Engaging Device Trees

Embedded Linux Conference 2014

Geert Uytterhoeven
`geert@linux-m68k.org`

Glider bvba

Tuesday, April 29

About Me (and Linux)

Hobbyist

- 1994 Linux/m68k on Amiga
- 1997 Linux/PPC on CHRP
- 1997 FBDev

Sony

- 2006 Linux on PS3/Cell at Sony

Glider bvba

- 2013 Renesas ARM-based SoCs

Introduction

- Where are Device Trees coming from?
- What problems do Device Trees solve?
- What challenges do Device Trees pose?
- Best Practices to improve bindings between IP cores in SoCs, devices on boards, and drivers.
- Make it easier to support faster a vast variety of SoCs, boards, and peripherals, also for production (LTSI) kernels.

History of Device Trees in Linux

Open Firmware

1991 Sun OpenBoot V2.x, SPARCstation 2

1994 IEEE 1275-1994

1997 My first experience with DT on PPC:
Real Open Firmware on CHRP LongTrail

- Forth
- Used on Apple PowerMac and IBM machines
- PCI devices represented in DT, generated by firmware
- Nodes for ISAPnP under `pci/isa/`
- DT not much used by Linux yet

History of Device Trees in Linux

Flattened Device Tree

- 2005 PPC starts switching to FDT
- 2006 First in-kernel DTS: `mpc8641_hpcn.dts`
- 2007 PS3: mandatory, but rudimentary DTS:
  - Dummy memory
  - 1 CPU with 2 threads
  - CPU cache
  - Dummy clock frequencies

```
/dts-v1/;
/ {
        model = "SonyPS3";
        compatible = "sony,ps3";
        #size-cells = <2>;
        #address-cells = <2>;
        chosen { };
        memory {
                device_type = "memory";
                reg = <0x00000000 0x00000000 0x00000000 0x00000000>;
        };
        cpus {
                #size-cells = <0>;
                #address-cells = <1>;
                cpu@0 {
                        device_type = "cpu";
                        reg = <0x00000000>;
                        ibm,ppc-interrupt-server#s = <0x0 0x1>;
                        clock-frequency = <0>;
                        timebase-frequency = <0>;
                        i-cache-size = <32768>;
                        d-cache-size = <32768>;
                        i-cache-line-size = <128>;
                        d-cache-line-size = <128>;
                };
        };
};
```

History of Device Trees in Linux

World Domination

- **2007** Common implementation for PPC and SPARC
  `drivers/of`
- **2009** New Linux architectures/platforms use DT:
  microblaze
- **2011** ARM switches to DT
- **2014** DT used by 12 out of 28 architectures:
  ppc, sparc, microblaze, mips, x86, arm, openrisc,
  c6x, arm64, metag, xtensa, arc
  (+ nios2)

What Are Device Trees?

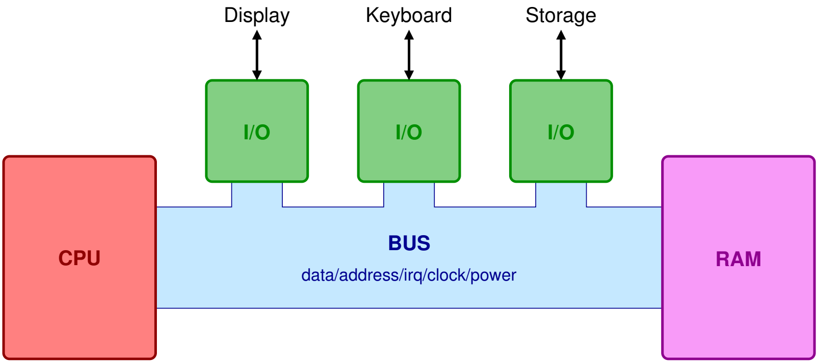
What?

