Device trees I: Are we having fun yet?

November 12, 2013
This article was contributed by Neil Brown

Given that Linus Torvalds's biography is titled *Just for Fun* it seems appropriate that people working on Linux might expect to have some fun too. Yet when one hears the device tree concept (herein referred to as "devicetree") discussed at conferences or reads about it on mailing lists, there seems to be more agonizing and (generally constructive) arguments than enjoyment. The blessing, or the curse, of devicetree forming a stable ABI, how backward compatibility is a bane to one and a boon to another, what exactly it means to describe the hardware but not the behavior, and the joy that would flow if only we could have discoverable buses, are all topics which are thrown around with much fervor but not so much joyfulness (for indeed there are many buses which are not and never will be discoverable).

By contrast, my own experience with devicetree has produced a lot of fun (though it must be admitted that some was akin to the pleasure received when one stops a repeated cranial impact with a brick wall). This is the first in a two-part series on my experience with the devicetree mechanism.

The GTA04, the replacement motherboard for the Openmoko mobile phone, has struggled along until recently with no help from devicetree. As devicetree is apparently the way of the future and a necessary precondition for full upstream support, that had to change. The opportunity arose recently to pursue two ends in parallel: implementing sufficient devicetree support to boot and use the GTA04, and learning what this devicetree thing was really all about. Both have been achieved to a level of approximately 80% (suggesting that 20% of the required time has been spent so far). This experience was a lot of fun and, in order to savor that joy a little longer, the highlights are collected below to form a practical introduction to devicetree. Most of the examples are taken from the GTA04, partly because that is all I really know, but also because it has a variety of components sufficient to highlight several interesting devicetree issues.

It's all about the platform

While devicetree started life serving SPARC and PowerPC architectures, its use with ARM is the current focus of development. It is reasonable to ask: amid all this name dropping, where is x86? What do the Intel architectures use if they don't use devicetree?

The early IBM-PC triggered a boom in personal computer sales in part because it was so easy for other manufacturers to make copies or "clones". The specification was open and it went much further than just the choice of a particular CPU instruction set. It included known devices (like a keyboard) with a known controller at an known address in the I/O address space. It also included a "BIOS" (basic IO system) in ROM with well-defined entry points to achieve simple tasks like reading a block from the floppy disk, or writing a character onto the monitor. As long as cloners copied these standard interfaces, they could build a device that standard software could run on without needing to know it was running on something new.

As time progressed this "PC platform" evolved — largely driven by Intel creating hardware standards and Microsoft enforcing them through market dominance with its software. The BIOS was repeatedly extended, giving us a VBIOS (for video-drivers), an SMM BIOS (for system management) and ACPI, which does lots of different things.

While the PC platform has changed substantially over the years, there has always been the one platform with strong market forces to ensure compatibility. Operating systems can safely be written to assume the presence of whatever the platform requires, and to be able to probe for optional components with a confidence that, for example, probing for a mouse will not accidentally switch off the main power supply.

ARM has taken a very different approach to creating a platform, in that it hasn't bothered. ARM specifies the instruction set and CPU behavior (memory model, etc.) and provides VLSI designs (aka "IP blocks") which can be integrated into hardware, typically as parts of a system on a chip (SoC). Details beyond the CPU are largely left up to the individual manufacturer. There are no standard components, and no required BIOS.

ARM has another challenge which did not affect the early PC platform: power management is crucially important on many ARM-based devices, whereas it was largely uninteresting in the PC space. Part of the standard for the PC platform is a variety of "discoverable" buses which provide the bus-master with a mechanism for asking devices on the bus to identify and describe themselves. This leads to a very general solution for handling control and data flow. It is less clear that it leads to optimal solutions for power management. We will see an example of this below.

This lack of imposed standards in the ARM world provides for a great deal of flexibility (and product differentiation) for the hardware engineers but presents a real challenge for operating system engineers: you don't know what to expect or where it might be found, and just poking around is impractical and unlikely to be successful.

This is where devicetree comes in. A devicetree essentially describes a specific device. There is no BIOS component to the platform (which is likely a good thing given what kernel engineers seem to think of BIOS quality) but rather a list of devices or device-details that cannot be discovered by poking. This devicetree (encoded as a binary "dtb" file) can be stored in ROM on the system, or can be loaded from wherever the kernel is loaded. The theory is that when a system (e.g. motherboard) is created, a "dtb" can be created for it, and it will work with all future software releases. It is a nice theory...

