Transactional Device Tree & Overlays
Making Reconfigurable Hardware Work

Pantelis Antoniou <pantelis.antoniou@konsulko.com>
Describing Hardware

- Platforms get increasingly more complex.
- ARM based systems are even more complex than ever.
- Platform data not cutting it anymore.
- Enter Device Tree.
- Originally on PowerPC, now on ARM+everything else besides x86.
- X86 left out? Maybe not.
Device Tree (vanilla flavor)

- According to ePAPR “Describes system hardware”
- "The Device Tree is a data structure for describing hardware. Rather than hard coding every detail of a device into an operating system, many aspect of the hardware can be described in a data structure that is passed to the operating system at boot time."
- Tree structure
- Describes information that can't be dynamically determined by running software
Device Tree complaining

- A popular pass-time.
  - “And for whatever your part is in the BBB device tree mess, I hope sincerely that you someday acquire enough wisdom to feel ashamed of what you did. Really. Okay, I flamed.”

- Nuggets of truth
  - One more language to learn (dts) and first timers find it complex.
  - Purely data driven, make it hard to wrap around old platform data + callback uses.
  - No syntax checks at compile time.
  - **Not every hardware piece can be statically defined at boot time.**
Bare Beaglebones

- BeagleBone is a low-cost, community-supported development platform for developers and hobbyists.
- CPU: AM335x 1GHz ARM® Cortex-A8
- Memory: 512MB
- A lot of standard interfaces (USB Host/Client, Ethernet, HDMI)
- Build your own stuff and connect them using the 2x46 pin connectors (passthrough) – capes.
- Lots of capes already available.
Beaglebone and the Device Tree

- Capes are identified using an onboard EEPROM.
- No way to support this scheme using static Device Tree.
- Trying to do Device Tree blob mangling in the bootloader is quite difficult (and it doesn't work with stacked capes).
- A method to dynamically alter the live Device Tree according to the probed cape required.
- Opening a can of worms...
Intermission

- Beaglebone's capes are not unique.
- Raspberry Pi (HAT specification).
- FPGAs can instantiate different peripherals according to the bitstream loaded.
- The view that hardware is something static is outdated. Hardware is software nowadays.
- Friends don't let friends (hardware hackers) use Arduino – but Linux is just too hard for mostly hardware hackers (write a kernel driver to interface to a LED?).
Going down in flames

31 Oct 2012: “Capebus; a bus for SoCs using simple expansion connectors”

Not a bus!

Booing from the peanut gallery.

They were right.

Back to the drawing board.
CONFIG_OF_DYNAMIC

- Allows modification of the Live Device Tree at runtime.
- Not very widely used until now – only on Power.
- Destructive editing of the live tree
  - Non atomic
  - Changes cannot be reverted
- No connection to the bus driver model; changes to the live tree do not get reflected.
- Part of the puzzle, but not enough as it was.
Part 1: Reworking OF_DYNAMIC

- /proc → /sys (gcl)
- struct device_node now a kobj (gcl)
- drivers/of/dynamic.c
- Semantics of the of_reconfig notifiers have changed.
- Major new user is dt selftests. Test case data dynamically inserted (/me nags about how).
- Already accepted in mainline (3.17)
Part 2: Dynamic Resolution

/* foo.dts */
/
{
  bar = <&FOO>;  /* compiles to bar = <1>; */
  FOO: foo { };  /* dtc assigns value of 1 to foo phandle */
};

/* qux.dts */
/
{
  qux = <&BAZ>;  /* compiles to qux = <1>; */
  quux = <&FOO>;  /* ??? Only possible to resolve on runtime */
  BAZ: baz { };  /* dtc assigns value of 1 to baz phandle */
};
Resolving phandles

- Phandles are pointers to other parts in the tree. For example pinmuxing, interrupt-parent etc.
- Phandles are internally represented by a single 32 scalar value and are assigned by the DTC compiler when compiling
- Extension to the DTC compiler required, patchset already in v2, minor rework is required.
- "dtc: Dynamic symbols & fixup support (v2)"
Changes made to the DT Compiler

+ ABSOLUTELY NO CHANGES TO THE DTB FORMAT.

+ `-@` command line option global enable.

