Device Tree

From eLinux.org

Contents

Request for Documentation Suggestions

If you have any comments or suggestions about the Device Tree documentation on elinux.org, please send them to frank.rowand@sonymobile.com

I am currently trying to make the information more organized, more comprehensive, and a more complete index of information available elsewhere. I am looking for comments on what is incorrect, incomplete, or missing. I would appreciate pointers to good documentation, tutorials, etc that I can link to.

Introduction

Device Tree data can be represented in several different formats. It is derived from the device tree format used by Open Firmware to encapsulate platform information and convey it to the Linux operating system. The device tree data is typically created and maintained in a human readable format in .dts source files and .dtsi source include files. The Linux build system pre-processes the source with cpp.

The device tree source is compiled into a binary format contained in a .dtb blob file. The format of the data in the .dtb blob file is commonly referred to as a Flattened Device Tree (FDT). The Linux operating system uses the device tree 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 expanded into a kernel internal data structure for more efficient access for later phases of the boot and after the system has completed booting.

Currently the Linux kernel can read device tree information in the ARM, x86, Microblaze, PowerPC, and Sparc architectures. There is interest in extending support for device trees to other platforms, to unify the handling of platform description across kernel architectures.

The Flattened Device Tree is...

The Flattened Device Tree (FDT) is a data structure. Nothing more.

It describes a machine hardware configuration. It is derived from the device tree format used by Open Firmware. The 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, and
- interrupt controllers and IRQ line connections.

Just like initrd images, an FDT image can either be statically linked into the kernel or passed to the kernel at boot time.

The Flattened Device Tree is not...

- is not a solution to all board port problems
  - Nothing will eliminate all board specific drivers for custom and complex boards.
- is not a firmware interface
  - It might be part of a generic firmware interface, but on its own the device tree is just a data structure.
  - does not replace ATAGS... but an FDT image can be passed via an ATAG.
  - See "Competing Solutions" below
- is not intended to be a universal interface.
  - It is a useful data structure which solves several problems, but whether or not to use it is still up to the board port author.
- is not an invasive change
  - ~~No requirement to use FDT approach in a board port~~
  - Device Tree is required for new board support in the ARM architecture.
  - No requirement to convert existing board ports
  - No requirement to modify existing firmware

History

- How device tree got into Linux and how it has evolved

Future

- How device tree is changing and where it is headed

Advantages

for distributions

- potentially fewer kernel images needed on an installer image (ie. for ARM netbooks)
  - Ship one FDT image per machine (<4k/machine) instead of 1 kernel image per machine (~1-2MB/machine) with a small number of sub-arch kernel images (ie. ARM11, CortexA8, CortexA9, etc).
  - Becomes feasible for current installer image to boot on future hardware platforms using same chipset.
  - Note: FDT is only part of the solution here. Some boot software is still required to select and pass in the correct FDT image.

for System on Chip (SoC) vendors

- Reduce or eliminate effort needed to write machine support code (ie arch/arm/mach-*). Focus on device driver development instead.

for board designers

- Reduce effort required to port.
  - SoC vendor supplied reference design binaries may also be bootable on custom machine.
- No need to allocate a new global ARM machine id for each new board variant.
  - Use the device tree <vendor>,<boardname> namespace instead
- Most board specific code changes constrained to device tree file and device drivers.
- Example: Xilinx FPGA toolchain has a tool to generate a device tree source file from the FPGA design files.
  - Since the hardware description is constrained to the device tree source, FPGA engineers can test design changes without getting involved with kernel code.
  - Alternately, kernel coders don't need to manually extract design changes from the FPGA design files.

for embedded Linux ecosystem

- smaller amount of board support code to merge
- greater likelyhood of mainline support for boards from "uninterested" vendors
- greater ability to correct poor board support by fixing or replacing broken FDT images.

for firmware/bootloader developers

- reduce impact of getting board description wrong (FDT stored as a separate image instead of statically linked into firmware). If initial release gets the board description wrong, then it is easily updated without a risky reflash of firmware.
- expressive format to describe related board variants without allocating new machine numbers or new ATAGs.
- Note: The FDT isn't a replacement to ATAGS, but does supplement them.