Getting bored of board files?

Support for ARM hardware has been in the kernel a lot longer than devicetree has been used for it. The "old" approach (still widely used where devicetree support isn't complete) is the so-called "board" file. This is essentially a description of the platform written in C.

Each board file ends with a MACHINE description something like:

```
MACHINE_START(GTA04, "GTA04")
    /* Maintainer: Nikolaus Schaller - http://www.gta04.org */
    .atag_offset    =   0x100,
    .reserve        =   omap_reserve,
    .map_io         =   omap3_map_io,
    .init_irq       =   omap3_init_irq,
    .handle_irq     =   omap3_intc_handle_irq,
    .init_early     =   omap3_init_early,
    .init_machine   =   gta04_init,
    .init_late      =   omap3630_init_late,
    .timer          =   &omap3_secure_timer,
    .restart        =   omap_prcm_restart,
MACHINE_END
```

which, as you can see, is completely generic for "omap3" devices except for "`init_machine`".

`gta04_init()` contains a fairly ad-hoc collection of initializations, probably the most interesting being calls to `platform_add_devices()` and `omap_register_i2c_bus()`. These provide lists of device identifiers together with "`platform_data`" data structures which describe many of the various components on the GTA04 board and how they are connected together. (See this article for information on the platform device API).

So a board file identifies the SoC which the board is built around, identifies all the other components, and has various bits of glue code to make things work (like initializing GPIO lines).

A devicetree source file (dts) contains the first two elements (SoC identification and component configuration) but doesn't have the ad-hoc glue code. This is the first source of joy — much less clutter in the platform description as you cannot include code. This means, of course, that the generic code must be improved so that individual platforms don't need their own code and this is the second source of joy: more complete and coherent design in the generic code.

To see the difference, my git tree contains a board file for the GTA04 and a devicetree file that does nearly as much.

Let's start with a leaf

The following fragment is taken from somewhere several levels deep into the devicetree for the GTA04. It doesn't actually appear in `omap3-gta04.dts` file linked above, but rather in `twl4030.dtsi` which the former file includes.

```
charger: bci {
    compatible = "ti,twl4030-bci";
    interrupts = <9>, <2>;
    bci3v1-supply = <&vusb3v1>;
};
```

This fragment describes the battery charger ("`bci`" for "battery charger interface"). This is one component on a multi-function chip which serves as the PMIC (Power Management IC) — the twl4030 from Texas Instruments.

The "`charger:`" string is simply a label which allows this node to be referenced from elsewhere in the tree as we will see shortly. "`bci`" is the name of this node within its parent. The full name in the GTA04 is actually `/ocp/i2c@48070000/twl@48/bci`. The choice of names seems to have little practical effect providing they are unique among siblings. There are standards that are best followed, such as the "@NNN" suffix to give the device's address. Failing to follow these standards certainly results in non-compliance but does not seem to result in non-functionality.

The "`compatible`" field, rather than the node name, tells you (and Linux) what this device does. Every device can list one or more strings that it is "compatible" with. Similarly every driver in the kernel can list one or more strings that it is compatible with (via the "`driver.of_match_table`" field). The driver that is bound to a particular device is the one that appears to be most compatible. In many cases both the device and the driver will only have one "compatible" name, but it is reasonable to assume that over time some IP blocks will prove particularly successful, will be refined in backward-compatible ways, and so will benefit from this flexibility.

The "`interrupts`" field lists two interrupt numbers as the BCI can generate two different interrupts. These are interpreted in the context of the nearest interrupt controller. As the "`twl@48`" node, which is the parent of the BCI, declares itself to be an interrupt controller (by having the attribute "`interrupt-controller`" present), these interrupt numbers (9 and 2) are interrupts within the twl4030 itself (interrupt management is one of the multiple functions of the chip).

Since the BCI is an integral part of the twl4030 with these interrupt lines hardwired in, there seems little point in explicitly listing the numbers in the devicetree node. Surely they can simply be hardcoded in any "`ti,twl4030-bci`" compatible driver. The reason they are explicit is that, in principle at least, the BCI IP block in the twl4030 might be reused in some other packaging when TI makes some new PMIC, and in that case the interrupts might be wired differently, but the same driver should be able to be used. So it is good forward-planning to keep their definition separate.

