Booting Linux on Embedded PowerPC™ Systems

Matt Tyrlik
Senior Software Engineer
Content

- Background
- Adding support for new platforms, the old way
- Linux arch/ppc versus arch/powerpc source code tree’s
- ePAPR (Embedded Power Architecture Platform Requirements) and Power.org
- Firmware to OS interface, Device Tree (DT)
- Content of the DT
- Passing DT information to the operating system
- The flattened DT
- The DT compiler
- Accessing information from the DT from within the operating system
- Summary/References
Background

- Firmware role
  - OS Loader
  - Hardware abstraction
  - Hardware initialization
  - **Hardware configuration**
  - Virtualization, basic platform management
  - Manufacturing instrumentation, data collection, problem determination

- Firmware usage
  - IBM PowerPC pSeries: Open Firmware
  - **Embedded PowerPC: PIBS, U-Boot, many others**
  - Apple PowerPC: Open Firmware
  - Sun SPARC: Open Firmware used in Sun Solaris and Sun Linux products
  - HP: proprietary
  - Intel: compatible BIOS, EFI, ACPI, UEFI
Background

- Problem: Varied firmware environment found in embedded platforms
  - Firmware provides the operating system with minimum configuration information
    - Ethernet MAC addresses
    - Size of RAM
    - Processor speed
  - Configuration stored in EEPROM, ELASH, other devices
  - Problems with console output during the boot process
  - Passing command line arguments to the kernel
  - Specialized boot wrappers (embed_config() function for various systems)
  - Impossible to build a single kernel image that would support multiple systems

- Goals: Standardize information passing from firmware to operating system
  - Speed up development of Linux Support Packages (LSP)
  - Minimize changes required to support new hardware
  - Enable N, N-1 OS version support
  - Enable single kernel image to support multiple systems
Adding support for new platform, the old way

- Code changes spread out among various kernel files (kernel, syslib, platforms)
- Difficult to determine what changes are required for new platform
- Number of files need to be changed
- Significant amount of custom code required for new platforms
- Custom “pseudo” device discovery code required
  - Required for complex System On Chip (SOC) devices
  - Statically defines and maps devices
- Kernel dependant on .config files (override normal driver probe logic)
- Location of devices often hard coded, PPC405GP example
  - Ppc/platofrms/4xx/ibm405gp.h
    - #define UART0_IO_BASE 0xEF600300
    - #define UART1_IO_BASE 0xEF600400
  - Ppc/syslib/ppc4xx_setup.c
    - io_block_mapping(···) call needs to be added to support new UART address
arch/ppc versus arch/powerpc

- Currently two directories (arch/ppc and arch/powerpc) support PowerPC architecture
- No new platform support is being added to arch/ppc tree
- arch/powerpc supports PowerPC 64-bit and PowerPC 32-bit targets
- arch/ppc only supports PowerPC 32-bit targets
- Use of device tree is required for booting Linux under arch/powerpc tree
  - Device tree information can be passed through Open Firmware
  - Device tree information can be provided through flattened device tree
- With real Open Firmware
  - Linux kernel calls OF to scan the device tree
  - Linux transfers information to internal representation that is then used at run-time
ePAPR and Power.org

- Power.org (www.power.org)
  - Power.org's mission is to develop, enable and promote Power Architecture technology
  - Supports a number of standard and specification initiatives
- sPAPR (Server Power Architecture Platform Requirements) available now
  - To create a stable platform architecture to be used by server platforms based on Power processors
  - To create an architecture which will allow platforms to operate with previous versions of the OS
  - To minimize the support cost for multiple OS versions through the definition of common platform abstraction techniques.
  - sPAPR provides complete server platform definition that is not fully applicable to embedded systems
- ePAPR (Embedded Power Architecture Platform Requirements) available 4Q2007
  - Main effort in standardization of firmware to OS interface
ePAPR and Power.org (cont)

- ePAPR 1.0 addresses only boot services
  - How firmware initializes hardware and boots an OS
  - How configuration information is passed to OS, Device Tree (DT)
  - Multi core considerations
  - 32-bit and 64-bit considerations
- ePAPR is loosely related to IEEE 1275 (Open Firmware)
  - Takes device tree structure
- Draws heavily on on-going work being done in the PowerPC Linux community
- In the future ePAPR will address issues dealing with
  - Optional. Runtime Abstraction Services (RTAS)
    - Firmware component resident at runtime
    - Examples: time of day, reboot, power-off, memory allocation
  - Optional, Virtualization
    - OS to Hypervisor API
Firmware to OS interface, Device Tree

