# Why Devicetree Needs to Evolve

Bill Fletcher – Linaro

## CONTENTS



## ABSTRACT

Devicetree is the core technology that enables the Arm ecosystem to build flexible and adaptable embedded systems. The core reason for the existence of Devicetree in Linux is to provide a way to describe non-discoverable hardware. Booting a rich Operating System (OS) image such as Linux on an application-specific System-on-Chip (SoC) requires that a description of the hardware platform configuration is passed to the OS at boot time. The diversity supported by the Arm ecosystem means that the task of describing this hardware configuration systematically is a complex and evolving problem. There are emerging requirements for the Devicetree specification to support such features as a system-level view, enhanced tooling, a location for source files that doesn't encourage duplication and a comprehensive view of the Devicetree lifecycle. This white paper explains the rationale behind the new Devicetree Evolution Lead Project in Linaro.

## INTRODUCTION

When first started, any software has to know what hardware it is booting on. At the simplest level, it may rely on assumptions about the hardware baked into its image at compile time. At the other end of the complexity and flexibility spectrum, it can assume no prior knowledge of the hardware but instead apply some kind of exploration or discovery mechanism. A third option is that it can be given some additional information about the hardware at boot time (as a file or a table) which it uses as a guide to boot and initialise the hardware which that information describes.

These are generalisations. In the real world, all software contains some explicit assumptions about the target hardware it expects to run on. Conversely, some adaptation mechanisms apply even at the simplest level - even an RTOS image may investigate the content a hardware version register and choose between different hard-coded configurations based on the register contents.

### *Sources of Target Hardware Information*

| Source of hardware information | Examples |
|---|---|
| Compile-time assumptions | RTOS running on microcontroller |
| Discovery mechanisms | PC booting and enumerating PCIe bus peripherals |
| Hardware description table or file | PC ACPI table<br>Arm Devicetree file |

Baked-in compile-time assumptions in a code image mean that the relationship between hardware and software image is fixed. Typically only one hardware platform will be supported, and any change to the hardware will require a new image to be built.

This situation would not be appropriate for a rich OS on PC hardware. No one would expect to rebuild Windows to run on their particular PC. PC hardware is relatively commodified and generally the OS knows what to expect. An image booting on a PC also receives a description of the underlying hardware in the form of the ACPI tables. Also, some bus-connected peripherals are discoverable at boot time.

Booting a rich OS image such as Linux on an application-specific System-on-Chip (SoC) has particular challenges. This is a common case in the Arm ecosystem. All application-specific SoCs are different by definition. The OS does not know what hardware to expect by default and yet (as with the PC) there's no desire to produce an OS that can only boot on one specific platform. The solution to this conundrum is to pass a description of the hardware platform configuration to the OS at boot time. For Arm based embedded SoCs, the specification for this configuration is called Devicetree.

