Solving Device Tree Issues

Use of device tree is mandatory for all new ARM systems. But the implementation of device tree has lagged behind the mandate. The first priority has been correct function. Lower priorities include device tree validation and facilities to debug device tree problems and errors. This talk will focus on the status of debug facilities, how to debug device tree issues, and debug tips and tricks. Suggestions will be provided to driver writers for how to implement drivers to ease troubleshooting.

Frank Rowand, Sony Mobile Communications  
June 4, 2015  
150603_1607
CAUTION

The material covered in this presentation is kernel version specific

Most information describes 3.16 - 4.1

In cases where arch specific code is involved, there will be a bias to looking at arch/arm/
Any slides with 'skip' in the upper right hand corner will be skipped over in my talk. They contain information that will be useful when the slides are used for reference.
Obligatory Outline

Device tree concepts
DT data life cycle
Comparing Device Tree Objects
  - DT at different points in the life cycle
  - the magic of dtdiff
Device Creation, Driver Binding
  - dyndbg
  - dt_stat
  - dtdiff
Why this talk?

Debugging device tree problems is not easy.
Why this talk?

Debugging device tree problems is not easy.
- tools do not exist or are not sufficient
- error and warning message may not be available or helpful
- state data is not easy to access and combine
- debug process is not well documented
- add your own reason here
Why this talk?

At the end of this talk, you will know how to:
- debug some common device tree problems
- access data to support the debug process

Debugging some types of device tree problems will be easier.
Chapter 1

Device tree concepts
why device tree?

A device tree describes hardware that can not be located by probing.
what is device tree?

“A device tree is a tree data structure with nodes that describe the devices in a system. Each node has property/value pairs that describe the characteristics of the device being represented.” (ePAPR v1.1)
Key vocabulary

node
- the tree structure
- contain properties and other nodes

property
- contains data value(s) providing information about a node
'compatible' property has pre-defined use

node '/':
  - will be used to match a machine_desc entry

other nodes:
  - will be used to match a driver
model = "Qualcomm APQ8074 Dragonboard";
compatible = "qcom,apq8074-dragonboard";
interrupt-parent = <&intc>;

soc: soc {
    compatible = "simple-bus";

    intc: interrupt-controller@f90000000 {
        compatible = "qcom,msm-qgic2";
        interrupt-controller;
        reg = <0xf9000000 0x1000>,
             <0xf9002000 0x1000>;
    };

    console: serial@f991e000 {
        compatible = "qcom,msm_uartdm-v1.4", "qcom,msm_uartdm";
        reg = <0xf991e000 0x1000>;
        interrupts = <0 108 0x0>;
    };
};
.dts - Node - a chunk of HW

/ {
  model = "Qualcomm APQ8074 Dragonboard";
  compatible = "qcom,apq8074-dragonboard";
  interrupt-parent = <&intc>;

  soc: soc {
    compatible = "simple-bus";

    intc: interrupt-controller@f90000000 {
      compatible = "qcom,msm-qgic2";
      interrupt-controller;
      reg = <0xf9000000 0x1000>,
           <0xf9002000 0x1000>;
    };

    console: serial@f991e000 {
      compatible = "qcom,msm-uartdm-v1.4", "qcom,msm-uartdm";
      reg = <0xf991e000 0x1000>;
      interrupts = <0 108 0x0>;
    };
  };

  concept: variable path
.dts - Property – HW attribute
/
{
    model = "Qualcomm APQ8074 Dragonboard";
    compatible = "qcom,apq8074-dragonboard";
    interrupt-parent = <&intc>;

    soc: soc {
        compatible = "simple-bus";

        intc: interrupt-controller@f90000000 {
            compatible = "qcom,msm-qgic2";
            interrupt-controller;
            reg = <0xf90000000 0x1000>,
                <0xf9002000 0x1000>; }

        console: serial@f991e000 {
            compatible = "qcom,msm-uartdm-v1.4", "qcom,msm-uartdm";
            reg = <0xf991e000 0x1000>;
            interrupts = <0 108 0x0>; }
    }
};

concept: variable name
.dts - Value – HW attribute data

/ {
    model = "Qualcomm APQ8074 Dragonboard";
    compatible = "qcom,apq8074-dragonboard";
    interrupt-parent = <&intc>;

    soc: soc {
        compatible = "simple-bus";

        intc: interrupt-controller@f90000000 {
            compatible = "qcom,msm-qgic2";
            interrupt-controller;
            reg = <0xf9000000 0x1000>,
                 <0xf9002000 0x1000>;
        }

        console: serial@f991e000 {
            compatible = "qcom,msm-uartdm-v1.4", "qcom,msm-uartdm";
            reg = <0xf991e000 0x1000>;
            interrupts = <0 108 0x0>;
        }
    }
};

concept: variable value
.dts - Reference

Thomas Pettazzoni's ELC 2014 talk
“Device Tree For Dummies” is an excellent introduction to

- device tree source
- boot loader mechanisms
- much more!

https://www.youtube.com/watch?v=uzBwHFfJ0vU
DT data life cycle

(source) .dts ➔ (compiler) dtc ➔ (binary blob) .dtb

boot loader: dtb'' ➔ dtb'

boot image: [ dtb' ]

memory: dtb'' FDT
(flattened device tree)

EDT
(expanded device tree)

linux kernel

vmlinux
DT data life cycle

dtc creates .dtb from .dts
boot loader copies .dtb into memory FDT
Linux kernel reads FDT, creates Expanded DT

.dtb may be modified by build process boot loader

FDT and Expanded DT may be modified by Linux kernel
DT data life cycle

(source) .dts → (compiler) dtc → (binary blob) .dtb → [ overlay ] .dtb

boot loader: dtb'
image:
dtb''

memory:
dtb'' FDT (flattened device tree)
EDT (expanded device tree)

boot image:
vmlinux [ dtb' ]

linux kernel
DT data life cycle (overlay)

dtc creates .dtb from .dts and .dtsi

Linux kernel reads overlay, modifies Expanded DT

Overlay .dtb may be modified by ???