Given this, it is somewhat surprising that the bci node (like all the other nodes in the "twl" parent) does not have a "`reg`" property. This property identifies the address at which the device can be found, in whatever address space the current node in the tree resides. For example the parent "`twl@48`" node contains:

```
reg = <0x48>;
```

meaning that the whole twl4030 multi-function device can be found at address 0x48 in the I2C bus that it is on. The twl4030 has a number of registers (accessed over the I2C bus) which are grouped into modules. The "`MAIN_CHARGE`" module is at address 0x74 and has 54 bytes of register space. So:

```
reg = <0x74 0x36>
```

in the `bci` node might seem appropriate (when given, the second number is the size of the register set). Unfortunately the BCI driver needs to access registers for other modules, such as `PM_MASTER`, `PM_RECEIVER`, and `INTERRUPTS`. This could be achieved with something like:

```
reg = <0x74 0x36> <0xB9 0x0E> <0x14 0x08> <0x00 0x14>
```

But those values, instead, are hardcoded in the driver. It's possible that the general solution just seemed to be more trouble than it is worth. While it is nice to imagine that each individual IP block can be treated as an individual device, independent from everything else, this is often not the case.

The final point of interest in this node is the attributes that *aren't* there.

Part of the role of the battery charger is to charge the backup battery (which, for example, keeps the real-time clock ticking when the main battery is removed). The parameters for charging the main battery can vary due to policy and circumstance so it is important that they are configured at run time, probably though a "sysfs" interface. However the parameters for charging the backup battery (which are a target voltage and a maximum current) depend primarily on the electrical characteristics of the battery (or super-capacitor) which is installed. So they are really part of the platform description.

Being part of the GTA04 platform, they should live in the "`omap3-gta04.dts`" devicetree file. However the above "`bci`" node lives in the "`twl4030.dtsi`" include file. So different fields for the one node might need to live in different files. This is where the "`charger:`" label comes in.

In the GTA04 file (`omap3-gta04.dts`) we have:

```
    &charger {
        ti,bb-uvolt = <3200000>;
        ti,bb-uamp = <150>;
    };
```

The "`&charger`" expands to the full name of the node with the label "`charger`", and this simply adds some attributes to that node. It can equally well replace attributes, so common defaults can be placed in an include file, and divergent values can be given in the main file. This ability to present the nodes in the tree in other than the obvious depth-first order is very valuable and quite widely used. I needed to follow several such indirections to piece together the full path of "`/ocp/i2c@48070000/twl@48/bci`".

It could be argued that specifying the charging parameters for the backup battery is specifying behavior (of the charger) rather than describing hardware. This is seen by some as contrary to the purpose of devicetree. This could be addressed by a simple sleight-of-hand. We simply create a new devicetree node that represents the battery.

```
    &backup-battery {
        ti,target-uvolts = <3200000>;
        ti,max-uamps = <150>;
    };
```

These are now physical properties of the battery: the voltage it produces when fully charged and the maximum charge current that it can comfortably accept. The driver could then inspect the properties of the battery (if one is present) and configure the charger accordingly.

This does raise a question of where in the devicetree the "backup battery" should be attached. It is conceptually connected to both the battery charger and the real time clock. As "devicetree" is in the form of a tree, the battery can only have one parent - which device node should that be? Hierarchical trees can be extremely powerful, but they don't always map cleanly to the real world.

From nodes to a tree

As we have already hinted at, devicetree nodes can contain child nodes as well as attributes. For example the "`twl@48`" node contains the "`bci`" node, as well as a real time clock, some power regulators, some GPIO pins, a watchdog timer, and more. The case where this is most obviously necessary is when the platform contains some devices on a bus that doesn't support discovery such as the I2C bus.

The OMAP3 contains three I2C buses. The GTA04 places just the twl4030 on one, leaves one unused, and includes quite a collection of sensors as well as the touchscreen and LED controllers on the third. There is no way that Linux could know anything about these unless they were described in a board file (passed to `omap_register_i2c_bus`), or in a devicetree.

Less obvious is the need for child nodes of the "twl@48" node. Certainly the driver for the twl4030 would know what component devices it contains. So the issue of having discoverable buses doesn't really apply here. There are at least two reasons that the member devices of the twl4030 need to be explicitly listed. One is that it allows platform-specific configuration to be added, as with the "`ti,bb-uvolt`" and "`ti,bb-uamp`" attributes for the battery charger (though if they were in a separate "battery" node, that would be less important).

