Device tree and embedded Linux

Vitaly Bordug
Principal Engineer
What is the device tree?

• Just a data structure representing:
  – a tree, layered out system of nodes;
  – only one parent allowed.

• Each node has following properties:
  – each node has a name;
  – node contains actual data, that is stored in a list of «properties».

• Source and binary of the device tree
Origin of the device tree

• Inspired from OpenFirmware (OF)
• Addresses problems to determine HW configuration
  – common stream: desktop, server and BIOS;
  – embedded platforms specifics.

• Why it was needed
  – devtree is clear, flexible and is a standard;
  – was a due for ppc/powerpc merge.
From theory to implementation

• But we were doing good without that stuff... How?
  – bd_t (already history). Only parameters, no real description.
  – ARM mach_types. Indicates platforms, but not enough flexibility to handle variants.

• Separating structure and code: the DTS (device tree source) way
Pros and Cons

- Formal and clear HW description
- Multiplatform kernels now possible
- Less board-specific code, more efficient device-driver binding

- Bigger kernel (in terms of footprint and overall size)
- Slower boot time
- Complex layers to enable devtree on new architectures
OF without real OF

- PPC32, u-boot and OF: first steps and questions
  - where to place the dtb (device tree binary);
  - Support older FW versions/implementations
    - Add functionality
    - Maintain backward compatibility
- Current state: how to do it right
  - DTC (device tree compiler) dependency removed
  - DTS (device tree source) files for all supported boards are maintained within kernel source
  - U-boot mainline (from v1.1.3) supporting device tree natively, with backward compatibility
What devicetree source looks like

```devicetree
memory {
    device_type = "memory";
    reg = <0x00000000 0x04000000>;
};

soc832e @00000000 {
    #address-cells = <1>;
    #size-cells = <1>;
    device_type = "soc";
    compatible = "simple.bus";
    ranges = <0x0 0x00000000 0x00100000>;
    reg = <0x00000000 0x00000200>;
    bus-frequency = <0>;

    wdtc200 {
        device_type = "watchdog";
        compatible = "mpc83xx_wdt";
        reg = <0x200 0x100>;
    };

    i2c@0000 {
        address-cells = <1>;
        #size-cells = <0>;
        cell-index = <0>;
        compatible = "fsl-i2c";
        reg = <0x3000 0x100>;
        interrupts = <14 0x0>;
        interrupt-parent = <gipic>;
        dvfsr;
    };

    serial0: serial@4500 {
        cell-index = <0>;
        device_type = "serial";
        compatible = "ns16550";
        reg = <0x4500 0x100>;
        clock-frequency = <0>;
    };
}
```
Implementation issues and activities to mitigate

• Devicetree OF specification does not provide a clear distinction between configuration options and h/w capabilities
  – Documentation revamp underway
  – New mailing list — first place to ask.. Not only when you're not sure. (devicetree-discuss@ozlabs.org)

• Multicore: model needs clear way for hypervisor to distribute resources between cores
  – Include devicetree source
  – Current workaround: devicetree merge
What about other architectures?

- Actively considered as an alternative for ARM mach-* mess
- We already have something to show...
[.......OMAP5912osk.......]
- What is still to do though:
  - Support for the U-Boot (take dtb and pass it over to the kernel
  - Kernel-side dtb support
  - OF-like interrupt controllers support (get rid of static mapping. Plenty of work :) )
DTS applications: beyond kernel

• Stepping outside initial goals and definitions
  – uImage and its limitations
  – Use devicetree as a container to construct new uImage

• New uImage is already in mainline — what does it mean in terms of support for existing products
  – Full backward-compatibility
  – Bunch of flexibility and functionality if it is needed
New uImage: how the whole thing works

- Image source file
- Image data files
- mkimage
- DTC
- new composite image file
- TARGET
Image tree source example

```c
images {
    kernel@1 {
        description = "Vanilla Linux kernel";
        data = /incbin("./vmlinux.bin.gz");
        type = "kernel";
        arch = "ppc";
        os = "linux";
        compression = "gzip";
        load = <00000000>;
        entry = <00000000>;
        hash@1 {
            algo = "crc32";
        };
        hash@2 {
            algo = "sha1"
        }
    };
    fdt@1 {
        description = "Flattened Device Tree blob";
        data = /incbin("./target.dtb");
        type = "flat_dt";
        arch = "ppc";
        compression = "none";
        hash@1 {
            algo = "crc32";
        };
        hash@2 {
            algo = "sha1"
        }
    };
};
```
Leveraging the New uImage Implementation

• Maximizes flexibility in kernel and RFS combinations:
  – single and multi-kernel are supported;
  – allows for support of a single “multiplatform image” with different DTBs.

• Not restricted to the kernel:
  – image tree source can store additional user-defined data - extremely useful to store configurations;
  – auto-update extended firmware feature was merged to the mainline u-boot see doc/README.update