Expanded DT may be modified by Linux kernel

Overlay architecture and implementation still under development.
Chapter 2
Comparing Device Tree Objects
Skipping forward about 55 slides

The stuff I am skipping is valuable and interesting. But I had to choose a big section to leave out due to lack of time...
Suspicions

When debugging

I do not trust anything
I suspect everything
Suspicion

When debugging

I do not trust anything
I suspect everything

How do I know if my Expanded Device Tree matches what is in my device tree source?
Suspicion

When debugging

I do not trust anything
I suspect everything

How do I know if my Expanded Device Tree matches what is in my device tree source?

If I wanted the bootloader to adjust the .dtb, how do I verify the changes?
**Compare DT source to EDT**

```
$ dtdiff qcom-apq8074-dragonboard.dts base | wc -l
 282

$ dtdiff qcom-apq8074-dragonboard.dts base \  
    | grep "^+"  | wc -l
  39

$ dtdiff qcom-apq8074-dragonboard.dts base \  
    | grep "^-"  | wc -l
  32
```

diff host device tree source with target EDT
Compare DT source to EDT

$ dtdiff qcom-apq8074-dragonboard.dts base | wc -l
282

That is too big a diff to fit on one slide.

I will instead diff at different points in the DT data life cycle to see if I can create smaller diff results that will be easier to examine and understand.
Can I trust dtc?

```bash
$ dtdiff qcom-apq8074-dragonboard.dts \
    qcom-apq8074-dragonboard.dtb
@@ -13,2 +13,2 @@
-       clock-controller {
+       kraitcc: clock-controller {
@@ -30,7 +30,7 @@
-               cpu@0 {
+               cpu0: cpu@0 {

... and many more ...

diff host device tree source with host .dtb
Can I trust dtc?

```bash
$ dtdiff qcom-apq8074-dragonboard.dts \\
    qcom-apq8074-dragonboard.dtb \\
    | grep "^+" | wc -l
31

$ dtdiff qcom-apq8074-dragonboard.dts \\
    qcom-apq8074-dragonboard.dtb \\
    | grep "^-" | wc -l
31
```

Same number of lines added and deleted in diff.

Visual inspection verifies that all changes are removing a label from a node.
Can I trust the bootloader?

$ dtdiff qcom-apq8074-dragonboard.dtb dragon_sys_fdt
@@ -11,2 +11,5 @@
   chosen { 
+           bootargs = "console=ttymSM0,115200,n8 and 
+           linux,initrd-end = <0x2918456>;
+           linux,initrd-start = <0x2000000>;
   }
@@ -147,5 +150,5 @@
   memory { 
           device_type = "memory";
-           reg = <0x0 0x0>;
+           reg = <0x0 0x40000000 0x40000000 0x40000000 0x40000000 0x40000000 0x40000000>
   }

diff host .dtb with target FDT
Can I trust Linux?

```bash
$ dtdiff dragon_sys_fdt base
@@ -7,2 +7,6 @@
+       __local_fixups__ {
+       }
+    
+    aliases {
+        testcase-alias = "./testcase-data";
+    }
```

diff target FDT with target EDT

note: I removed the /testcase-data node from
the EDT for each diff with target EDT
Full Disclosure

The content of the previous diffs are modified so they will fit on slides.

I removed the /testcase-data node from the target EDT for each diff with the target EDT.

The /testcase-data nodes will be present on the target if CONFIG_OF_UNITTEST=y.
Resources

See the entry for this talk on the “Resources” slide for more details on how to access the DT data at various stages of the build and boot process.

FDT and EDT are from the target system
  FDT is /sys/firmware/fdt
  EDT is /proc/device-tree
  (currently a link to /sys/firmware/devicetree/base)
Takeaway

A diff tool exists to examine how the device tree data is modified in the build, boot loader, and boot process.
dtdiff

Wait a minute!!!

What is this tool?
Where do I get it?
Why don't I just use 'diff'?
dtiff - What is this tool?

dtiff compares device trees in various formats

- source (.dts and the .dtsi includes)
- dtb (binary blob)
- file system tree

For one source device tree

- pre-process includes and create resulting source (that is, converts .dts and included .dtsi into a single .dts)
dtdiff - Where do I get it?

It might be packaged for your distribution:

device-tree-compiler
dtc

The maintainer's git repo:

  git clone git://git.kernel.org/pub/scm/utils/dtc/dtc.git

These locations also contain the dtc compiler.
Note that the Linux kernel build process uses its own version of the dtc compiler from the Linux kernel source tree, built as:

  ${KBUILD_OUTPUT}/scripts/dtc/dtc
**dtdiff - Where do I get it?**

**WARNING:** the current version does not properly handle `#include` and `/include/` for `.dts` and `.dtsi` files in the normal locations in the Linux kernel source tree.

**Work In Progress patch** to fix this and to add the pre-process single `.dts` file feature is at:

http://elinux.org/Device_Tree_frowand
http://elinux.org/images/a/a3/Dtdiff_add_cpp.patch
dtendiff - Why don't I just use 'diff'?

Device tree .dts and .dtsi source files are ascii, similar to C .c and .h files. You can use diff!

Device tree .dtb files are binary files. diff does not work on binary files.

Device tree file system trees are nested directories containing a mix of ascii and binary files. You can normally use diff on ascii files but DT fs trees are produced from /proc/device-tree and are not '\n' terminated, so diff treats them as binary files (use diff -a or --text.)
dtdiff - Why don't I just use 'diff'?

real-life answer: Because dtdiff is
   - so much better than diff
   - easier to use than diff

Except in the rare cases where it hides information that you need!
dtdiff - Why don't I just use 'diff'?  

The answer to this question is going to be a long meandering journey through many slides. I may skip over many of those slides today but suggest you read them later at your leisure.
dt diff meander - how C compiles

$ cat v1/dup.c
#include <stdio.h>
const int model = 1;
main() {
    printf("model is: %d
", model);
};
$ gcc v1/dup.c
$ ./a.out
model is: 1
$ diff -u v1/dup.c v2/dup.c
--- v1/dup.c
+++ v2/dup.c
@@ -1,6 +1,7 @@
#include <stdio.h>

const int model = 1;
+const int model = 2;

main() {
    printf("model is: %d\n", model);
The C language does not allow redefinition of a variable.
dtdiff meander - how dtc compiles

1) **Compile** from v1/test.dts to v1/test.dtb

2) **De-compile** from v1/test.dtb to v1/dcmm.dts

