Device trees II: The harder parts

This article brought to you by LWN subscribers

Subscribers to LWN.net made this article — and everything that surrounds it — possible. If you appreciate our content, please buy a subscription and make the next set of articles possible.

November 18, 2013
This article was contributed by Neil Brown

A devicetree describes the hardware in a system using a tree of nodes, with each node describing a single device. As we observed last week, there are often relationships between devices which do not fit with the model of a strict hierarchical tree. Devicetree can address these needs through a range of techniques best described as cross-tree linkages.

Cross-tree linkages

Two of the more messy things to deal with in board files are interrupts and GPIOs (General Purpose Input/Output pins). This is because there are several different interrupt controllers and several different GPIO controllers. Both interrupts and GPIOs are identified by simple numbers; keeping track of the allocation of those numbers can become clumsy.

In the GTA04, the OMAP3 SoC contains 16 banks of 32 GPIOs, which can reasonably be treated as a single block and will probably be numbered 0-191. The twl4030 has a further 18 GPIO lines (unused) which will presumably be 192-209. The tca6507 LED driver can be configured to treat any of the seven output lines as a GPIO and one of them is. So it is GPIO 210.

There are two approaches to tracking these numbers. One is to hard-code the numbers, or at least to use lots of `#defines` like:

```
    #define GPIO_WIFI_RESET (OMAP_MAX_GPIO_LINES + TWL4030_GPIO_MAX)
```

This approach is simple but can be fragile in the face of change. The other is to use callbacks.

When the "gpio-twl4030" driver registers its 18 GPIOs it will be assigned a range of numbers; it would be best to not assume what numbers they will be until they are assigned. To this end, the platform_data provided to gpio-twl4030 can include a function to be called when initialization is complete, as is used by board-omap3beagle.c in the beagle_twl_gpio_setup(). This function can then store the numbers where appropriate and register the platform devices which depend on those GPIOs.

This hand-coded, delayed initialization can get very messy and is consequently error prone. Devicetree (that bringer of joy) makes this much easier. When one device depends on the service of another, such as a GPIO, an interrupt, a regulator, a timer, etc., the target device is identified by a reference to the relevant node in the devicetree. Unfortunately there is not as much uniformity here as we might like.

To reference an interrupt, the controller node and the interrupt number within the set controlled by that node are given separately, so:

```
interrupt-parent = <&intc>;
interrupts = <76>;
```

means that the interrupt to attach to is number 76 of those controlled by the node called "intc".

If no "interrupt-parent" is present, the ancestors of the current node are searched until either "interrupt-parent" or "interrupt-controller" is found. In the latter case the node containing "interrupt-controller" is the target node.

If a node responds to interrupts from different controllers, that situation cannot be represented with this approach. For that reason there is work to provide a syntax like:

```
interrupts-extended = <&intc 76>
```

so the parent and the offset are both specified for each interrupt.

Depending on the interrupt controller, it might be necessary to specify more than one number to identify an interrupt. The number of numbers needed is specified with the attribute "#interrupt-cells" in the node for the interrupt controller. The exact meaning of the numbers can only be discovered by examining the devicetree bindings documentation (or the code); often one number will contain flag bits describing whether the interrupt should be edge- or level-triggered and whether high or low (or both) levels are interesting.

To reference a GPIO a syntax similar to the proposed "interrupts-extended" is the standard, so:

```
gpios = <&gpio1 7 GPIO_ACTIVE_HIGH>;
```

(where `GPIO_ACTIVE_HIGH` is defined somewhere in `include/dt-bindings/`) will sort out the required GPIO number.

Naturally, in each case the device which provides the interrupt or GPIO will need to be initialized before it can be found and used. It wasn't very many kernel versions ago that this was a real problem. However in the 3.4 kernel, drivers gained the ability for their initialization (or "probe") routine to return the error "`EPROBE_DEFER`" which would cause the initialization to be tried again later. So if a driver finds that a GPIO line is listed in the devicetree, but no driver has registered GPIOs for the target node yet, it can fail with `EPROBE_DEFER` and know it can try again later. This can even be used to remove the need for callbacks and delayed registration in board files, but it is really essential for devicetree, and happily it works quite well.

It is worth highlighting that the standard attribute name to identify the GPIO for a device is "`gpios`" in the plural because, of course, a device might require multiple GPIOs and the descriptors can simply be listed on the one line. Hunting through the sample devicetree files in `arch/arm/boot/dts`, one finds extremely few cases where multiple GPIOs are specified in one attribute. What seems to happen more often is that there are multiple different "*xx*`-gpios`" attributes. For example, an MMC card driver might expect a "`cd-gpios`" to identify the "Card Detect" line, and a "`wp-gpios`" to identify the "Write Protect" line. This approach has the benefit of being more explicit (and so less confusing) and of making it easy to indicate that a particular line is simply not present on some board.

