SCALE 13x

Contemporary Device Tree

Matt Porter
Konsulko
mporter@konsulko.com

Overview

- History of Device Tree (DT)
- Linux kernel use of DT
- Device Tree Basics
- Device Tree Examples
- Modifying Device Trees
- Dynamic Device Tree Overlays

History of Device Tree (DT)

- Sun Microsystems - Open Boot / Open Firmware (1988)
  - Used in SPARC systems
  - Uses DT to describe hardware
- IEEE-1275 formalized specification (1994)
  - Documents DT
- Apple adopts Open Firmware on Power Mac 7200 (1995) using DT
- Common Hardware Reference Platform (CHRP) specifies DT (1995)
- ePAPR specifies DT (2008)

DT in the Linux kernel

- SPARC
  - All systems pass DT so it's supported for a long time in the kernel
- PowerPC
  - PowerMacs also drove support of DT in the kernel
  - Major reorg in 2005 merging 32-bit/64-bit required all platforms to use DT
- x86
  - Yes, really. CE4100 Falconfalls uses DT

DT in the Linux kernel

- ARM
  - Linus' ultimatum to ARM community (2011)
    - http://lists.infradead.org/pipermail/linux-arm-kernel/2011-April/048543.html
  - All new platforms must use DT to describe hardware
- MIPS
  - Fall out from Linus' ultimatum, Ralf Baechle begins MIPS conversion to DT
    - http://www.linux-mips.org/archives/linux-mips/2011-06/msg00059.html
- Microblaze
  - Using DT in 2008, driven by flexible FPGA I/O

Defining Device Tree

- ePAPR defines DT
    - https://www.power.org/documentation/epapr-version-1-1/
        - ***a concept called a device tree to describe system hardware.***
        - ***A device tree is a tree data structure with nodes that decribe the physical devices in a system.***
        - ***[a] device tree describes device information in the system that cannot be dynamically detected by a client.***

Device Tree Basics

- Nodes
  - Groupings of properties and child nodes
- Properties
  - Key-Value Pairs
    - Text strings “my string”
    - Cells (32-bit) <0xdeadbeef 11 0xf00d>
    - Binary data [0x01 0x02 0x03 0x04]
    - mixed data, concatenate with a comma
      - “my string”, <0xdeadbeef>, [0x04], “your string”
- Phandles
  - Reference to another node

Device Tree Source Format

Node name and unit address

String, list, and binary property values

```
/ {
    node1@0 {
        a-string-property = "A string";
        a-string-list-property = "first string", "second string";
        a-byte-data-property = [0x01 0x23 0x34 0x56];
        child-node1@0 {
            first-child-property;
            second-child-property = <1>;
            third-child-property = <&node2>;
            a-string-property = "Hello, world";
        };
        child-node2@1 {
        };
```

Phandle

Label

```
    node2: node2@1 {
        an-empty-property;
        a-cell-property = <1 2 3 4>; /* each number (cell) is a uint32 */
        child-node1 {
        };
    };
};
```

Device Tree Node Example

Unit address is the memory mapped I/O address of this UART

```
uart0: serial@44e09000 {
```

An identifier for this hardware which allows the operating system to match a compatible driver to the peripheral.

```
          compatible = "ti,omap3-uart";
```

Vendor specific properties are prefixed with a vendor ID

```
          ti,hwmods = "uart1";
```

Input clock frequency for this UART

```
          clock-frequency = <48000000>;
```

Base address of length of the memory mapped I/O

```
          reg = <0x44e09000 0x2000>;
```

Interrupt number

```
          interrupts = <72>;
```

Device is not enabled

```
          status = "disabled";
```

DMA engine properties, phandle to DMA deivice node and channels used

```
          dmas = <&edma 26>, <&edma 27>;
          dma-names = "tx", "rx";
};
```

arch/arm/boot/dts/am33xx.dtsi

Device Tree Driver Example

Compatible strings this driver matches

```
#if defined(CONFIG_OF)
static const struct of_device_id omap_serial_of_match[] = {
        { .compatible = "ti,omap2-uart" },
        { .compatible = "ti,omap3-uart" },
        { .compatible = "ti,omap4-uart" },
        {},
};
MODULE_DEVICE_TABLE(of, omap_serial_of_match);
#endif
```