```bash
$ dtc -I dts -O dtb -o v1/test.dtb v1/test.dts

$ dtc -I dtb -O dts -o v1/dcmm.dts v1/test.dtb
```
dTdiff meander - how dtc compiles

$ cat v1/test.dts
/dts-v1/;
/
  /
  model = "model_1";
  compatible = "test";
  c {
    model = "model_c";
  };
/;
/
  /
  model = "model_3";
  compatible = "test";
  a {
    model = "model_a";
  };
/;
dtdiff meander - how dtc compiles

$ cat v1/dcmp.dts
/dts-v1/;
/
{
    model = "model_3";
    compatible = "test";

    c {
        model = "model_c";
    };

    a {
        model = "model_a";
    };
};
dtdiff meander - how dtc compiles

$ dtdiff v1/test.dts v1/test.dtb
$ dtdiff v1/test.dts v1/dcmp.dts

dtdiff says all 3 objects are the same
- v1/test.dts
- v1/test.dtb
- v1/dcmp.dts
dtdiff meander - how dtc compiles

But diff knows the 'truth':

```bash
$ diff -u v1/test.dts v1/dcmp.dts
--- v1/test.dts
+++ v1/dcmp.dts
@@ -1,17 +1,12 @@

diff original .dts with decompiled .dtb
shows the transformations by the dtc compiler
dtdiff meander - how dtc compiles

/dts-v1/;
/
{
  model = "model_1";             <-- removes since redefined
+  model = "model_3";           <-- moves to top of node
    compatible = "test";
    c {
      model = "model_c";
    };
-};
-
-/
{
  model = "model_3";            <-- collapses duplicate nodes
  compatible = "test";        <-- move to top of node
  a {
    model = "model_a";        <-- move to top of node and
      deletes 1st as redefined
When a property at a given path occurs multiple times, the earlier values are discarded and the latest value encountered is used.
C:

Redefinition of a variable initialization value is likely to be an error
dtdiff meander - C vs dtc

dtc:

.dtsi source file describes a HW object which may be used in many ways
When .dts includes a .dtsi, it may need to change the general HW description because of how it is used in the current system

Redefinition of properties is a critical and common pattern in DT source files
dtdiff meander - C vs dtc

Redefinition of properties in DT source files means the mental model for comparing two device trees is often different than for comparing the source files for two C programs.
Example:

reverse the order of the two instances of node "/"
$ diff -u v1/test.dts v2/test.dts
--- v1/test.dts
+++ v2/test.dts
@@ -1,19 +1,19 @@

Text diff is cluttered - hard to determine impact
(see next slide).
dtdiff meander - node/prop order

@@ -1,19 +1,19 @@
 /dts-v1/;
 /
 { 
 -  model = "model_1";
 +  model = "model_3";
 +    compatible = "test";
 
 -  c {
 -    model = "model_c";
 +  a {
 +    model = "model_a";
 +    }
 +  }
 +  }
 /
 { 
 -  model = "model_3";
 +  model = "model_1";
 +    compatible = "test";
 
 -  a {
 -    model = "model_a";
 +  c {
 +    model = "model_c";
 +    }
 +  }
 +  };


dtiff meander - node/prop order

diff of decompiled .dtb files

result is less cluttered, easier to understand

(see next slide).
$ diff -u \
>    <(dtc -I dtb -O dts v1/test.dtb) \
>    <(dtc -I dtb -O dts v2/test.dtb)
--- /dev/fd/63
+++ /dev/fd/62
@@ -1,14 +1,14 @@
 /dts-v1/;
 
 / {
-   model = "model_3";
+   model = "model_1";
+       compatible = "test";

-   c {
-       model = "model_c";
-   };
- 
-   a {
-       model = "model_a";
-   };
+   c {
+       model = "model_c";
+   };
+ 
+};
dtdiff meander - node/prop order

diff of decompiled .dtb files

adds a sort to the decompile step

result is much less cluttered, easier to understand

(see next slide).
$ diff -u \\
>   <(dtc -I dtb -O dts -s v1/test.dtb) \\
>   <(dtc -I dtb -O dts -s v2/test.dtb)
--- /dev/fd/63
+++ /dev/fd/62
@@ -2,7 +2,7 @@

/ {
    compatible = "test";
-   model = "model_3";
+   model = "model_1";

    a {
        model = "model_a";
dtdiff meander - node/prop order

dtdiff adds a sort to the decompile step

same result as previous 'diff'
result is much less cluttered,
easier to understand

(see next slide).
dtdiff meander - node/prop order

$ dtdiff v1/test.dts v2/test.dts
--- /dev/fd/63
+++ /dev/fd/62
@@ -2,7 +2,7 @@

/ {
    compatible = "test";
-   model = "model_3";
+   model = "model_1";

   a {
       model = "model_a";
   }
dtdiff meander - node/prop order

dtdiff adds a sort to the decompile step

RED FLAG

Sometimes order in Expanded DT does matter!!!