The other is that other devices might need to make reference to the individual components. As mentioned the twl4030 contains some regulators and some GPIO pins. While the GTA04 doesn't use any of the GPIOs it does use the regulators. For example the WiFi chip is powered by "`VAUX4`" from the twl4030. The "MMC" driver which communicates with the WiFi chip needs to know about this so it can power up the WiFi interface to talk to it, then power it down while it isn't in use. This interconnection is specific to this particular platform and so needs to be explicit in the devicetree file. For that to be possible, each of the power regulators in the twl4030 must be listed as a child node. Given these obvious needs for many things to be listed, it makes a certain amount of sense to list everything.

One interesting twist in the GTA04 involves the module used for communicating with the GSM telephone network — the Option GTM601. This is primarily a USB device that is permanently connected to one of the USB ports on the OMAP3. As USB is considered to be a discoverable bus, there is no mechanism in devicetree to list the devices attached to a USB bus. However the GTM601 doesn't *only* use USB. It also has a DAI port (Digital Audio Interface) for bi-directional audio (which is connected to one of the McBSP audio ports on the OMAP3), and a "wake-up" line which is pulsed on an incoming phone call or text message.

While these three connections all relate to the one device, there is no way to describe this hardware in devicetree as a single unit. Rather the input "wake-up" signal and the audio channel are specified as separate devices, the USB interface is left to be discovered, and the relationship between the three is left for some application to just "know" about.

Both the audio data and the wake-up signal could be sent over the USB connection which would make it much easier for the operating system to treat it as a single device. However, as this approach would result in significantly higher energy usage, it is not used. Having a separate "wake-up" signal connected to an "always-on" GPIO bank on the OMAP3 allows the whole USB path to be completely powered off when idle. Having a separate DAI allows it to be connected directly to the audio codec/amplifier (in the magic twl4030 chip) so that the CPU is not involved in audio routing and can enter deep sleep during a phone call.

The second installment in this series will focus on some of the difficulties involved with the device tree abstraction and whether it will ever truly be possible to support system-independent kernel images on non-discoverable platforms.

(Log in to post comments)

Device trees I: Are we having fun yet?
Posted Nov 12, 2013 23:09 UTC (Tue) by **mwsealey** (subscriber, #71282) [Link]

There is a way to define USB devices on a USB bus in device trees, just as there are bindings to present PCI devices on a PCI bus which is intended to be "discovered" by the OS.

There definitely should be a little standardization around it - imagine a platform where USB ports are present but also parts are soldered to the PCB. They will *always* appear at the same places in the USB topology.

PCI device "compatible" properties are usually derived from the device and vendor ids (and sub ids) in the configuration space. USB has much the same thing. If you ever enabled dynamic network device naming on a modern Linux kernel, it is much the same gobbledegook string - v8081d1504u8181s8171.

If it is fixed to the board, then it should be fixed in the tree. And sometimes, hubs and other USB devices need configuring to tell them they're fixed to a board and their ports are not hot-pluggable.

For a dynamic firmware implementation that can generate a device-tree on the fly (more relevant on PowerPC than ARM, but there is no reason it cannot happen on ARM), it is still expected that it would fill in /chosen properly in the final device tree for things like it's boot device. In this case, a USB device has to be present in the tree to be referenced by phandle.

For WiFi and suchlike, it is acceptable that a 'pretend' rfkill node might be created to control the devices and follow the Linux subsystem, but this would go against the OS-neutrality of device trees. In this case, the solution is simple. If it is soldered down or shipped as a fixed option in a consumer device (such as in a Netbook, under a door you may remove but wouldn't want to remove the warranty sticker to do so) then it should be listed.

Device trees I: Are we having fun yet?
Posted Nov 13, 2013 6:03 UTC (Wed) by **jcm** (subscriber, #18262) [Link]

There certainly will be more standardization in the ARM platform going forward, in particular in the server space. You saw the public announcement that ACPI is now part of the UEFI forum. I would be surprised if that were the last such announcement over the coming months.