## ABOUT DEVICETREE

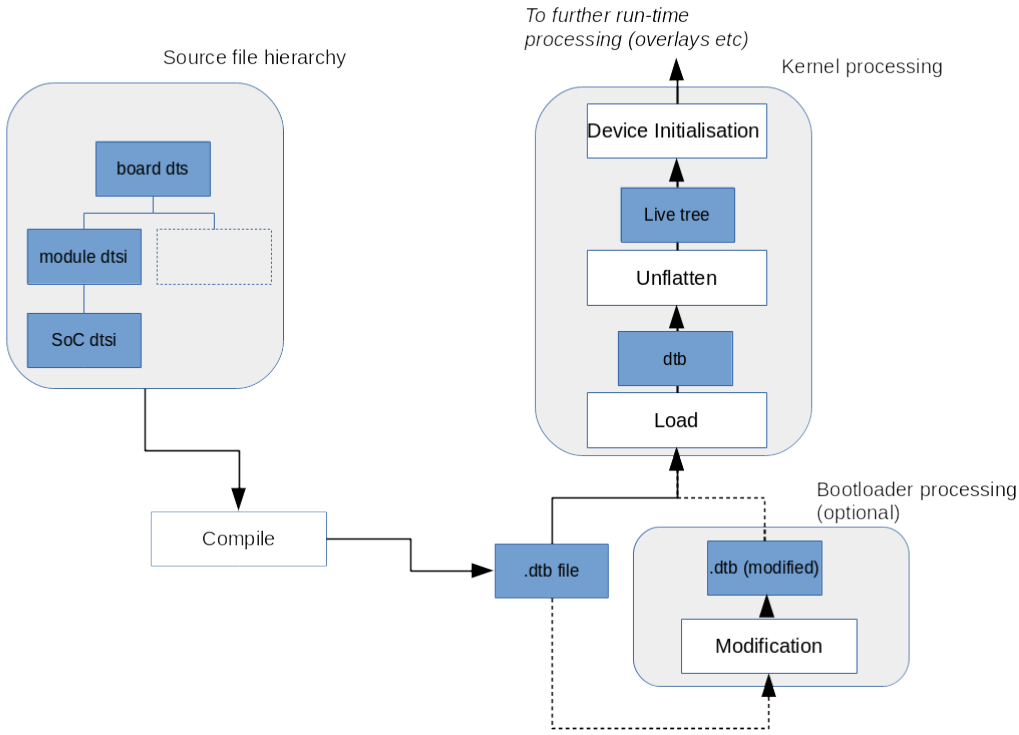
Devicetree is the core technology that enables the Arm ecosystem to build flexible and adaptable embedded systems. The core reason for the existence of Devicetree in Linux is to provide a way to describe non-discoverable hardware.

The ecosystem supported by the Arm CPU architecture is very diverse. Embedded application processors contain a custom selection of peripherals from a wide range of potential options and memory interfaces. The vast majority of these peripherals and interfaces need to be described in configuration information which is typically passed to the OS at boot time. The diversity supported by the Arm ecosystem means that the task of describing this hardware configuration systematically is a complex problem.

Devicetree data can be represented in several different formats. It was originally derived from the Devicetree format used by Open Firmware to encapsulate platform information. The Devicetree data is typically created and maintained in a human readable format in .dts and .dtsi source files.

The Devicetree source is compiled into a binary format contained in a Devicetree Blob (.dtb) file. The format of the data in the .dtb file is also referred to as a Flattened Devicetree (FDT). The Linux operating system uses the Devicetree data to find and register the devices in the system. The FDT is accessed in the raw form during the very early phases of boot, but is unpacked into a live tree for more efficient access for later phases of the boot and after the system has completed booting.

*Devicetree Lifecycle Diagram*




The Devicetree format is expressive and able to describe most board design aspects including:

- the number and type of CPUs
- base addresses and size of RAM
- busses and bridges
- peripheral device connections
- interrupt controllers and IRQ line connections
- pin multiplexing
- Clock and power domains