If you are debugging a problem related to device creation or driver binding ordering then you may want to be aware of changes of node order. (Edit dtdiff, remove '-s')
The previous example of two instances of the same node in the same file is somewhat contrived.

But multiple instances of a node in a compilation unit is an extremely common pattern because of the conventions for using .dtsi files.
dtdiff meander - .dtsi convention

$ cat v1/acme_hub_full.dtsi
/dts-v1/
/include/ "acme_serial.dtsi"
/include/ "acme_modem.dtsi"

$ cat v1/acme_serial.dtsi
/ {
  serial {
    compatible = "acme,serial-card";
    port_type = "rs-232";
    ports = < 6 >;
    status = "disabled";
  };
};

$ cat v1/acme_modem.dtsi
/ {
  modem {
    compatible = "acme,modem-card";
    baud = < 9600 >;
    ports = < 12 >;
    status = "disabled";
  };
};

<--- common platform

<--- optional serial subsystem

<--- optional modem subsystem
dtdiff meander - .dtsi convention

$ cat v1/acme_hub_full.dtsi  <-- common platform
/dts-v1/;
/include/ "acme_serial.dtsi"
/include/ "acme_modem.dtsi"

$ cat v1/acme_serial.dtsi  <-- optional subsys
/ {
    serial {
        compatible = "acme,serial-card";
        port_type = "rs-232";
        ports = < 6 >;
        status = "disabled";
    }
};
};
dtdiff meander - .dtsi convention

System .dts – enable and customize HW blocks

$ cat v1/acme_hub_cheap.dts
/include/ "acme_hub_full.dtsi"
/
{
  compatible = "acme,hub-cheap";
  serial {
    ports = < 3 >;
    status = "ok";
  };
};
$ dtc v1/acme_hub_cheap.dts
/dts-v1/;
/
{
    compatible = "acme,hub-cheap";

    serial {
        compatible = "acme,serial-card";
        port_type = "rs-232";
        ports = <0x3>;
        status = "ok";
    };

    modem {
        compatible = "acme,modem-card";
        baud = <0x2580>;
        ports = <0xc>;
        status = "disabled";
    };
};
dtdiff - Why don't I just use 'diff'?

... and thus ends the long meander
Exercise for the advanced student

Extend the tools and techniques from this section for use with overlays.
Takeaway

- There are many ways that a device tree can be changed between the original source and the Extended DT in Linux kernel memory.
- DT includes suggest a different mental model than C language includes, when investigating.
- dtiff is a powerful tool for investigating changes, but may hide an important change.
- In some cases diff is more useful than dtiff.
A common problem that dtdiff does not solve:

A property is defined in multiple .dts and .dtsi files.

Which of the many locations is the one that ends up in the .dtb?
.dtb ---> .dts

current solution:
   scan the cpp output, from bottom to top, for
   the cpp comment that provides the file name

cpp output is available at
   ${KBUILD_OUTPUT}/arch/${ARCH}/boot/dts/XXX.dts.dtb.tmp
for XXX.dtb
example, where does the value of 'status' come from for pm8941_coincell?

```plaintext
# 1 "/.../arch/arm/boot/dts/qcom-pm8941.dtsi" 1
...
  pm8941_coincell: qcom,coincell@2800 {
    compatible = "qcom,pm8941-coincell";
    reg = <0x2800>;
    status = "disabled";
  }
...
# 4 "/.../arch/arm/boot/dts/qcom-apq8074-dragonboard.dts" 2
...
&pm8941_coincell {
  status = "ok";
```
Skipped to HERE

(go back)
Chapter 3

Kernel boot

Creating devices

Registering drivers / Binding drivers
When DT problems occur

Most DT problems happen during kernel boot:
  Creating devices
  Registering drivers / Binding drivers

How do you investigate these problems?
DT kernel boot - Reference

Frank Rowand's ELCE 2014 talk:

    devicetree:
    Kernel Internals and Practical Troubleshooting

http://elinux.org/ELC_Europe_2014_Presentations
My pseudocode conventions

Will obviously fail to compile
Will usually not show function arguments
Each level of indentation indicated either
body of control statement (if, while, etc)
entry into function listed on previous line
Double indentation indicates an intervening
level of function call is not shown
Will often leave out many details or fabricate
specific details in the interest of simplicity
extremely simplified boot

start_kernel()
    pr_notice("%s", linux_banner)
setup_arch()
    unflatten_device_tree()
pr_notice("Kernel command line: %s\n", ...)
init_IRQ()
...
time_init()
...
rest_init()
    kernel_thread(kernel_init, ...)
    kernel_init()
    do_initcalls() // device creation, driver binding
Takeaway

do_initcalls() is where
- devices are created
- drivers are registered
- drivers are bound to devices
Chapter 3.1

kernel boot

Creating devices

Registering drivers / Binding drivers
Initcalls occur in this order:

```c
char *initcall_level_names[] = {
  "early",
  "core",
  "postcore",
  "arch",
  "subsys",
  "fs",
  "device",
  "late",
};
```
initcall - of_platform_populate()