Probe will be run when a matching compatible string is found

```
static struct platform_driver serial_omap_driver = {
        .probe          = serial_omap_probe,
        .remove         = serial_omap_remove,
        .driver         = {
                .name   = DRIVER_NAME,
                .pm     = &serial_omap_dev_pm_ops,
                .of_match_table = of_match_ptr(omap_serial_of_match),
        },
};
```

drivers/tty/serial/omap-serial.c

Device Tree Platform Example

```
&am33xx_pinmux {
...
```

Child node contains platform specific pin mux settings for UART0

```
       uart0_pins: pinmux_uart0_pins {
               pinctrl-single,pins = <
                       0x170 (PIN_INPUT_PULLUP | MUX_MODE0)
                       0x174 (PIN_OUTPUT_PULLDOWN | MUX_MODE0)
               >;
        };
…
};


...
```

Pinctrl properties contain pin mux settings that are referenced here via phandle

```
&uart0 {
        pinctrl-names = "default";
        pinctrl-0 = <&uart0_pins>;
```

Marking this device enabled instantiates the device in the kernel

```
        status = "okay";
};
```

arch/arm/boot/dts/am335x-bone-common.dtsi

Device Tree Bindings

- ePAPR defines as
  - ***...requirements, known as bindings, for how specific types and classes of devices are represented in the device tree.***
- Maintained in Linux kernel
  - Documentation/device-tree/bindings/*
  - Follow standards and conventions established in IEEE1275 and ePAPR
- Each type and class of device has a DT binding describing all of the valid properties and their behavior

Device Tree Binding Example

```
OMAP UART controller

Required properties:
- compatible : should be "ti,omap2-uart" for OMAP2 controllers
- compatible : should be "ti,omap3-uart" for OMAP3 controllers
- compatible : should be "ti,omap4-uart" for OMAP4 controllers
- ti,hwmods : Must be "uart<n>", n being the instance number (1-based)

Optional properties:
- clock-frequency : frequency of the clock input to the UART
```

Whoa, wait, this is a terrible example. This binding doesn’t show the reg, interrupt, or dmas properties that are clearly used in the working dtsi source file’s UART node.