Device trees I: Are we having fun yet?
Posted Nov 13, 2013 6:26 UTC (Wed) by **olof** (subscriber, #11729) [Link]

ACPI and UEFI doesn't magically solve the ARM server problems, and it's troubling that people just walk around that topic thinking it will. They just give you yet another mechanism to communicate the mess that the vendors create.

The x86 space has been more organized because Microsoft forced people to comply with the standard specs (and a product wouldn't ship, plain and simple, if it didn't boot an unmodified Windows install). That's not true for these ARM servers since Linux will be the primary target for many of them -- the same strict requirements aren't there.

On the other hand, maybe, just maybe we can avoid the mess of SMM. It's definitely a mixed blessing to have (or require) that level of under-the-covers workaround for compliance, since we all know just how messy and buggy firmware can be. On the other hand, it avoids the same mess to crawl into the kernel.

Device trees I: Are we having fun yet?
Posted Nov 13, 2013 6:57 UTC (Wed) by **jcm** (subscriber, #18262) [Link]

Olof, I agree that ACPI doesn't magically solve the problems. But what I mean is, I expect to see such platform standardization in the ARM server space. I would be /very/ surprised if there aren't movements in that regard. As far as SMM, sadly I raised this 3 years ago that I thought vendors would abuse TrustZone (and in particular create their own apps running on the TEE) to emulate SMM on AArch64, and that's roughly what I see happening. I've made recommendations to mitigate the impact, which I see being lesser because: a). You don't stop all CPUs in the system. b). Most of this crap can be handled by management cores in the designs that I have reviewed so far.

Device trees I: Are we having fun yet?
Posted Nov 13, 2013 7:39 UTC (Wed) by **olof** (subscriber, #11729) [Link]

Oh, I have hope that the vendors that have had enough planning to involve you in their designs will come out with something fairly sane, even though some of the recent patch sets can introduce a good amount of doubt.

My greater concern is the second-tier vendors, or the mass-market vendors that will do the bare minimum (and not quite know what they're doing, or at least not seeing the bigger picture), etc. It'll take quite a bit of education of them to sort things out right, and it only takes a few of them going to market anyway to make the kernel side of thing very interesting and possibly messy. It'll be interesting to see how it plays out.

Device trees I: Are we having fun yet?
Posted Nov 14, 2013 9:23 UTC (Thu) by **deepfire** (guest, #26138) [Link]

> The x86 space has been more organized because Microsoft forced people to
> comply with the standard specs (and a product wouldn't ship, plain and
> simple, if it didn't boot an unmodified Windows install).

I find "boot an unmodified Windows install" being a rather weak substitute for standards compliance.

Device trees I: Are we having fun yet?
Posted Nov 14, 2013 14:24 UTC (Thu) by **raven667** (subscriber, #5198) [Link]

Well it isn't just being able to boot, MS has pushed for several device standards such as for webcams so they didn't have to deal with buggy crashy vendor drivers making them look bad.

Device trees I: Are we having fun yet?
Posted Nov 26, 2013 17:18 UTC (Tue) by **nye** (guest, #51576) [Link]

On a related note, anyone know why there's no such thing as a standardised ethernet class driver?

Device trees I: Are we having fun yet?
Posted Nov 15, 2013 17:52 UTC (Fri) by **drag** (subscriber, #31333) [Link]

> I find "boot an unmodified Windows install" being a rather weak substitute for standards compliance.

Yet it's the most effective standards compliance mechanism that has ever existed as far as hardware goes.

Device trees I: Are we having fun yet?
Posted Nov 13, 2013 13:33 UTC (Wed) by **arnd** (subscriber, #8866) [Link]

A lot of the arm64 "server" components that people are talking about are not really PC-style servers as we know them from x86 but rather system-on-chip designs more akin to today's embedded 32-bit products. There is no real infrastructure in the kernel to deal with these using ACPI, nor is that likely to get added any time soon.

There is a real danger of people trying to use ACPI on these systems anyway (as shown by some patches posted for X-Gene), which implies that they come up with random out-of-tree and out-of-spec hacks that they cannot rectify before shipping products to end-users. Unlike for dtb files (which can be shipped alongside a kernel), there is not really a backwards-compatible way to replace an ACPI BIOS, so a system like that is either stuck with a custom kernel (i.e. the thing we're all trying to avoid here) or with an ugly per-product workaround (i.e. a board file) in the kernel.