Other advantages

- Device tree source and FDTs can easily be machine generated and/or modified.
  - Xilinx FPGA tools do device tree source generation
  - U-Boot firmware can inspect and modify an FDT image before booting

Competing Solutions

board specific data structures

Some platforms use board-specific C data structures for passing data from the bootloader to the kernel. Notable here is embedded PowerPC support before standardizing on the FDT data format.

Experience with PowerPC demonstrated that using a custom C data structure is certainly an expedient solution for small amounts of data, but it causes maintainability issues in the long term and it doesn't make any attempt to solve the problem of describing the board configuration as a whole. Special cases tend to grow and there is no way for the kernel to determine what specific version of the data structure is passed to it. PowerPCs board info structure ended up being a mess of #ifdefs and ugly hacks, and it still only passed a handful of data like memory size and Ethernet MAC addresses.

ATAGs have the elegance of providing an well defined namespace for passing individual data items (memory regions, initrd address, etc) and the operating system can reliably decode them. However, only a dozen or so ATAGs are defined and is not expressive enough to describe the board design. Using ATAGs essentially requires a separate machine number to be allocated for each board variant, even if they are based on the same design.

That being said, an ATAG is an ideal method for passing an FDT image to the kernel in the same way an ATAG is used to pass the initrd address.

ACPI

Firmware providing the Advanced Configuration and Power Interface (http://en.wikipedia.org/wiki/Advanced_Configuration_and_Power_Interface) exports a hardware description in the form of the Differentiated System Description Table (DSDT). ACPI is found on x86 compatible systems and has its roots in the original IBM PC BIOS.

UEFI

The Extensible Firmware Interface (http://en.wikipedia.org/wiki/Extensible_Firmware_Interface) is an interface specification for passing control from a platforms firmware to the operating system. It was designed by Intel as a replacement for the PC BIOS interface.

ARM holdings is a member (http://www.uefi.org/join/list) of the United EFI Forum (http://www.uefi.org/home). It is conceivable that there will be an ARM implementation of UEFI.

Open Firmware

Open Firmware (http://en.wikipedia.org/wiki/Open_Firmware) is a firmware interface specification designed by Sun in the late 1980's, and ported to many architectures. It specifies a runtime OS client interface, an cross platform device interface (FCode (http://www.openfirmware.info/Forth/FCode)), a user interface, and the Device Tree layout for describing the machine.

FDT is to Open Firmware what DSDT is to ACPI. The FDT reuses Open Firmware's established device tree layout. In fact, Linux PowerPC support uses the same codebase to support both Open Firmware and FDT platforms.

Some Notes on the Competing Solutions

Most of the competing solutions listed above provided feature rich firmware interfaces including both machine description and runtime services. Conversely, the FDT is only a data structure and doesn't specify any firmware interface details. Board ports using the FDT are typically booted from simple firmware implementations like U-Boot and don't provide any form of runtime services.

A common design goal of the feature rich firmware interfaces is to provide an abstract boot interface that factors away the differences between different hardware platforms, at least enough for the OS to initialize its own native device drivers. The idea is to be able to boot 'old' OS images on 'new' hardware, like how a Linux LiveCD image doesn't have explicit knowledge of the hardware configuration, but relies on the information provided to it by firmware.

Typical design goals for embedded firmware is to a) boot the OS as quickly as possible, b) upgrade the OS image, and maybe c) provide some low level debug support during initial board bringup. Focus tends to shift away from firmware once the OS is bootable since the kernel drivers the hardware directly (doesn't depend on firmware runtime services). In fact, firmware updates are discouraged due to the risk of rendering a board unbootable. ACPI, UEFI and OpenFirmware solutions, while arguably 'better', often don't boot as fast, and are more complex than required by the embedded system. In this regard the FDT approach has the advantage due to its simplicity. ie. the FDT provides an equivalently expressive way to describe hardware, but it works with existing firmware and can be updated without reflashing firmware.