of_platform_populate(, NULL, , , )
for each child of DT root node
   rc = of_platform_bus_create(child, matches, lookup, parent, true)
   if (node has no 'compatible' property)
      return
   auxdata = lookup[X], where:
      # lookup[X]->compatible matches node compatible property
      # lookup[X]->phys_addr matches node resource 0 start
   if (auxdata)
      bus_id = auxdata->name
      platform_data = auxdata->platform_data
   dev = of_platform_device_create_pdata(, bus_id, platform_data, )
   dev = of_device_alloc(np, bus_id, parent)
   dev->dev.bus = &platform_bus_type
   dev->dev.platform_data = platform_data
   of_device_add(dev)
      bus_probe_device()
         ret = bus_for_each_drv(,, __device_attach)
         error = __device_attach()
            if (!driver_match_device()) return 0
            return driver_probe_device()
   if (node 'compatible' property != "simple-bus")
      return 0
   for_each_child_of_node(bus, child)
      rc = of_platform_bus_create()
      if (rc) break
   if (rc) break
initcall - of_platform_populate()

```c
of_platform_populate(, NULL,,,)  /* lookup is NULL */
  for each child of DT root node
    rc = of_platform_bus_create(child, )
    if (node has no 'compatible' property)
      return

<< create platform device for node >>
<< try to bind a driver to device >>

if (node 'compatible' property != "simple-bus")
  return 0
for_each_child_of_node(bus, child)
  rc = of_platform_bus_create(child, )
  if (rc) break
if (rc) break
```
<< create platform device for node >>
<< try to bind a driver to device >>

auxdata = lookup[X], with matches:
  lookup[X]->compatible == node 'compatible' property
  lookup[X]->phys_addr  == node resource 0 start
if (auxdata)
  bus_id = auxdata->name
  platform_data = auxdata->platform_data
dev = of_platform_device_create_pdata(, bus_id,
                                        platform_data,)
dev = of_device_alloc(, bus_id,)
device->dev.bus = &platform_bus_type
dev->dev.platform_data = platform_data
of_device_add(dev)
  bus_probe_device()
  ret = bus_for_each_drv(, __device_attach)
  error = __device_attach()
  if (!driver_match_device())
    return 0
  return driver_probe_device()
platform device created for
- children of root node
- recursively for deeper nodes if 'compatible' property == "simple-bus"

platform device not created if
- node has no 'compatible' property
Drivers may be bound to the devices during platform device creation if

- the driver called platform_driver_register() from a core_initcall() or a postcore_initcall()
- the driver called platform_driver_register() from an arch_initcall() that was called before of_platform_populate()
Creating other devices

Devices that are not platform devices were not created by `of_platform_populate()`.

These devices are typically non-discoverable devices sitting on more remote busses. For example:

- i2c
- SoC specific busses
Creating other devices

Devices that are not platform devices were not created by of_platform_populate().

These devices are typically created by the bus driver probe function
Non-platform devices

When a bus controller driver probe function creates the devices on its bus, the device creation will result in the device probe function being called if the device driver has already been registered.

Note the potential interleaving between device creation and driver binding.
What got skipped

When does driver attempt to bind to device?

- If a device already exists, when the driver is registered
- If a driver is already registered, when the device is created
- If deferred on the first attempt, then again later.
Debugging Boot Problems

What can go wrong while trying to:

- register driver
- create device
- bind driver to device
Debugging Boot Problems

What can go wrong while trying to:
- register driver
- create device
- bind driver to device

I will provide
- some examples of failures at various stages
- tools and techniques to investigate
Another new tool

What is this tool?
Where do I get it?
dt_node_info - What is this tool?

/proc/device-tree and /sys/devices provide visibility into the state and data of

- Flattened Device Tree
- Expanded Device Tree
- Devices
dt_node_info - What is this tool?

/proc/device-tree and /sys/devices provide visibility into the state and data of
- Flattened Device Tree
- Expanded Device Tree
- Devices

dt_stat probes this information to provide several reports

dt_node_info packages the information from dt_stat in an easy to scan summary
dt_node_info - Where do I get it?

**Work In Progress patch** is at:

- [http://elinux.org/Device_Tree_frowand](http://elinux.org/Device_Tree_frowand)

Dependency:

- requires device tree information to be present in sysfs

Tested:

- only on Linux 4.1-rc2 dragonboard

**Might** work as early as Linux 3.17. Please let me know if it works for you on versions before 4.1.
$ dt_stat --help

usage:
  dt_stat

    -h          synonym for --help
    -help       synonym for --help
    --help      print this message and exit

    --d          report devices
    --n          report nodes
    --nb         report nodes bound to a driver
    --nd         report nodes with a device
    --nxb        report nodes not bound to a driver
    --nxd        report nodes without a device
dt_stat - usage:

Reports about nodes in /proc/device-tree/
Nodes without a compatible string are not reported

data fields reported:
--d       Device Node
--n       Node Compatible
--nb      Node Compatible
--nd      Node Compatible Device Driver
--nxb     Node Compatible
--nxd     Node Compatible
$ dt_stat --nb
/clock-controller qcom,krait-cc-v2
/cpu-pmu qcom,krait-pmu
/soc/clock-controller@fc400000 qcom,gcc-msm8974
/soc/clock-controller@fd8c0000 qcom,mmcc-msm8974
/soc/i2c@f9967000 qcom,i2c-qup-v2.1.1
/soc/pinctrl@fd510000 qcom,msm8974-pinctrl
/soc/restart@fc4ab000 qcom,pshold
/soc/rng@f9bff000 qcom,prng
/soc/sdhci@f9824900 qcom,sdhci-msm-v4
/soc/serial@f991e000 qcom,msm-uartdm-v1.4qcom,msm-uartdm
/soc/spmi@fc4cf000 qcom,spmi-pmic-arb
/soc/spmi@fc4cf000/pm8841@4 qcom,spmi-pmic
/soc/spmi@fc4cf000/pm8841@5 qcom,spmi-pmic
/soc/spmi@fc4cf000/pm8941@0 qcom,spmi-pmic
/soc/spmi@fc4cf000/pm8941@0/qcom,coincell@2800 qcom,pm894
/soc/spmi@fc4cf000/pm8941@1 qcom,spmi-pmic
$ dt_stat --nd