Documentation/devicetree/bindings/serial/omap_serial.txt

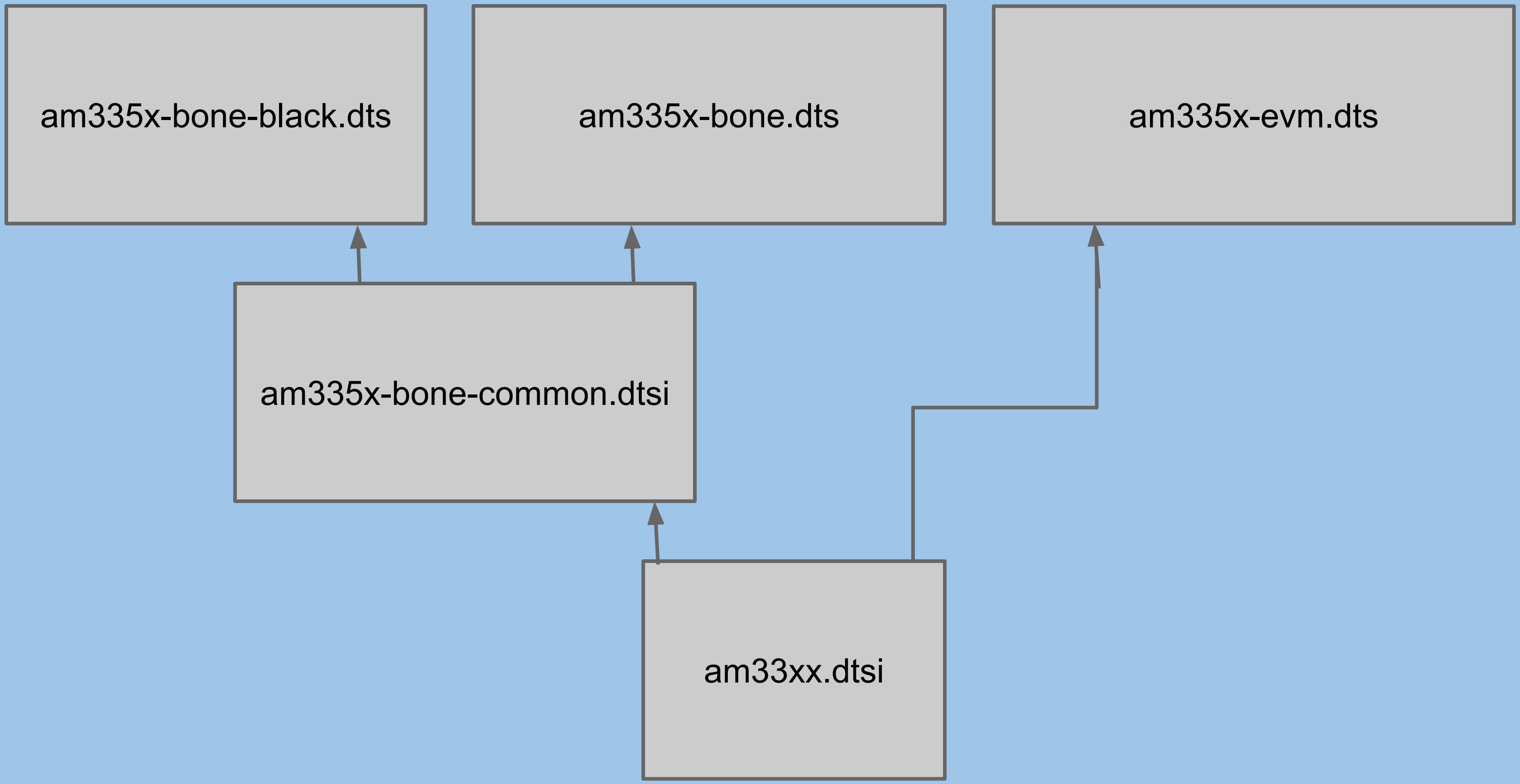
FIXED Device Tree Binding Example

```
OMAP UART controller

Required properties:
- compatible : should be "ti,omap2-uart" for OMAP2 controllers
- compatible : should be "ti,omap3-uart" for OMAP3 controllers
- compatible : should be "ti,omap4-uart" for OMAP4 controllers
- reg : address and length of the register space
- interrupts: Should contain the uart interrupt number
- ti,hwmods : Must be "uart<n>", n being the instance number (1-based)

Optional properties:
- clock-frequency : frequency of the clock input to the UART
- dmas: DMA specifier, consisting of a phandle to the DMA controller
        node and a DMA channel number.
- dma-names: “rx” for RX channel, “tx” for TX channel.


Example:
uart0: serial@44e09000 {
          compatible = "ti,omap3-uart";
          ti,hwmods = "uart1";
…
};
```

Managing Device Tree Source (DTS)

- DTS files found in arch/foo/boot/dts/ on ARM, MIPS, PowerPC, and MicroBlaze
- A DTS file is compiled into a Device Tree Blob (DTB)
  - The resulting DTB is passed to the kernel at boot
  - All devices are created using the contents of the DTB.
- DTS files may include other files

BeagleBone White/Black DTS/DTSI



Compiling DTBs

- The Device Tree Compiler (dtc) is used to compile DTS -> DTB
    - Found in ***scripts/dtc***
- Invoking dtc from the kernel tree
    - ***make dtbs***
    - will generate DTBs for each DTS in ***arch/foo/boot/dts/****
    - Runs C preprocessor on source files
- Invoking dtc directly
    - ***dtc -I dts -O dtb -o myboard.dtb myboard.dts***

Adding an I2C device to a board

Add this i2c1 pins node which defines pinmux settings for BeagleBlack P9-18 and P9-17 to be muxed as I2C1_SDA and I2C1_SCL, respectively. The property values are gathered from a combination of the board manual, SoC datasheet, and the pinctrl single binding.

```
&am335x_pinmux {
...
       i2c1_pins: pinmux_i2c1_pins {
               pinctrl-single,pins = <
                       0x158 (PIN_INPUT_PULLUP | MUX_MODE2)
                       0x15c (PIN_INPUT_PULLUP | MUX_MODE2)
               >;
        };
...
};
```