Resources

Wiki and in-kernel documentation

The main device Tree wiki is at: http://www.devicetree.org/Main_Page

```
http://www.devicetree.org/Device_Tree_Usage is an excellent introduction to device tree
concepts and the representation of those concepts in device tree source.  (But check the
"last modified" date at the bottom of the page (currently 23 October 2010) when comparing
to other resources.)
```

Documentation about device tree is available in the Linux kernel Documentation directory: Documentation/devicetree. See https://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/tree/Documentation/devicetree

Some especially useful files are:

- ABI.txt: comments on stable binding and general bindings rules
- resource-names.txt: -name properties containing an ordered list of names corresponding to another property
- usage-model.txt: information about different elements of bindings
- vendor-prefixes.txt: vendor prefix registry
- the bindings directory has details about the syntax and expected elements for each device type representable in the dts and used by kernel frameworks and drivers
- bindings/submitting-patches.txt: important details for
  - patch submitters
  - kernel maintainers

FAQ, tips and/or best practices

See the Linux Drivers Device Tree Guide.

Presentations, Papers and Articles

- Solving Device Tree Issues, LinuxCon Japan 2015 by Frank Rowand
- "The Device Tree as a Stable ABI: A Fairy Tale?", ELC 2015 by Thomas Petazzoni
  - http://elinux.org/images/0/0a/The_Device_Tree_as_a_Stable_ABI-_A_Fairy_Tale%3F.pdf
- "Transactional Device Tree & Overlays: Making Reconfigurable Hardware Work", ELC 2015 by Pantelis Antoniou
  - PDF
  - YouTube video (http://www.youtube.com/watch?v=3Ag7ZBC_Nts)
- "Device Tree for Dummies", ELC 2014 by Thomas Petazzoni
  - PDF
  - YouTube video (https://www.youtube.com/watch?v=uzBwHFjJ0vU)
- "Engaging Device Trees", ELC 2014 by Geert Uytterhoeven
  - PDF
  - YouTube video (https://www.youtube.com/watch?v=MlYeT_nUK4Y)
- "Trees need care: A Solution to Device Tree Validation Problem", ELC 2014 by Tomasz Figa
  - PDF
  - Free Electrons Videos (http://free-electrons.com/blog/elc2014-videos/)
- Device trees I: Are we having fun yet? (https://lwn.net/Articles/572692/) - Neil Brown, LWN.net November 2013
- Device trees II: The harder parts (https://lwn.net/Articles/573409/) - Neil Brown, LWN.net November 2013
- "Device Tree for Dummies", ELC Europe 2013 by Thomas Petazzoni
  - PDF
  - YouTube video (https://www.youtube.com/watch?v=m_NyYEBxfn8)
- "Transactional Device Tree & Overlays: Making Reconfigurable Hardware Work", ELC Europe 2014 by Pantelis Antoniou
  - Media:Antoniou--transactional_device_tree_and_overlays.pdf
- "devicetree: Kernel Internals and Practical Troubleshooting", ELC Europe 2014 by Frank Rowand
  - Media:Rowand--devicetree_kernel_internals.pdf
- "Device Tree, the Disaster so Far", ELC Europe 2013 by Mark Rutland
  - Media:Rutland-presentation_3.pdf
  - YouTube video (https://www.youtube.com/watch?v=xamjHjjyeBI)
- "Best Practices for Long Term Support and Security of the Device-Tree (DT)" ELC Europe 2013 by Alison Chaiken
  - Media:Chaiken-DT_ELCE_2013.pdf
- "Board file to Device Tree Migration" ELC Europe 2013 by Pantelis Antoniou
  - Media:ELCE2013_-_DT_War.pdf
- "ARM support in the Linux kernel", Presented at FOSDEM 2013 by Thomas Petazzoni
  - https://archive.fosdem.org/2013/schedule/event/arm_in_the_linux_kernel/attachments/slides/273/export/events/attachments/arm_in_the_linux_kernel/slides/273/arm_support_kernel.pdf
  - Has good material on how device tree is part of the overall ARM architecture refactoring, with some details on how it is used
- "Linux kernel: consolidation in the ARM architecture support" - Libre Software Meeting, 2013 by Thomas Petazzoni
  - http://free-electrons.com/pub/conferences/2012/lsm/arm-kernel-consolidation/arm-kernel-consolidation.pdf
- "Experiences With Device Tree Support Development For ARM-Based SOC's", Thomas P. Abraham, ELC 2012
  - Media:Experiences_With_Device_Tree_Support_Development_For_ARM-Based_SOC's.pdf
  - slides and videos for ELC 2012: http://free-electrons.com/blog/elc-2012-videos/
- "Device Tree Status Report", Grant Likely, ELC Europe 2011
  - Slides and videos for ELC Europe 2011: http://free-electrons.com/blog/elce-2011-videos/