While interrupts and GPIOs allow a list of targets with some implicit meaning, regulators don't. Every request for a regulator must include a supply name, so the battery charger declares its dependence on a regulator with:

```
    bci3v1-supply = <&vusb3v1>;
```

Every regulator request is of the form "*xxx*`-supply`".

As described above, interrupts can sometimes specify the type of trigger, and GPIOs can sometimes specify the active level. Regulators, instead, have no extra parameters that can be passed, even though it would sometimes be useful to specify the required voltage — many regulators are programmable and the GTA04 WiFi chip requires 3.15 volts, which isn't the default (and which cannot yet be set at all using devicetree).

One final cross-tree linkage is implicit in the "`reg`" attribute mentioned in part 1. As with the "`interrupts`" attribute, the device that provides the registers is implicit, though, unlike "`interrupts`", it cannot even be made explicit with something like "`reg-parent`". Rather the device that provides the registers is always exactly the parent of the device which uses the registers.

We've already observed that a hierarchical tree often cannot accurately reflect reality. Were we to create a "`reg-extended`" attribute following the pattern of "`interrupts-extended`" we may well be able to discard the hierarchy altogether and replace the "device tree" with a "device list" where each device contains references to the other devices that it depends on to provide registers, interrupts, GPIOs, etc. This is already happening to some extent. Many so-called "platform devices" are described by devicetree nodes which appear at the top level of the tree rather than where they fit in a device hierarchy.

A simple example is the "aux-keys" device node for the GTA04.

```
aux-keys {
    compatible = "gpio-keys";

    aux-button {
        label = "aux";
        linux,code = <169>;
        gpios = <&gpio1 7 GPIO_ACTIVE_HIGH>;
        gpio-key,wakeup;
        pinctrl-names = "default";
        pinctrl-0 = <&aux_pins>;
    };
};
```

The GTA04 has two physical buttons, one of which is referred to as the "AUX" button and is connected to a GPIO input on the OMAP3. This node describes that part of the hardware. As you can see it identifies a particular GPIO, notes the key-code that it should generate in Linux, asserts that the key could wake the device from suspend, and gives some "`pinctrl`" information which assures that the particular pad on the OMAP3 is configured as a GPIO input.

Given what we have already learned about the tree structure of devicetree you might expect this node to appear as a child of a node describing the particular GPIO, which in turn would be a child of the GPIO controller within the OMAP3. However that is not the case. Instead, this "`aux-keys`" node appears at the top level, immediately under "`/`". While this seems a little odd in the case of a single button, it would make more sense if you imagined a device with multiple related buttons, such as volume-up and volume-down. If they were wired to two separate GPIOs, then placing the node as a child of both GPIOs is impossible and as a child of either would be untidy. Having two separate nodes (one per key) would obscure the fact that there is a single conceptual device: the "volume control".

So we find that some devices, such as the accelerometer and other sensors on the I2C bus, appear in devicetree at the end of a path reflecting how the CPU would address the device, some devices such as a GPIO-based keypad exist at the top level and refer to the components that they combine, and still other devices, such as the GSM modem in the GTA04, cannot be represented as a single device at all.

Not all fricasseed frogs and eel pie.

While exploring and enabling devicetree for the GTA04 has been a lot of fun, there have been some less exciting discoveries.

Firstly, the fact that devicetree support is still quite incomplete is a mixed blessing. On the one hand it is very easy to add devicetree support to many devices and this results in a positive feeling of achievement. On the other, there are fairly significant elements of functionality that are far from trivial. These, such as the `omap-dss` display driver and the cpu-freq support for OMAP, can largely be worked-around by hacking in some old-style "board-file" style initialization, but that isn't nearly so rewarding.

Secondly the devicetree compiler "`dtc`" which converts `.dts` source files to `.dtb` binaries is fairly primitive. If you do something wrong you'll mostly get either:

```
Error: /home/git/gta04-mainline/arch/arm/boot/dts/omap3-gta04.dts:407.12-13 syntax error
FATAL ERROR: Unable to parse input tree
```

or silent success as ARM maintainer Russell King recently observed (there are a couple of other error messages, but not many).