/clock-controller qcom,krait-cc-v2 /sys/devices/platform/clock-controller clock-krait
/cpu-pmu qcom,krait-pmu /sys/devices/platform/cpu-pmu arm-pmu
/soc/clock-controller@fc400000 qcom,gcc-msm8974 /sys/devices/platform/soc/fc400000.clock-controller gcc-msm8974
/soc/clock-controller@fd8c0000 qcom,mmcc-msm8974 /sys/devices/platform/soc/fd8c0000.clock-controller mmcc-msm8974
/soc/i2c@f9967000 qcom,i2c-qup-v2.1.1 /sys/devices/platform/soc/f9967000.i2c i2c_qup
/soc/pinctrl@fd510000 qcom,msm8974-pinctrl /sys/devices/platform/soc/fd510000.pinctrl msm8x74-pinctrl
/soc/restart@fc4ab000 qcom,pshall /sys/devices/platform/soc/fc4ab000.restart msm-restart
/soc/rng@f9bff000 qcom,prng /sys/devices/platform/soc/f9bff000.rng msm_rng
/soc/sdhci@f9824900 qcom,sdhci-msm-v4 /sys/devices/platform/soc/f9824900.sdhci sdhci-msm
/soc/serial@f991e000 qcom,msm-uartdm-v1.4qcom,msm-uartdm /sys/devices/platform/soc/f991e000.serial msm_serial
/soc/spmi@fc4cf000 qcom,spmi-pmic-arb /sys/devices/platform/soc/fc4cf000.spmi spmi_pmic_arb
/soc/spmi@fc4cf000/pm8841@0 qcom,pm8841@0 /sys/devices/platform/soc/fc4cf000.spmi/spmi-0/0-00 pmic_spmi
/soc/spmi@fc4cf000(pm8841@0 qcom,pm8841@0 /sys/devices/platform/soc/fc4cf000.spmi/spmi-0/0-00 pmic_spmi
/soc/spmi@fc4cf000(pm8841@0 qcom,pm8841@0 /sys/devices/platform/soc/fc4cf000.spmi/spmi-0/0-00 pmic_spmi
/qcom,apq8074-dragonboardqcom,apq8074 /sys/devices/platform/alarmtimer alarmtimer
/qcom,apq8074-dragonboardqcom,apq8074 /sys/devices/platform/reg-dummy reg-dummy
/qcom,apq8074-dragonboardqcom,apq8074 /sys/devices/platform/snd-soc-dummy snd-soc-dummy
/qcom,apq8074-dragonboardqcom,apq8074 /sys/devices/platform/soc/f9824900.sdhci/mmc_host/mmc0/mmc0:0001 mmcblk
dt_stat - example --nd

$ dt_stat --nd
/clock-controller qcom,krait-cc-v2 /sys/devices/platform/clock-controller
cpu-pmu qcom,krait-pmu /sys/devices/platform/cpu-pmu arm-pmu
clock-controller@fc400000 qcom,gcc-msm8974 /sys/devices/platform/soc/fc400000
clock-controller@fd8c0000 qcom,mmcc-msm8974 /sys/devices/platform/soc/fd8c0000
i2c@f9967000 qcom,i2c-qup-v2.1.1 /sys/devices/platform/soc/f9967000.i2c
pinctrl@fd510000 qcom,msm8974-pinctrl1 /sys/devices/platform/soc/fd510000
restart@fc4ab000 qcom,pshold /sys/devices/platform/soc/fc4ab000.restart
rng@f9bff000 qcom,prng /sys/devices/platform/soc/f9bff000 rng msm_rng
sdhci@f9824900 qcom,sdhci-msm-v4 /sys/devices/platform/soc/f9824900.sdhci
serial@f991e000 qcom,msm-uartdm-v1.4qcom,msm-uartdm /sys/devices/platform/soc
spmi@fc4cf000 spmi-pmic-arb /sys/devices/platform/soc/fc4cf000.spmi
spmi@fc4cf000/0q8414 qcom,spmi-pmic /sys/devices/platform/soc/fc4cf000/0q8414
spmi@fc4cf000/0q8415 qcom,spmi-pmic /sys/devices/platform/soc/fc4cf000/0q8415
spmi@fc4cf000/0q8941 qcom,spmi-pmic /sys/devices/platform/soc/fc4cf000/0q8941
spmi@fc4cf000/0q8941/qcom,coincell@2800 qcom,pm8941-coincell /sys/devices/fc4cf000
spmi@fc4cf000/0q8941/qcom,coincell@2800 qcom,pm8941-coincell
spmi@fc4cf000/0q8941/qcom,pm8941-qcom,apq8074-dragonboardqcom,apq8074 /sys/devices/platform/alarmtimer
spmi@fc4cf000/0q8941/qcom,apq8074-dragonboardqcom,apq8074 /sys/devices/platform/reg-dummy
spmi@fc4cf000/0q8941/qcom,apq8074-dragonboardqcom,apq8074 /sys/devices/platform/snd-soc-dummy
spmi@fc4cf000/0q8941/qcom,apq8074-dragonboardqcom,apq8074 /sys/devices/platform/soc/f9824900.sdhci

skip
$ dt_stat --nxb
/cpus/cpu@0  qcom,krait
/cpus/cpu@1  qcom,krait
/cpus/cpu@2  qcom,krait
/cpus/cpu@3  qcom,krait
/cpus/idle-states/spc  qcom,idle-state-spca,rm,idle-state
/cpus/l2-cache  cache
/cpus/spmi@fc4cf000  qcom,spmi-pmic-arb
/cpus/spmi@fc4cf000/pm8841@4  qcom,pm8841
/cpus/spmi@fc4cf000/pm8841@5  qcom,pm8841
/cpus/spmi@fc4cf000/pm8941@0  qcom,pm8941
/cpus/spmi@fc4cf000/pm8941@1  qcom,pm8941
/soc  simple-bus
/soc/clock-controller@f9016000  qcom,hfpll
/soc/clock-controller@f9088000  qcom,kpss-acc-v2
/soc/clock-controller@f908a000  qcom,hfpll
/soc/clock-controller@f9098000  qcom,kpss-acc-v2
$ dt_stat --nxd
/cpus/idle-states/spc qcom,idle-state-spcarm,idle-state
/cpus/l2-cache cache
/cpus/spmi@fc4cf000 qcom,spmi-pmic-arb
/cpus/spmi@fc4cf000/pm8841@4 qcom,pm8841
/cpus/spmi@fc4cf000/pm8841@5 qcom,pm8841
/cpus/spmi@fc4cf000/pm8941@0 qcom,pm8941
/cpus/spmi@fc4cf000/pm8941@1 qcom,pm8941
/soc/sdhci@f98a4900 qcom,sdhci-msm-v4
Boot Problem - device not created