Notes on various sub-systems that device-tree describes

- "Pin Control Subsystem – Building Pins and GPIO from the ground up" - Presented at Linaro Connect, 2013 by Linus Walleij
  - http://www.df.lth.se/~triad/papers/pincontrol.pdf

older stuff

There is documentation describing device tree support (with information current as of 2006) in the Linux kernel source tree at: Documentation/powerpc/booting-without-of.txt (http://git.kernel.org/?p=linux/kernel/git/torvalds/linux-2.6.git;a=blob_plain;f=Documentation/powerpc/booting-without-of.txt;hb=HEAD)

- "Using the Device Tree to Describe Embedded Hardware" - Grant Likely, Embedded Linux Conference, 2008
  - http://www.celinux.org/elc08_presentations/glikely--device-tree.pdf
- "A Symphony of Flavours: Using the device tree to describe embedded hardware" - Grant Likely and Josh Boyer - paper for OLS 2008
  - http://ols.fedoraproject.org/OLS/Reprints-2008/likely2-reprint.pdf
- Note from Device Tree Birds of a Feature session at OLS 2008:
  - http://lists.ozlabs.org/pipermail/devicetree-discuss/2008-July/000004.html
- Links to the Open Firmware device tree bindings and recommended practices which also apply to the FDT:
  - http://www.openfirmware.info/Bindings
- A view from outside from the FreeBSD ARM community:
  - http://wiki.freebsd.org/FreeBSDArmBoards

Tools

- Device Tree Compiler (dtc) - converts between the human editable device tree source "dts" format and the compact device tree blob "dtb" representation usable by the kernel or assembler

source. dtc is also a dtb decompiler.
    - The linux version of dtc is maintained in `scripts/dtc/` in the kernel source directory.
    - The upstream project is maintained in
        - https://git.kernel.org/cgit/utils/dtc/dtc.git
        - git clone git://git.kernel.org/pub/scm/utils/dtc/dtc.git
- Xilinx EDK device-tree generator - Generates an FDT from Xilinx FPGA design files.
    - http://xilinx.wikidot.com/device-tree-generator

```
"The device tree generator is a Xilinx EDK tool that plugs into the
Automatic BSP Generation features of the tool, XPS"
```

Debugging

You can set CONFIG_PROC_DEVICETREE to be able to see the device tree information in /proc after booting. Build the kernel with this option set to 'Y', boot the kernel, then 'cd /proc/device-tree'

For newer kernels where the CONFIG_PROC_DEVICETREE option does not exist, /proc/device-tree will be created if CONFIG_PROC_FS is set to 'Y'.

You might also try CONFIG_DEBUG_DRIVER=Y.

Also, often, you can set the line: "#define DEBUG 1" to an individual C file, to produce add debug statements to the routines in that file. This will activate any pr_debug() lines in the source for that file.

Alternatively, you can add the following to drivers/of/Makefile:

```
CFLAGS_base.o := -DDEBUG
CFLAGS_device.o := -DDEBUG
CFLAGS_platform.o := -DDEBUG
CFLAGS_fdt.o := -DDEBUG
```

Device-Tree irc

The Device Tree irc channel is #devicetree on freenode.net.

Device-tree Mailing List

After July 2013:

```
http://vger.kernel.org/vger-lists.html#devicetree
archive: http://www.spinics.net/lists/devicetree/
archive: http://dir.gmane.org/gmane.linux.drivers.devicetree
```

Up through July 2013:

```
https://lists.ozlabs.org/listinfo/devicetree-discuss
archive: https://lists.ozlabs.org/pipermail/devicetree-discuss/
archive: http://news.gmane.org/gmane.linux.drivers.devicetree
```

Core devicetree binding Mailing List

```
http://vger.kernel.org/vger-lists.html#devicetree-spec
archive: http://dir.gmane.org/gmane.comp.devicetree.spec
```

Device-tree Compiler and Tools Mailing List

```
http://vger.kernel.org/vger-lists.html#devicetree-compiler
archive: http://dir.gmane.org/gmane.comp.devicetree.compiler
```

Retrieved from "http://elinux.org/index.php?title=Device_Tree&oldid=387941"

Categories: Bootloader | Device tree | Kernel