- Description of the hardware
- Relationships between various hardware components
- OS-agnostic

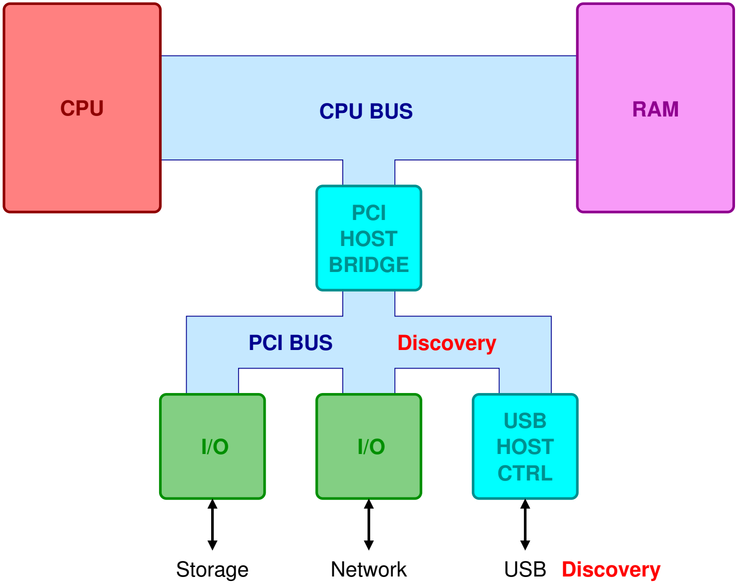
Why?

- Why do we need it?
- What problems does it solve?
- Other solutions?

Simple Computer



- Simple bus
- Expansion cards?


CPU
CPU BUS
RAM
PCI
HOST
BRIDGE
PCI BUS
Discovery
I/O
I/O
USB
HOST
CTRL
Storage
Network
USB
Discovery

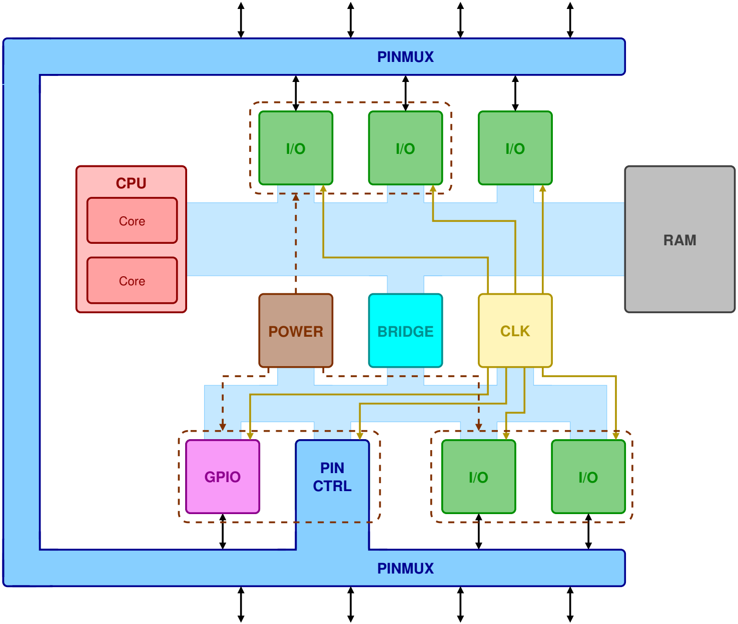
End of 20th Century

State of hardware

- Mostly completed moving from hardwired logic blocks to discoverable buses like PCI, USB, ...
- IsaPNP

State of Linux

- No device framework, no platform devices
- Mostly single-platform kernels
  (excl. m68k, PowerPC, ...)
- PCI discovery
- Still some ISA probing
  non-x86: don’t compile in the driver to avoid crashes
- Live CDs, e.g. Knoppix


PINMUX
I/O
I/O
I/O
CPU
Core
Core
RAM
POWER
BRIDGE
CLK
GPIO
PIN
CTRL
I/O
I/O
PINMUX

SoC + Board

Embedded Device

- SoC + board peripherals

Return of the Non-Discoverable Buses

- Lots of hardwired logic on-chip
- Peripherals on simple buses: spi, i2c, i2s, 1-wire, SDIO, ...
- Buses behind other buses
- Power regulators and power domains
- Clock generators and clock domains
- Multiple interrupt controllers
- Pinctrl and pinmux
- Complex topologies and dependencies
- Buses with support for discovery for expansion

Linux kernel needs to know which hardware it's running on

Need good description of the hardware

1. (A)TAGS: m68k, ARM
   ABI boot loader / kernel
2. Board code with platform devices
   Code, complex, boring
3. DT
   Better separation of code and data
4. ACPI
   Hmmm . . .