+ Generates extra nodes in the root (`__symbols__`, `__fixups__`, `__local_fixups__`) containing resolution data.

+ `/plugin/` marks a device tree fragment/object (controls generation of `__fixups__` and `__local_fixups__` nodes).

+ To perform resolution the base tree needs to be compiled using the `-@` option and causes generation of `__symbols__` node only.
Compiling foo.dts (base tree)

```bash
$ dtc -O dtb -o foo.dtb -b 0 -@ foo.dts && fdtdump foo.dtb

/ {
    bar = <0x00000001>;
    foo {
        linux,phandle = <0x00000001>;
        phandle = <0x00000001>;
    }
    __symbols__ {
        FOO = "/foo";
    }
};
```
$ dtc -O dtb -o qux.dtbo -b 0 -@ qux.dts & & fdtdump qux.dtbo

/ {
    qux = <0x00000001>;
    quux = <0xdeadbeef>;
    baz {
        linux,phandle = <0x00000001>;
        phandle = <0x00000001>;
    }
};

__symbols__ { BAZ = "/baz"; }

__fixups__ { FOO = "/:quux:0"; }

__local_fixups__ { fixup = "/:quux:0"; }

};
How the resolver works

- Get the max device tree phandle value from the live tree + 1.
- Adjust all the local phandles of the tree to resolve by that amount.
- Using the __local__fixups__ node information adjust all local references by the same amount.
- For each property in the __fixups__ node locate the node it references in the live tree. This is the label used to tag the node.
- Retrieve the phandle of the target of the fixup.
- For each fixup in the property locate the node:property:offset location and replace it with the phandle value.
Part 3: Changesets/Transactions

- A Device Tree changeset is a method which allows us to apply a set of changes to the live tree.
- Either the full set of changes apply or none at all.
- Only after a changeset is applied notifiers are fired; that way the receivers only see coherent live tree states.
- A changeset can be reverted at any time.
- Part of mainline as of 3.17.
Changesets in kernel API

+ Issue `of_changeset_init()` to prepare the changeset.
+ Perform your changes using `of_changeset_{attach_node|detach_node|add_property|remove_property|update_property}()`()
+ Lock the tree by taking the `of_mutex`;
+ Apply the changeset using `of_changeset_apply()`;
+ Unlock the tree by releasing `of_mutex`.
+ To revert everything `of_changeset_revert()`;
Part 4: Device Tree Overlays

- A method to dynamically insert a device tree fragment to a live tree and effect change.

- Simplest example: turn the status property of a device node from “disabled” to “okay” and have the device corresponding to that node be created.

- Low level interface; A generic configfs manager is provided, but for platforms like the beaglebone a more elaborate manager may be required.

- Good enough for hardware hackers – no reboots required (if all the platform device removal bugs are fixed).

- 7th version of the patchset was posted, 8th will be forthcoming ELCE14/ Plumbers discussion.
Device Tree Overlay format

`/plugin/;

/ {
    /* set of per-platform overlay manager properties */
    fragment@0 {
        target = <&target-label>; /* or target-path */
        __overlay__ {
            /* contents of the overlay */
        }
    }
    fragment@1 {
        /* second overlay fragment... */
    }
};
`
Device Tree Overlay in kernel API

- Get your device tree overlay blob in memory – using a call to `request_firmware()` call, or linking with the blob is fine.

- Use `of_fdt_unflatten_tree()` to convert to live tree format.

- Call `of_resolve_phandles()` to perform resolution.

- Call `of_overlay_create()` to create & apply the overlay.

- Call `of_overlay_destroy()` to remove and destroy the overlay. Note that removing overlapping overlays must be removed in reverse sequence.
Device Overlay ConfigFS manager

- Generic Overlay manager.
- Very simple file based interface
  
  ```
  # mkdir /config/device-tree/overlays/test
  # cp OVERLAY.dtbo \n  /config/device-tree/overlays/test/dtbo
  # rmdir /config/device-tree/overlays/test
  ```
- Requires a binary configfs attribute patch
- Patches reviewed, and will be reposted.
In the pipeline

- Overlays based FPGA manager by Alan Tull
- Beaglebone cape manager
- Board revision run-time detection - using a single kernel and device tree blob for different revisions of a board.
- Your ideas?
Thank you for listening