- Device tree provides a way to:
  - Represent hardware configuration in hierarchical way (each node except root has a parent)
  - Pass information from firmware to OS
- Device tree is made up of “nodes”

```
/  
+--- cpus
    |    
    |    +--- PowerPC,750CL@0
    |    |    
    |    +--- PowerPC,750CL@1
    |          
    +--- soc
          
          +--- ethernet
          |          
          |          +--- serial
```
Firmware to OS interface, Device Tree (cont)

• Nodes represent devices and busses (there are few special cases)
  • /chosen node represents information passed from Firmware to OS
• /memreserve structure represents memory used by Firmware and DT itself
• Nodes have properties made out of name value pairs
  • Node values can be: numeric, strings, list of strings, tables, other structured information

```plaintext
soc
  {device_type = "soc";
   #address-cells = <1>;
   #size-cells = <1>;
   ranges = <0 e0000000 00100000>;
   reg = <e0000000 00000200>;
  }

serial
  {device_type = "serial"
   compatible = "ns16550"
   reg = <4600 100>
   clock-frequency = < 151E40 >
   interrupts = <a 8>
   interrupt-parent = < &ipic >
  }
```
Firmware to OS interface, Device Tree (cont)

- Devices that can be discovered dynamically generally don’t have to be included
  - PCI device discovery is well standardized and PCI devices don’t have to be included
  - USB devices generally don’t have to be included since they can be easily enumerated
- PCI host bridges generally have to be included
- Devices with atypical interrupt routing should be included
- Nodes representing devices and buses must have “compatible” property
  - Example: compatible = "ibm,uic-440gp", "ibm,uic", compatible = "ns16550"
  - Compatible property is used for device driver matching in the kernel
  - Compatible property with in turn specifies properties that will be used to describe the node
    - Example for serial device: clock-frequency = <3F9C6000>, current-speed = <1c200>;
- Node name is made out of “unit name” and optional “unit address”
  - Example: serial@7c08, serial@6600
  - Unit addresses are used to differentiate multiple devices at the same level of hierarchy
- Node can be referred to from within the DT by using full node path names*

*: phandle can also be used
Firmware to OS interface, Device Tree (cont)

- Property values, as interpreted by client software (Linux kernel)
  - Empty, example: interrupt-controller.
  - u32, example: #size-cells = <6000>
  - string, example: bootargs = “console=ttyS0,115200”
  - prop-encoded-array, example: interrupt-map = <0800 0 0 1 &RT0 24 0 0800 0 0 2 &RT…>
  - phandle, example: phandle = < 12340001 >
  - stringlist, example: compatible = "ns16550", “i8250”

- Some standard properties (in addition to compatible). Names are case sensitive.
  - reg: physical address and length of devices memory mapped register space
  - ranges: for bus node describes bus address mapping
  - device_type: defines device programming model
  - model: manufacturers model
  - phandle: unique numerical identifier for the node (Linux uses: linux,phandle) *
  - #address-calls and #size-cells: describe how the child devices should be addressed

*: flatten device tree property
Content of the Device Tree (cont)

• **cpu node**
  • Describes CPUs or cores in the system
  • Standard properties include: reg, clock-frequency, reservation-granule-size, etc
  • TLB, L1 cache, as well as multi level and shared caches can be described

• **memory node**
  • Required for all DT
  • Describes physical memory layout of the system
  • Only read/write memory should be described using memory node

• **Number of other device nodes**
  • See Documentation/powerpc/booting-without-of.txt file in Linux source code
  • Various bindings to “IEEE 1275” standard
  • ePAPR on Power.org WEB site, when available
Content of the Device Tree (cont)