$ dt_node_info coincell

===== devices

===== nodes
/soc/spmi@fc4cf000/pm8941@0/qcom,coincell@2800 qcom,

===== nodes bound to a driver

===== nodes with a device

===== nodes not bound to a driver
/soc/spmi@fc4cf000/pm8941@0/qcom,coincell@2800 qcom,

===== nodes without a device
/soc/spmi@fc4cf000/pm8941@0/qcom,coincell@2800 qcom,
Look at Expanded DT

1) copy /proc/device-tree from target to base/

2) decompile base/
dtdiff base
Look at Expanded DT

pm8941@0 {
    #address-cells = <0x1>;
    #size-cells = <0x0>;
    compatible = "qcom,spmi-pmic";
    reg = <0x0 0x0>;

    qcom,coincell@2800 {
        compatible = "qcom,pm8941-coincell";
        qcom,charge-enable;
        qcom,rset-ohms = <0x834>;
        qcom,vset-millivolts = <0xbb8>;
        reg = <0x2800>;
        status = "disabled";
        stratus = "ok";
    }
};
Look at Expanded DT

qcom,coincell@28000 {  
  compatible = "qcom,pm8941-coincell";
  qcom,charge-enable;
  qcom,rset-ohms = <0x834>;
  qcom,vset-millivolts = <0xbb8>;
  reg = <0x2800>;
  status = "disabled";
  stratus = "ok";
};
Chapter 3.2

kernel boot

Creating devices

Registering drivers / Binding drivers
platform_driver_register()
    driver_register()

while (dev = iterate over devices on the platform_bus)
    if (!driver_match_device()) return 0
    if (dev->driver) return 0

    driver_probe_device()
    really_probe(dev, drv)
    ret = pinctrl_bind_pins(dev)
    if (ret)
        goto probe_failed

    if (dev->bus->probe)
        ret = dev->bus->probe(dev)
        if (ret) goto probe_failed
    else if (drv->probe)
        ret = drv->probe(dev)
        if (ret) goto probe_failed

driver_bound(dev)
    driver_deferred_probe_trigger()

    if (dev->bus)
        blocking_notifier_call_chain()
initcall - // driver binding

Reformatting the previous slide to make it more readable (see next slide)
initcall - // driver binding

platform_driver_register()
  while (dev = iterate over devices on platform_bus)
    if (!driver_match_device()) return 0
    if (dev->driver) return 0
  driver_probe_device()
    really_probe(dev, drv)
    ret = pinctrl_bind_pins(dev)
    if (ret)
      goto probe_failed
    if (dev->bus->probe)
      ret = dev->bus->probe(dev)
      if (ret) goto probe_failed
    else if (drv->probe)
      ret = drv->probe(dev)
      if (ret) goto probe_failed
  driver_bound(dev)
  driver_deferred_probe_trigger()
  if (...) blocking_notifier_call_chain()
Problem - driver not bound

Many possible problems that may result in driver not binding to the device.

Will debug several problems...
Problem - driver not bound (1)

$ dt_node_info coincell
==== devices
/sys/devices/platform/soc/fc4cf000.spmi/spmi-0/0-00/

==== nodes
/soc/spmi@fc4cf000/pm8941@0/qcom,coincell@2800 qcom,

==== nodes bound to a driver

==== nodes with a device
/soc/spmi@fc4cf000/pm8941@0/qcom,coincell@2800 qcom,

==== nodes not bound to a driver
/soc/spmi@fc4cf000/pm8941@0/qcom,coincell@2800 qcom,

==== nodes without a device
Problem - driver not bound (1) skip

$ dt_node_info coincell
===== devices
/sys/devices/platform/soc/fc4cf000.spmi/spmi-0/0-00/

Output from dt_node_info truncated on the right.

Most slides showing dt_node_info output will be truncated in this manner.
Problem - driver not bound (1)

Was the driver configured into the kernel?

Device tree node:

```
pm8941_coincell: qcom,coincell@2800 {
    compatible = "qcom,pm8941-coincell";
    reg = <0x2800>;
    status = "disabled";
};
```

Search for compatible = "qcom,pm8941-coincell" in the kernel source
Problem - driver not bound (1)

Search for compatible = "qcom,pm8941-coincell"
in the kernel source

$ git grep "qcom,pm8941-coincell"

arch/arm/boot/dts/qcom-pm8941.dtsi:                     compatible = "qcom,pm894
drivers/misc/qcom-coincell.c:   { .compatible = "qcom,pm8941-coincell", },
drivers/misc/qcom-coincell.c:     .name = "qcom,pm8941-coincell"

(driver is drivers/misc/qcom-coincell.c)

Search for the config option to compile the driver
Problem - driver not bound (1)

Search for the config option to compile the driver

$ grep qcom-coincell \ drivers/misc/Makefile
  obj-$(CONFIG_QCOM_COINCELL) += qcom-coincell.o

$ grep CONFIG_QCOM_COINCELL \ ${KBUILD_OUTPUT}/.config
# CONFIG_QCOM_COINCELL is not set
Problem - driver not bound (1)