We found out the hard way with DT that it's not really possible to have a stable ABI while simultaneously trying to figure out how to describe things in the first place. ACPI on ARM today is in a worse position than DT on ARM was three years ago when we could sufficiently describe all PowerPC SoCs, and there is no way for ACPI to not be an ABI.

Device trees I: Are we having fun yet?
Posted Nov 13, 2013 10:23 UTC (Wed) by **fidelleon** (subscriber, #91950) [Link]

MANY thanks for this article, after looking at a lot of places I wasn't able to find a *right* description of what a devicetree is (yep, I was unaware of their meaning).

Device trees I: Are we having fun yet?
Posted Nov 14, 2013 14:43 UTC (Thu) by **rwmj** (guest, #5474) [Link]

I'd love someone to explain why it is in practice that I see *.dtb files shipped with the kernel, rather than with the hardware or bootloader. AIUI device tree describes the hardware to the kernel, and so it's something you should get with the hardware (or perhaps with uboot).

Device trees I: Are we having fun yet?
Posted Nov 14, 2013 20:16 UTC (Thu) by **mjg59** (subscriber, #23239) [Link]

Because there's no fixed device tree ABI, and if you ship a DTB in flash there's no guarantee that you'll be able to boot newer kernels. It's actually worse than that, in that vendor kernels and mainline are often ABI-incompatible when it comes to Device Tree, so if a board vendor ships with the SoC vendor kernel you wouldn't be able to boot a mainline kernel.

Device trees I: Are we having fun yet?
Posted Nov 14, 2013 21:24 UTC (Thu) by **scottwood** (subscriber, #74349) [Link]

Not having a fixed device tree ABI breaks use cases such as a boot loader fixing up portions based on run time information, or a hypervisor using it to tell the guest OS what devices the VM has been configured with, or sharing the tree with other OSes (we do all of these on Freescale PPC, and on some ARM boards the device tree is shared with U-Boot -- not just to pass to the kernel, but for U-Boot to learn about the hardware). A fixed ABI is how it works with actual Open Firmware, and is how ACPI works, etc. Likewise with PCI -- it works because there's a standard, that isn't subject to revision as part of Linux patch acceptance.

Granted, it's harder to standardize things well when the hardware being described is changing so rapidly, but by treating the device tree as a Linux implementation detail rather than an ABI, you take away the incentive to get it (reasonably close to) right the first time. You could still have a separate Linux-internal tree (we already do -- struct device and friends) that encodes Linux concepts and/or fixes quirks in the hardware tree, which is generated from the hardware tree plus quirk code in the kernel, and possibly some supplementary external data file which *is* a Linux implementation detail and not meant for sharing with other components. In extreme cases the "quirk and supplementary external data" could be an entire replacement tree, if the standard one is hopeless for anything other than identifying which replacement tree to use, but most cases should not be that bad.

Moving the hardware tree outside the kernel source would send a clear message that it is not a Linux implementation detail (and thus compatibility between the trees and OSes including but not limited to Linux needs to be considered), and that things which are need to go somewhere else.

Also note that the dts sometimes contains implementation details of the bootloader -- it may make assumptions about how the loader sets up the address map, or it may make assumptions about what things the loader will fill in before passing the tree to the kernel. Both of these are true on Freescale PPC. In those cases ideally the trees would be stored in the bootloader tree.

Device trees I: Are we having fun yet?
Posted Nov 14, 2013 22:14 UTC (Thu) by **mathstuf** (subscriber, #69389) [Link]

> if a board vendor ships with the SoC vendor kernel you wouldn't be able to boot a mainline kernel.

Sounds like a job for GPL enforcement ;) .

Device trees I: Are we having fun yet?
Posted Nov 14, 2013 22:48 UTC (Thu) by **mjg59** (subscriber, #23239) [Link]

Why? GPL doesn't require that you be able to boot the mainline kernel.

Device trees I: Are we having fun yet?
Posted Nov 14, 2013 22:52 UTC (Thu) by **mathstuf** (subscriber, #69389) [Link]

No, but it can get you the source that you would need to use a custom kernel.

Device trees I: Are we having fun yet?

Posted Nov 15, 2013 0:06 UTC (Fri) by **mjg59** (subscriber, #23239) [Link]

Getting the vendor tree typically isn't a problem. Finding a vendor tree that's up to date with security fixes, on the other hand...