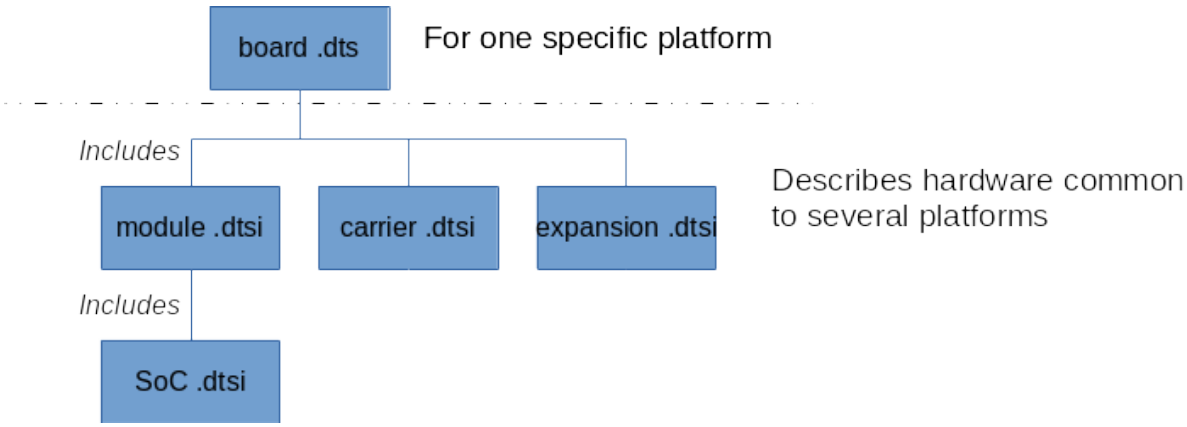
## THE CURRENT STATE OF DEVICETREE

To deal with the complexity in handling families of related SoCs, and also the same SoC populated into different evaluation of production boards, the source format of .dts and .dtsi files in the kernel is already structured to separate SoC family, specific SoC, and board.

The pattern is:

- SoC family defines all common attributes of family
- Specific SoC adds portions specific to that SoC
- Board disables any peripheral that does not have a meaningful connection

### *Devicetree files hierarchy*




Devicetree is a successful project used to configure not only the Linux kernel, but also many other software components. For example: Trusted Firmware (TF-A), U-Boot, OP-TEE, and also the XEN hypervisor and Zephyr RTOS.

### *Some software components that use Devicetree*

| Typical Linux Boot | Optional additions to Linux | Other OS or Hypervisor |
|---|---|---|
| Trusted Firmware (TF-A) | OP-TEE | Zephyr |
| U-Boot | | Xen |
| Linux kernel | | FreeBSD |

A number of enhanced use cases for Devicetree already exist. A 'master' Devicetree can be parsed. A subset configuration might be used by a bootloader, or multiple parallel subsets might be used to allocate static resources between hypervisor client partitions. Multiple different system views can allow for heterogeneous multiprocessing, for example on a complex SoC or FPGA.

### *Devicetree enhanced use cases*

| Use case | Example |
|---|---|
| Parsing for one subset | U-boot's ftdgrep tool selects a subset of the u-boot dtb for use in SPL |
| Parsing for multiple parallel subsets | Xen in Dom-0 less mode uses DT formats to partition resources between domains |
| Parsing for different heterogeneous cores | Tools to generate configurations for heterogeneous cores in FPGA designs |
| Assembling a Devicetree from separate binaries | Android is standardizing on separation of SOC DTB and OEM (aka "phone") DTB. They have a partition format for each. |
| Compile-time generation of configuration for RTOS | Zephyr is doing code generation today from DT on a case by case basis |

## WHY DEVICETREE NEEDS TO EVOLVE

### *System View*

Multiple software components need to reference subsets of the description of the hardware e.g. for boot, security or virtualisation. To avoid Devicetree diversity or source file duplication between these software components, there needs to be a unique system level Devicetree definition that will be referenced in some form by all software components running on the hardware platform.

Devicetree usage should be defined for each boot stage and execution context. This could be a unique DTB that can be modified on the fly by the different firmware, or a fixed DTB by boot stage. Potentially it could also define the way to provide data in a verified/secured way between the different boot stages.

Similarly the system view should provide a unique system description that contains all peripherals and memories, with an associated view per processor complex - these could be (real or virtualized) Cortex-A, Cortex-M or DSP cores.

The system view should manage peripherals and memory carveouts assignment between the different execution context and the different software components and include "virtualized" devices like secure clocks handled by firmware.

The approach needs to work across heterogeneous SoCs and FPGAs, board variations, bootloaders and OSes.

### *Tools*

As Devicetree evolves towards the increased complexity of supporting an overall system view, it's important to be sure that the source code description for the system is reasonably sane, rather than only syntactically correct. Currently the Devicetree compiler dtc only makes syntax checks. For example, it doesn't make any conformance checks against bindings.

Currently there isn't any way for an end-user to know which information in the Devicetree is configurable and which information is considered static. Having the mutability of properties defined and having some reference tool example that would expose user modifiable information and then generate a Devicetree would be extremely useful. Having this infrastructure would both improve the user experience and make it less error prone to configure/modify a Devicetree as well as allow for third party implementations of configuration tools.

