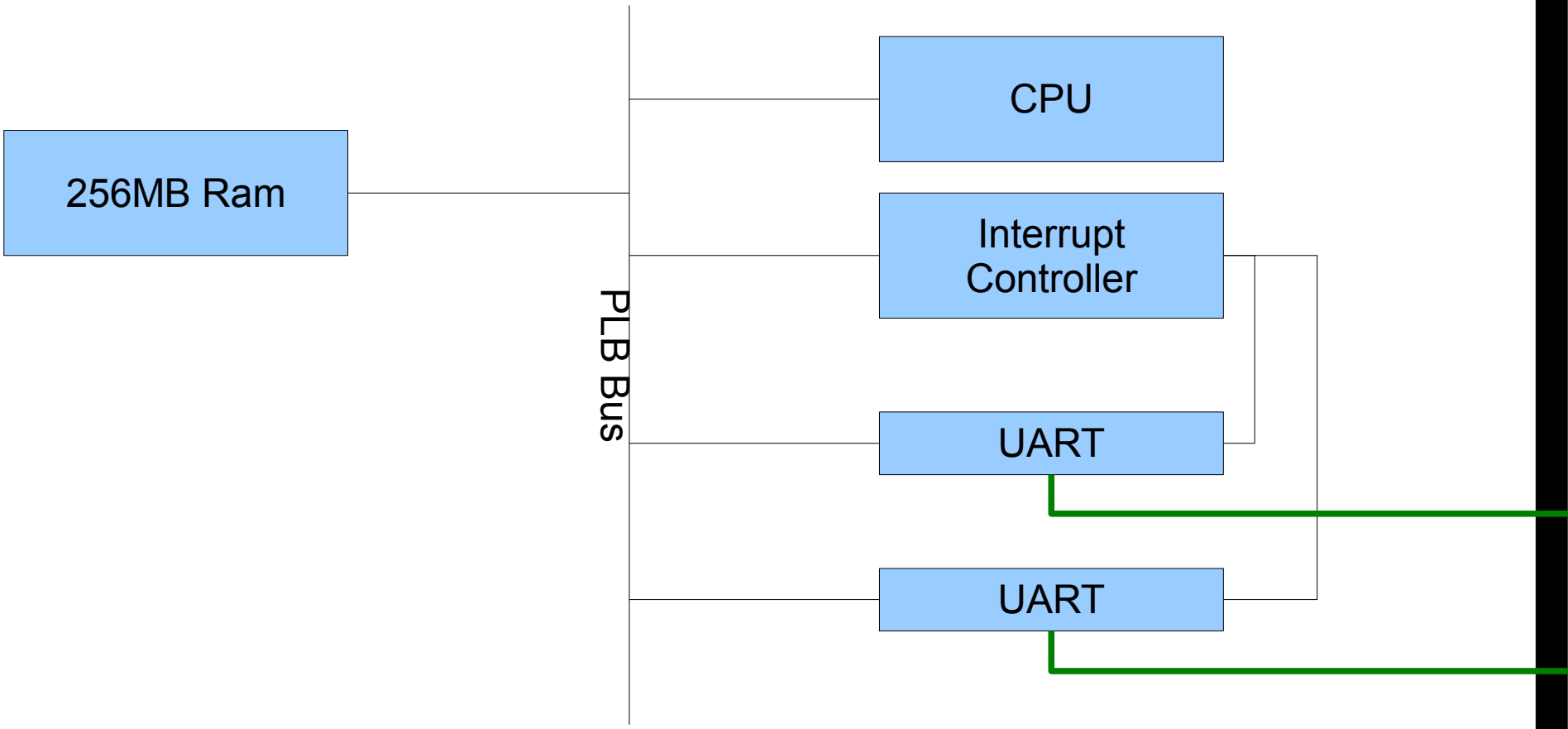
What is the Device Tree?

- Simplest definition: A Data Structure
- Nodes organized into a tree
  - Each node is named
  - Each node has exactly one parent node
  - Each node has properties containing data
- Same layout used by Open Firmware

So? What's So Special About a Tree Data Structure?

- It's not the tree; it's the data!
- Describes functional layout
  - CPUs
  - Memory
  - Peripherals
- Describes configuration
  - Console output
  - Kernel parameters
  - Device names (ttyS0, eth2, etc)

Example



- arch/powerpc requirement
- Alternative to
  - bd_info (arch/ppc)
  - mach-types (arch/arm)
- How the kernel figures out what platform it is running on

I've got a way to figure out the platform: bd_info/mach-types

- bd_info
  - Gives parameters (mem size, cpu speed) but doesn't describe platform
  - Platform determined at compile time
- Mach-types
  - Does indicate platform
  - Doesn't handle variants well
- Still doesn't tell you about attached peripherals

I've got a peripheral list too. It's a table of *struct platform_device*

- That table is hard coded in C
- Kernel has to figure out which ones are present
- No method for firmware to manipuate device setup

Why is this better?

- Define 'better'

Okay then, what are the advantages?

- Fomalize hardware description
- Multi-platform kernel images
  - Without storing config of all platforms inside kernel image
- Simplified board ports
- Less platform specific code
- Simpler device driver code

Disadvantages?

- Less compile time optimization
- Slower boot time
- More flexability == More complexity

Example

- mpc8349emitx.dts

How is it used?

- Two phases;
  - Early Boot
  - Driver initialization

How do drivers use it?

- 'compatible' property is everything
- of_platform device
- Platform device
- Other methods