Why Device Trees?

Multi-Platform

Single-Platform Kernels

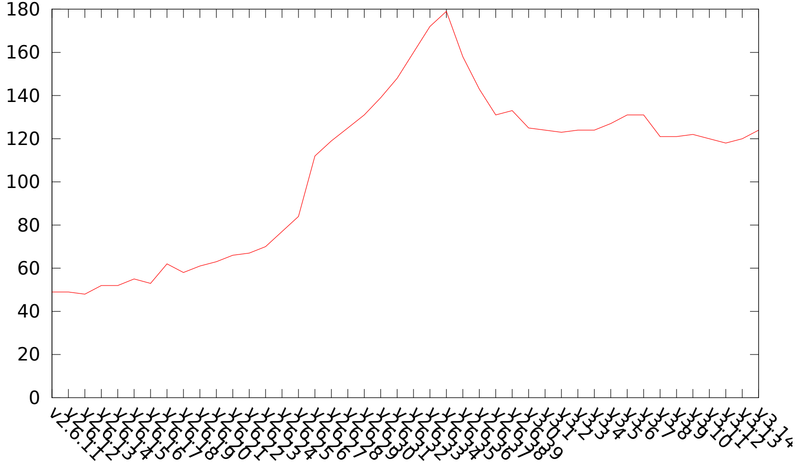
- Differentiate by kernel config
- N devices: N configs, N kernels

Multi-Platform Kernels

- Differentiate by DT
- N devices: 1 config, 1 kernel, N DTs
- Easier to deploy, convenient for Distributions
- Compile-coverage

Why Device Trees?

Evolution of the number of ARM defconfig files



Why Device Trees?

Hardware/Software Reuse

SoCs

- Many have the same IP cores, LEGO-like building blocks
- Just need different DTs!

Boards

- The same SoCs may be used on multiple boards
- Differences are in:
  - Which IP blocks are enabled
  - Child devices
    typically on non/semi-discoverable buses like spi and i2c
  - Clocks, regulators, pinctrl, . . .

IP Core Versioning and Compatibility

Discrete ICs

Before SoCs:

- Hardware block is IC
- *Unique* part number
- Optional name, PCI ID
- Example: `DECchip 21040`, `Tulip`

IP Core Versioning and Compatibility

System on Chip

- No part numbers for hardware blocks ("IP cores")
- Which IP cores? Abstract names? Which version?
- Use SoC part number? SoC family name?
- Examples:
  - `"renesas,scifa"`
  - `"renesas,ether-r8a7791"`
  - `"renesas,gpio-rcar"`
  - `"renesas,rcar_sound-gen2"`
- Softcores:
  - HDL sources for IP core available,
  - OpenRISC: `"opencores,<name>-rtlsvn<version>"`?

IP Core Versioning and Compatibility

2 versions of the IP core are definitely different

- How to represent differences?

2 versions of the IP core are different, but the current Linux driver doesn't care

- Still need to differentiate, future driver versions may use the differences

2 versions of the IP core are the same (same version in 2 SoCs)

- Are they really the same?
- What if they turn out not to be the same later?

Compatible

Generic names vs. specific names

Initially

- DTS: `compatible = "vendor,device-soc<type>", "vendor,device"`
- driver: match `"vendor,device"`

New compatible SoC

- DTS: `compatible = "vendor,device-soc<newtype>", "vendor,device"`
- driver: no changes needed

Compatible

Generic names vs. specific names

New incompatible SoC

- DTS: `compatible = "vendor,device-soc<newtype>"`
- old driver:
  match `"vendor,device"` and `"vendor,device-soc<type>"`
- new driver or enhanced old driver:
  match `"vendor,device-soc<newtype>"`

Stable ABI Nonsense

No stable ABI for in-kernel code

- Module ABI
- Platform data ABI
- Out-of-tree code is second (if any at all) class citizen

Stable ABI Sense

User space ABI is stable

- Small
- Well-thought abstractions (syscalls, `/sys` (hmm), ...)