### *Source File Location*

Devicetree source files are located in the Linux kernel sources under the arch directory. For Arm systems, they are in arch/arm and arch/arm64. Other software components that reference them need to decide whether to reference the source files from outside, or maintain a local copy. In these additional external use cases, although the Devicetree description language used by the different components is the same, the way it is used may be different: the bindings may not be 100% aligned. Ultimately this means that, even for the same SoC, the source .dts files and associated definitions have to be duplicated. This is despite the fact that the Devicetree sources are intended to be OS independent.

.dts files need to be in a single separate repository that all users can work on and use equally. This will be shared by all software components to avoid file duplication. An issue is that on one side, software components want to be self-contained and have all required information/files to run, but SoC vendors would also like to limit file duplication and the associated maintenance.

### *Lifecycle*

The Devicetree lifecycle needs to be better defined. Devicetrees may be assembled on-the-fly with overlays, verified or updated. This may need to accommodate, or at least detect, differences in specification version between old and new, or base and overlay .dtb files. i.e.

- It should be possible to verify the DTB and identify the DTB version like any software components for support, validation, deployment purposes
- Devicetree overlays should be included in the lifecycle model
- Provide mechanisms to update the DTB

- Provide a mechanism to be able to deprecate legacy bindings whilst maintaining support for old kernels during the transition.

### *Specification*

The last official release of the Devicetree specification was Dec 20, 2017 (v0.2). There are several open issues on github related to the specification. The specification should evolve to capture the above evolution of Devicetree.

## DEVICETREE EVOLUTION - A LINARO LEAD PROJECT

Linaro has defined a Lead Project with its membership to address the needs of the Arm ecosystem as summarised above. The Devicetree Evolution project aims to:

- collect and consolidate the different Devicetree requirements from the Linaro membership and the ecosystem.
- update the different specifications (Devicetree, EBBR,...) as needed
- work to ensure coherency between all software components using DT

The project is expected to deliver:

- A Devicetree specification update
- Modification of each software component to support the Devicetree evolution
- Implementation on selected reference platforms

Groups and projects within Linaro will provide a set of reference platforms covering the Devicetree diversity we want to address with this project and will provide regression testing and maintenance.

The project has several logical areas that we are proposing to track as individual Initiatives. It is expected that multiple different Linaro member engineers and working groups will focus on these Initiatives. We will further sub-divide the Initiatives and add more detail as necessary as we track work at the Epic and Story level in Linaro's Jira database.

## ABOUT LINARO

Together with Arm, Linaro co-maintains the Arm software ecosystem, providing the tools, security and Linux kernel quality needed for a solid base to differentiate on. Enabling markets on Arm architecture since 2010, Linaro works with companies to consolidate Arm code bases in the ecosystem as a whole, as well as in specific market segments. We do this by providing an engineering forum where industry and community can work together on open source software to solve common problems. This collaborative approach reduces software fragmentation across the many Arm platforms and enables industry and community to reduce costs for development and validation of Arm-based software.

## ABOUT LINARO LEAD PROJECTS

Linaro Lead Projects are significant activities spanning working groups, market segments and the Linaro Membership to deliver specific key work items. They are:

- created via the Linaro Big Ideas Process
- have a beginning and an end
- have a charter
- are approved by the Linaro TSC
- are reviewed for relevance during each Linaro Connect

## REFERENCES

https://binarydebt.wordpress.com/2018/10/06/how-does-an-x86-processor-boot/
https://events.static.linuxfound.org/sites/events/files/slides/petazzoni-device-tree-dummies.pdf
https://elinux.org/images/1/15/Device_tree_in_U-Boot_SPL.pdf

## MORE INFORMATION

**Contact the author**
bill.fletcher@linaro.org

**Contact Linaro**
https://www.linaro.org/contact/

**Attend Linaro Connect**
https://connect.linaro.org/