The i2c1 device node included from the am33xx.dtsi is overlayed with board specific properties. The pinctrl properties reference the above pinmux configuration for the I2C pins and the clock frequency is set according to the device capabilities on the I2C bus. Finally, the i2c controller is enabled. Note that the i2c1 device node defaults to disabled in am33xx.dtsi.

```
…
&i2c1 {
        pinctrl-names = "default";
        pinctrl-0 = <&i2c1_pins>;
        clock-frequency = <400000>;
        status = "okay";

        at24@50 {
                compatible = "at,24c256";
                pagesize = <64>;
                reg = <0x50>;
        };
};
```

The at24@50 child node instantiates a 24c256 I2C serial eeprom on the parent I2C bus at address 0x50.

Modifying arch/arm/boot/dts/am335x-boneblack.dts

Dynamic Device Trees

- Power architecture has had CONFIG_OF_DYNAMIC
  - Destructive changes to the live device tree
- Capebus introduced in 2012, a kernel framework to dynamically modify the DT based on pluggable hardware
  - Driven by Beaglebone’s standardized expansion capes (Capebus shipped in the Beaglebone 3.8 kernel) .
- Capebus discussions made way upstream for the staged introduction of dynamic DT overlays.

Dynamic DT Overlays

- Part of the kernel since 3.19
- Introduces the concept of a DT fragment
  - A DT fragment is a DTB which contains symbols that can only be resolved at runtime against the live tree.
- Allows a user to insert a DT fragment into the live tree at runtime and activate it
  - The inserted fragment becomes part of the tree in the same manner as if it were compiled into the DTB and passed at boot time.
- A configfs interface is used to specify DT fragments to be applied to the live tree

DT Fragment Format

- Use overlay enabled dtc from https://github.com/pantoniou/dtc
- Borrowing Pantelis Antoniou's example:

```
/* qux.dts */
/dts-v1/;
```

Notifies dtc that this is a DT fragment.

```
/plugin/;

/ {
```

The dtc compiler will assign a phandle value of 0x00000001 to the baz node.

```
        BAZ: baz { };
```

The dtc compiler will resolve &BAZ accordingly for a value of <0x00000001>.

```
        qux = <&BAZ>;
```

The dtc compiler will mark property quux as requiring a fixup at runtime to resolve &FOO.

```
        quux = <&FOO>;
};
```

DT Fragment Format Analysis

- Compile it

```
$ dtc -O dtb -o qux.dtbo -b 0 -@ qux.dts
```

- Dump it

```
$ fdtdump qux.dtbo
```

```
/ {
qux = <0x00000001>;
quux = <0xdeadbeef>;
baz {
linux,phandle = <0x00000001>;
phandle = <0x00000001>;
};
__symbols__ { BAZ = "/baz"; };
__fixups__ { FOO = "/:quux:0"; };
__local_fixups__ { qux = <0>; };
};
```

Add I2C device: DT Overlay Version

```
/dts-v1/;
/plugin/;

/ {
    fragment@0 {
        target = <&am3353x_pinmux>;
        __overlay__ {
           i2c1_pins: pinmux_i2c1_pins {
                    pinctrl-single,pins = <
                            0x158 0x72
                            0x15c 0x72
                    >;
           };
        };
    };

    fragment@1 {
        target = <&i2c1>;
        __overlay__ {
            pinctrl-names = "default";
            pinctrl-0 = <&i2c1_pins>;
            clock-frequency = <400000>;
            status = "okay";

            at24@50 {
                    compatible = "at,24c256";
                    pagesize = <64>;
                    reg = <0x50>;
            };
        };
    };
};
```

at24-i2c1.dts

Add I2C device: Build and Load Overlay

```
$ dtc -O dtb -o at24-i2c1.dtbo -b 0 -@ at24-i2c1.dts
$ mkdir /config/device-tree/overlays/a24-i2c1
```

Loads and activates the DT fragment, enabling the I2C1 controller and instantiating the 24C256 I2C EEPROM

```
$ cp at24-i2c1.dtbo /config/device-tree/overlays/at24-i2c1/dtbo
```

Deactivates and removes the DT fragment, disabling the I2C1 controller and the 24C256 I2C EEPROM

```
$ rmdir /config/device-tree/overlays/at24-i2c1
```