DT API is stable

- Big, growing, a few orders of magnitudes more changes
- Zillions of different hardware devices
- Complex for complex hardware
- Lots of review to do
(`devicetree@vger.kernel.org` is a more boring firehose than lkml ;-)

Stable DT ABI

Backward/Forward Compatibility, Synchronization

- New optional properties
  - E.g. `"spi-rx-bus-width"`: Dual/Quad SPI
  - SPI core rejects slave if feature not supported by master
  - New DT will not work with old kernel
- Move from device-specific to generic subsystem properties
  - `renesas,clock-indices` and `clock-indices`
  - Update
    - Bindings
    - Subsystem code (incl. backward compatibility)
    - DTS
- What with future external DT repo? How to synchronize?

Complex Topologies

Examples

- SoC module has to change function depending on the state of a GPIO
  - USB host/gadget detection on Lager, via platform data callback in legacy code.
- Graphics
- Audio

Where does the DTB come from?

- DTB is created from `*.dts` and `*.dtsi` by dtc
- DTB is passed from bootloader
  - Where is it stored?
  - How is it updated?
  - Backward compatibility: see *Stable DT API*
- Alternatives:
  - Appended to zImage
  - Included in vmlinux
  - Always up-to-date

Dynamic DT

Hotplug

Device Tree Overlays (WIP)

- Dynamically altering the kernel's live Device Tree
- E.g. BeagleBone (Black) cape plug-in boards

FPGA Platforms

- No fixed DT, hardware may change
- Derive from/store in HDL?

Binding Documentation

- Submit early vs. together with driver patch
- CC devicetree@vger.kernel.org for review
- Use `*-names` if there can be more than one:
  - Registers: `reg` and `reg-names`
  - Interrupts: `interrupts` and `interrupt-names`
  - Clocks: `clocks` and `clock-names`
  - Example:

```
interrupts = <0 238 IRQ_TYPE_LEVEL_HIGH>,
             <0 239 IRQ_TYPE_LEVEL_HIGH>,
             <0 240 IRQ_TYPE_LEVEL_HIGH>;
interrupt-names = "error", "rx", "tx";
```

- List all compatible names in bindings, even if the driver doesn't match against them yet, so checkpatch can validate DTSes against them
  - `Documentation/devicetree/bindings/vendor-prefixes.txt`
  - `Documentation/devicetree/bindings/i2c/trivial-devices.txt`

Bindings

KISS

- Simple bindings:
  - `compatible = ...`
  - + a few properties
- Avoid adding more properties to differentiate
  - You may be/guess wrong about compatibility
  - What if you discover an incompatibility later?
    → Use SoC-specific compatible properties from the start

I think you can do at least some of this without committing to bindings all that early. Keep in mind that bindings can be amended over time, so if you start a driver with a trivial binding you can add properties over time as needed. — *Olof Johansson*

SoC-specific devices

- `arch/<arch>/boot/dts/<soc>.dtsi`
- All possible devices, `status "disabled"`
- Example:

```
sata0: sata@ee300000 {
        compatible = "renesas,sata-r8a7791";
        reg = <0 0xee300000 0 0x2000>;
        interrupts = <0 105 IRQ_TYPE_LEVEL_HIGH>;
        clocks = <&mstp8_clks R8A7791_CLK_SATA0>;
        status = "disabled";
};
```

SoC versus Board

Board

Board-specific devices and configuration

- arch/<arch>/boot/dts/<soc>-<board>.dts
- Include SoC-specific dtsi for SoC-specific devices
- Enable devices: status "ok"
- Child devices for e.g. spi and i2c
- External clocks, pinctrl, aliases, ...
- Example:

```
#include "r8a7791.dtsi"

&i2c6 {
        status = "okay";
        clock-frequency = <100000>;
};
```

Includes and Macro Definitions

Dtc now uses cpp

- Prefer `#include "file.dtsi"` over `/include/ "file.dtsi"`
- Cpp macros in `include/dt-bindings/`
- Can be included as `#include <dt-bindings/....h>` by DTS (and code)
- Useful for e.g. clock indices, or other boring definitions
- Example: `include/dt-bindings/gpio/gpio.h`

```
#define GPIO_ACTIVE_HIGH 0
#define GPIO_ACTIVE_LOW 1
```

- Actual values are part of the DT ABI!