The compiler will often succeed for files which will make no sense to the kernel because there is no checking for the validity of attribute names or value ranges. The kernel does have fairly good schema documentation in "`Documentation/devicetree/bindings`", but this is not machine readable and `dtc` couldn't read it even if it were. Fortunately there is hope on the horizon. Tomasz Figa recently posted a proposed mechanism for writing machine-readable devicetree schemata which would allow more checking to be added to `dtc`. A proposal is certainly a long way from working code, but this is still an encouraging step.

What does the future hold?

Devicetree has already had clear benefits, such as the fun this author had in learning something new and the various cleanups that it has motivated in the driver support code in Linux. However it has also required a substantial amount of effort and that effort is ongoing. Such effort needs more justification than some fun and cleanup.

The significant benefit that devicetree promises is for operating system vendors and their clients. Currently, a Linux distribution can create a release for the x86_64 architecture and it can be expected to run on every x86_64 machine in existence. This is because there is just one platform to target. This is not the case for ARM. For ARM, there are many platforms. However if we can expect every ARM device to come with a devicetree description, then the Linux distributors could target "ARM + devicetree", and that is a credible platform concept.

If we could get to the point where a device plus a devicetree file could be reasonably expected to run with every subsequent release of Linux, then that would be a happy place to be. I would be able to upgrade Debian or openSUSE on my device with confidence, even though no-one else in the world has tested the combination.

My own personal experience suggests that might be overly optimistic. I've been regularly updating my GTA04 to the latest kernel, updating the board file as necessary, and I have always had regressions. Sometimes minor (the display has a green tinge), sometimes major (the battery won't charge). Every patch that broke my device was tested by its developer, often on several devices. But none had quite the same set of components as mine and so nobody noticed until I did.

A consequence of the lack of a standard platform is that there are lots of different components available which different designers interconnect in lots of different ways resulting in an impossibly large test matrix. Based on my (limited) experience, I have very little confidence that a kernel that nobody has tested on my device will actually work on my device. And so the promise that devicetree offers seems particularly hollow to me. Of course it is entirely possible that my recent experience is not the norm for others nor for the future. We can but hope.

(Log in to post comments)

interrupt-map and loss of hierarchy
Posted Nov 18, 2013 23:07 UTC (Mon) by **scottwood** (subscriber, #74349) [Link]

A gpio-style version of interrupts would certainly be cleaner, but you can currently accomplish the same thing by pointing interrupt-parent at a separate node that exists only to be an interrupt-map, distributing each interrupt to its proper true parent.

As for the tree becoming flatter with a similar version for reg, that would actually let us move some of those misparented platform devices (e.g. PCI controllers that have their own controller registers that should sit under a parent bus, but also need to have PCI bus addresses in ranges) back to where they belong in the tree. The hierarchy would still be useful for human organizational purposes, and there ought to be some shorthand to indicate that a particular resource should use the actual tree parent (the obvious one is a phandle of zero, but gpio already uses that for the absence of the resource, which is also useful...). Perhaps a shift to named resources would be a good thing.

Device trees II: The harder parts
Posted Nov 19, 2013 2:34 UTC (Tue) by **dlang** (guest, #313) [Link]

> I've been regularly updating my GTA04 to the latest kernel, updating the board file as necessary, and I have always had regressions. Sometimes minor (the display has a green tinge), sometimes major (the battery won't charge). Every patch that broke my device was tested by its developer, often on several devices. But none had quite the same set of components as mine and so nobody noticed until I did.

My reaction to this is to remind you of the "bad old days" before the PCI bus on x86 systems where devices were not discoverable, and the rate of problems with kernel upgrades then.

I'd say that the types of things you are running into show major progress in comparison, and if reasonably sophisticated users can do that sort of testing of new kernels without having to hack board files it's a major advantage.

Device trees II: The harder parts
Posted Nov 19, 2013 7:24 UTC (Tue) by **glikely** (subscriber, #39601) [Link]

Regarding stability, when we met for an ARM Linux summit a few weeks ago in Edinburgh we discussed stability of the device tree bindings at length. While there are strong reasons why DT had to be unstable in the early days, we're going to stay enforcing stability, particularly for shipping products. ie. If a device boots in mainline for version x, then it is a bug to regress in version x+1.

Device trees II: The harder parts
Posted Nov 19, 2013 15:07 UTC (Tue) by **dougg** (subscriber, #1894) [Link]

For examples of bleeding edge kernels on ARM SoCs, DTs, uboot and more see: http://eewiki.net/display/linuxonarm/Home

Device trees II: The harder parts
Posted Nov 19, 2013 19:36 UTC (Tue) by **kleptog** (subscriber, #1183) [Link]

Given that these files describe the layout of the hardware, it seems it should be possible to take the file and create a circuit diagram from it. If only because visualising it might help people spot problems.