- Example (partial) DT for PPC750CL system

```c
/ {
    model = "41K7339";
    compatible = "ibm,holly";
    #address-cells = <1>;
    #size-cells = <1>;
    cpus {
        #address-cells = <1>;
        #size-cells = <0>;
        PowerPC.750CL@0 {
            device_type = "cpu";
            reg = <0>;
            d-cache-line-size = <20>;
            i-cache-line-size = <20>;
            d-cache-size = <8000>;
            i-cache-size = <8000>;
            d-cache-sets = <80>;
            i-cache-sets = <80>;
            timebase-frequency = <2faf080>;
            clock-frequency = <23c34600>;
            bus-frequency = <bebc200>;
            32-bit;
        }
    }
    memory@0 {
        device_type = "memory";
        reg = <00000000 20000000>;
    }
    ...
}

tsi109@c0000000 {
    device_type = "tsi-bridge";
    compatible = "tsi-bridge";
    #address-cells = <1>;
    #size-cells = <1>;
    ranges = <00000000 c0000000 00010000>;
    reg = <c0000000 00010000>;
    ethernet@6200 {
        device_type = "network";
        compatible = "tsi-ethernet";
        #address-cells = <1>;
        #size-cells = <0>;
        reg = <6000 200>;
        local-mac-address = [ 00 00 00 00 00 00 ];
        interrupt-parent = <
        &/tsi109@c0000000/pic@7400 >;
        interrupts = <10 2>;
        phy-handle = <&PHY1>;
    }
    MPIC: pic@7400 {
        device_type = "open-pic";
        compatible = "chrp,open-pic";
        interrupt-controller;
        #interrupt-cells = <2>;
        reg = <7400 400>;
        big-endian;
    }
    ...
}
```
Passing DT Information to the OS

- Device Tree information needs to be passed to the OS (or bootstrap) code
  - OS receives flattened (binary) representation of the Device Tree from bootstrap code (zImage)
- Through Open Firmware
  - Open Firmware constructs Device Tree on the fly
  - Bootstrap code calls Open Firmware and constructs binary representation of the DT
- Binary representation of the device tree passed directly to bootstrap code
  - Non “Open Firmware” firmware
  - Firmware contains largely static binary representation of the device tree
  - Firmware updates several values in the device tree (ex. network hardware address)
- Binary representation of the device tree compiled into bootstrap code
  - Used for boards with legacy firmware
  - Enables support for legacy platforms under arch/powerpc tree
Flattened Device Tree

- Flattened DT represents DT in a compact binary format
  - Relocatable. Can be moved without knowledge of the internals (no pointers)
  - Permits easy insert/delete/update operations (limits use of internal offsets)
  - Compact. Use of common string block.
  - Easily parsed by software

- Flattened DT made out of 4 sections
  - Header. Provides offsets to other sections and other basic information (boot CPU ID)
  - Memory reserve table (information contained in /memreserve node)
  - String block. All the ASCII string representing property names are contained in this section
  - Structure block. Contains structured data representing the DT

```
+ off_struct
  + 0x00 0x04
  + 0x08 OF_DT_BEGIN_NODE
  + 0x0C 0x00000005
  + 0x10 0x00000008
  + 0x14 'M' 'y' 'B' 'o'
  + 0x18 'a' 'r' 'd' 0
  + ...
  + OF_DT_PROP
  + OF_DT_END_NODE
  + OF_DT_END
```

Example from “Device trees everywhere” paper
Device Tree Compiler

- Constructing binary representation of the DT by hand is error prone
- Device Tree Compiler (DTC) converts various DT representation
  - Input format
    - Text format (see page 14), binary representation (see page 16)
    - Filesystem (format in the /proc/device-tree filesystem)
  - Output format: binary representation, text format, assembler source
- DT checking: syntactic structure, semantic structure (moving away from semantic checking), Linux requirements
- Support for references, ex: interrupt-parent = < &/tsi109@C0000000/pic@7400 >
- Support for labels, ex: MPIC: pic@7400
- Source code for the current version of the DTC can be accessed by executing
  - git clone git://www.jdl.com/software/dtc.git
- Sample device tree files can be found under arch/powerpc/boot/dts
- DTC package contains library (libfdt) of functions that manipulate binary DT
Accessing DT Information within OS

- Bootstrap code uses DT
  - Serial port type, location
  - Host bridge setup
  - finddevice(), getprop() calls used to retrieve information from the DT
- The pointer to DT binary representation is passed in r3 to the Linux kernel
  - Rich set of API calls to retrieve information from the DT (arch/powerpc/kernel/prom.c)
  - Some API calls also included in arch/powerpc/kernel/prom_parse.c
  - of_find_compatible_node(), find node based in token in “compatible” property
  - of_get_property(), find property given node and property name
  - of_find_node_by_type(), searched for node given “device_type” property
  - of_translate_address(), translates address from DT to CPU physical address:
- At run time the DT information can be accessed through /proc/device-tree FS
Summary/References

Summary

• Use of DT can minimize changes required to support new hardware
• Speed up development of Linux Support Packages (LSP) through improved code reuse
• Enable single kernel image to support multiple systems

References

• “Booting the Linux/ppc kernel without Open Firmware”, Documentation/powerpc/booting-without-of.txt in Linux source code