Platform devices and DT compatility

Why?

Sometimes you still want to use platform devices:

- Drivers for IP cores used on legacy platforms
- Platform devices in board code for prototyping
- Sharing with legacy platforms

Think about the upgrade path . . . to DT!

Platform devices and DT compatility

Differences

Platform Devices

- Match by Platform Device Name,
- Platform Device Resources: IO, MMIO, IRQ,
- Platform Data: C-struct, can be anything!

DT

- Match by `compatible`-property,
- `reg`-properties for IO or MMIO,
- `interrupts`-properties,
- `clocks`-properties,
- `pinctrl`-properties,
- Platform, subsystem, bus, and device-specific properties

Platform devices and DT compatility

Platform Data

Avoid platform data

- Esp. callback functions
- "translate" other fields to properties
- Example:

```
 struct rspi_plat_data {
     unsigned int dma_tx_id;
     unsigned int dma_rx_id;
     unsigned dma_width_16bit:1;
+    u16 num_chipselect;
+    u8 data_width;        /* Data reg access width */
+    unsigned txmode:1;    /* TX only mode */
+    unsigned spcr2:1;     /* Set parity register */
 };
```

Use multiple platform device names to differentiate if needed

- Then `of_device_id.data` and `platform_device_id.driver_data` can contain a pointer to parameters, if needed
- Example:

```
static struct platform_device_id spi_driver_ids[] = {
  { "rspi",    (kernel_ulong_t)&rspi_ops },
  { "rspi-rz", (kernel_ulong_t)&rspi_rz_ops },
  { "qspi",    (kernel_ulong_t)&qspi_ops },
  {},
};
static const struct of_device_id rspi_of_match[] = {
  { .compatible = "renesas,rspi",    .data = &rspi_ops },
  { .compatible = "renesas,rspi-rz", .data = &rspi_rz_ops },
  { .compatible = "renesas,qspi",    .data = &qspi_ops },
  {},
};
```

Platform devices and DT compatility

Resources

Use resources: These are automatically compatible

- (named) I/O and MMIO ranges,
- (named) interrupts,
- Named resources allow support for optional/different sets, e.g. separate interrupts vs. one multiplexed interrupt.
- Example:

```
static const struct resource rspi0_resources[] {
        DEFINE_RES_MEM(0xe800c800, 0x24),
        DEFINE_RES_IRQ_NAMED(270, "error"),
        DEFINE_RES_IRQ_NAMED(271, "rx"),
        DEFINE_RES_IRQ_NAMED(272, "tx"),
};
int irq = platform_get_irq_byname(pdev, "rx");
```

Platform devices and DT compatitility

Clocks

Use `NULL` name match:

```
struct clk *clk_get(struct device *dev,
                    const char *con_id);

struct clk *clk = clk_get(&pdev->dev, NULL);
```

Clock name comes from device name:

- Platform device name
- DT node name (DT without Common Clock Framework)
  E.g. `e61f0000.thermal`
- DT clock name (DT with Common Clock Framework), as specified by `clocks`-property

Long Term Support Initiative

- `http://ltsi.linuxfoundation.org`
- LTSI-3.10
- Backporting drivers / SoC / board support
- DT Multi-Platform
- DT Compatibility
  - Submit bindings early
  - Avoid long term support of potentially premature DT bindings

Questions?

- When will m68k migrate to DT?

Thanks & Acknowledgements

- **Renesas Electronics Corporation**, for contracting me to do Linux kernel work,
- The **Linux Foundation**, for organizing this conference and giving me the opportunity to present here,
- The **Renesas Linux Kernel Team**, for DT insights and discussions,
- The **Linux Kernel Community**, for having so much fun working together towards a common goal.