As for the hierarchy, perhaps it should be called DeviceGraph? :) In any case, having a very flat tree and using only hierarchies when it makes sense seems like an eminently practical approach.

But nice article.

Device trees II: The harder parts
Posted Nov 20, 2013 12:45 UTC (Wed) by **FrankT** (subscriber, #82936) [Link]

Another device tree compiler generating .png (or .dot) instead of .dtb?

Device trees II: The harder parts
Posted Nov 20, 2013 14:37 UTC (Wed) by **glikely** (subscriber, #39601) [Link]

Shouldn't be hard. Sounds like a weekend project. :-)

Device trees II: The harder parts

Posted Nov 19, 2013 21:30 UTC (Tue) by **dashesy** (guest, #74652) [Link]

I still do not know why protobuffers cannot be used for binary format (other than being newer technology), the interface is backward/forward compatible, clean and well defined.

Device trees II: The harder parts

Posted Nov 20, 2013 10:10 UTC (Wed) by **glikely** (subscriber, #39601) [Link]

That's the first I've heard of protobuffers. While that may be usable, the DTB binary format was defined a long time ago and to change it now would cause huge ABI breakage.

Device trees II: The harder parts

Posted Nov 19, 2013 22:40 UTC (Tue) by **dougg** (subscriber, #1894) [Link]

"In the GTA04, the OMAP3 SoC contains 16 banks of 32 GPIOs, which can reasonably be treated as a single block and will probably be numbered 0-191."

Please don't encourage that line of thinking. Use the SoC manufacturer's naming which historically has two parts: a letter and a number. As finer grain control over gpios is added to gpiolib you will find attributes that do depend on which bank you are in. Then it becomes unreasonable to treat them as a single block.

Atmel AT91 SoCs control their gpios with multiple instances of the PIO macrocell. And PIO stands for Parallel Input Output. So when you group gpios then a bank is a natural unit (most often 32 gpios, sometimes less).

We have been through flat space, numeric naming of AT91 gpios: first it was origin 32 (yes, WTF), then it was origin 0 and now they are transitioning to what they should have used in the first place. Naming counts, especially at the interface between hardware and software engineers; please use the naming conventions already established by hardware engineers and manufacturers.

Device trees II: The harder parts

Posted Nov 19, 2013 22:46 UTC (Tue) by **corbet** (editor, #1) [Link]

Happily, GPIO numbers are on their way out; the descriptor-based interface was merged for 3.13.

Device trees II: The harder parts

Posted Nov 25, 2013 3:08 UTC (Mon) by **alison** (subscriber, #63752) [Link]

>If you do something wrong you'll mostly get either:
> Error: /home/git/gta04-mainline/arch/arm/boot/dts/omap3-gta04.dts:407.12-13 syntax error
> FATAL ERROR: Unable to parse input tree
>or silent success as ARM maintainer Russell King recently observed (there are a couple of other error messages, but not many)

The best way to investigate dtc failures that I've found is to output the C-preprocessed hierarchy by invoking the compiler directly from CLI rather than using "make." Here

http://she-devel.com/makedts

is a bash script that runs the C preprocessor and dtc on a set of nested DTS files and produces a single human-readable output file with ASCII strings. A lot of mistakes in DTS will be obvious from inspection of the output: some node is at the wrong level of the hierarchy, or has an empty property that was expected to be populated. A working similar device-tree can be compared by running fdtdump (in scripts/dtc) on a binary and lining up the forward-compiled failure with the reverse-compiled working DTB.

Thanks, Neil Brown, for another exceptional pair of articles. I've read your "object orientation in the kernel" several times each.

Device trees II: The harder parts
Posted Nov 25, 2013 14:53 UTC (Mon) by **mathstuf** (subscriber, #69389) [Link]

Would a "dtsdebug" target which does this make sense? Seems useful enough to me to be supported directly.

Device trees II: The harder parts
Posted Nov 25, 2013 15:55 UTC (Mon) by **alison** (subscriber, #63752) [Link]

>Would a "dtsdebug" target which does this make sense?

A "verbose", chatty mode of dtc would be a start. A "dtsdebug" could also leave the intermediate files around on the disk and print a message about their contents rather than delete them silently, as dtc now does.

Device trees II: The harder parts
Posted Dec 2, 2013 8:56 UTC (Mon) by **zhang24xiao** (guest, #94260) [Link]

哇。。真的是果然hard part，我看不懂了 - -!

Copyright © 2013, Eklektix, Inc.
Comments and public postings are copyrighted by their creators.
Linux is a registered trademark of Linus Torvalds