FIX and try again

Enable config option for the driver

$ grep CONFIG_QCOM_COINCELL \\n  ${KBUILD_OUTPUT}/.config
CONFIG_QCOM_COINCELL=y
Sidetrack

Q. Why is there no tool to generate a list of config options required by a device tree?

A. There are several proposed tools, but you have to find them. And they are not very mature yet.

-----  Opportunity for improvement
Problem - driver not bound (2)

$ dt_node_info coincell
===== devices
/sys/devices/platform/soc/fc4cf000.spmi/spmi-0/0-00/

===== nodes
/soc/spmi@fc4cf000/pm8941@0/qcom,coincell@2800 qcom,

===== nodes bound to a driver

===== nodes with a device
/soc/spmi@fc4cf000/pm8941@0/qcom,coincell@2800 qcom,

===== nodes not bound to a driver
/soc/spmi@fc4cf000/pm8941@0/qcom,coincell@2800 qcom,

===== nodes without a device
Problem - driver not bound (2)

Was the driver registered at boot?

-----  Target system  -----  

Kernel command line: debug
dyndbg="func bus_add_driver +p"

$ dmesg | grep coin
$ dmesg | grep "add driver"
bus: 'platform': add driver CCI-400 PMU
bus: 'platform': add driver CCI-400
...

Assumptions

Kernel command line:
  dyndbg="func bus_add_driver +p"

'dyndbg' requires CONFIG_DYNAMIC_DEBUG=y

'debug' may be used to set the loglevel so debug messages appear on the console

CONFIG_MESSAGE_LOGLEVEL_DEFAULT may also be used to set the loglevel

The dmesg command can be used to print the debug messages.
Problem - driver not bound (2)

Was the driver registered at boot?

-----  Host system  -----  

$ grep qcom_coincell System.map
$

Look for driver registration in source code

Cause: no driver registration in source code
Problem - driver not bound (2) skip

FIX and try again

Add driver registration in source code

```c
static const struct of_device_id qcom_coincell_match_table[] = {
    { .compatible = "qcom,pm8941-coincell", },
    {}};

MODULE_DEVICE_TABLE(of, qcom_coincell_match_table);

static struct platform_driver qcom_coincell_driver = {
    .driver = {
        .name = "qcom,pm8941-coincell",
        .of_match_table = qcom_coincell_match_table,
    },
    .probe = qcom_coincell_probe,
};

module_platform_driver(qcom_coincell_driver);
```
Problem - driver not bound (2)

FIX and try again

Add driver registration in source code

module_platform_driver(qcom_coincell_driver);
Problem - driver not bound (2)

FIX and try again

$ grep qcom_coincell System.map
  c054f880  t  qcom_coincell_probe
  c078ea28  r  qcom_coincell_match_table
  c09cec8c  t  qcom_coincell_driver_init
  c09e5d64  t  qcom_coincell_driver_exit
  c09f2f18  t  __initcall_qcom_coincell_driver_init6
  c0a4153c  d  qcom_coincell_driver
Problem - driver not bound (3)

$ dt_node_info coincell

==== devices
/sys/devices/platform/soc/fc4cf000.spmi/spmi-0/0-00/

==== nodes
/soc/spmi@fc4cf000/pm8941@0/qcom,coincell@2800 qcom,

==== nodes bound to a driver

==== nodes with a device
/soc/spmi@fc4cf000/pm8941@0/qcom,coincell@2800 qcom,

==== nodes not bound to a driver
/soc/spmi@fc4cf000/pm8941@0/qcom,coincell@2800 qcom,

==== nodes without a device
Problem - driver not bound (3)

Was the driver probe successful at boot?

-----  Target system  -----  

Kernel command line:  
  dyndbg="func bus_add_driver +p"
  dyndbg="func really_probe +p"

$ dmesg | grep coin
bus: 'platform': add driver qcom,pm8941-coincell
bus: 'platform': really_probe: probing driver qcom,pm8941-coincell
  with device fc4cf000.spmi:pm8941@0:qcom,coincell@2800
qcom,pm8941-coincell: probe of fc4cf000.spmi:pm8941@0:qcom,coincell@2800 failed with error -22
Problem - driver not bound (3)

qcom,pm8941-coincell: probe of ... failed with error -22

include/uapi/asm-generic/errno-base.h:

    #define EINVAL          22      /* Invalid argument */

$ grep EINVAL drivers/misc/qcom-coincell.c
   return -EINVAL;
   return -EINVAL;
   return -EINVAL;
Problem - driver not bound (3)

$ grep EINVAL drivers/misc/qcom-coincell.c
  return -EINVAL;
  return -EINVAL;
  return -EINVAL;

Debug strategy (1):
  Add printk() for each EINVAL return.

Result:
  None of the printk() occur.
Debug strategy (1):
Add printk() for each EINVAL return.

There are some alternatives to printk(), eg:
- read the C source, follow all possible paths returning error values, examine the decompiled EDT to see if missing or existing properties would trigger the error
- trace_printk()
- kernel debugger breakpoint
- kernel debugger tracepoint

To keep the slides concise, I will only list printk().
Problem - driver not bound (3)

qcom_coincell_probe() calls several other functions which may return errors. The common pattern is:

```c
rc = xxx();
if (rc)
    return rc;
```

Debug strategy (2):
Add printk() for each rc return.
Problem - driver not bound (3)

Debug strategy (2):
   Add printk() for each rc return.

Result:
   The error is returned from:

   rc = of_property_read_u32(node, "qcom,rset-ohms", &rset);
EINVAL is many call levels deep

This type of error is hard to find by reading source

    of_property_read_u32()
of_property_read_u32_array()
    val = of_find_property_value_of_size()
    *prop = of_find_property()
    if (!prop):
        if (!prop):
            return ERR_PTR(-EINVAL)
    if (IS_ERR(val))
        return PTR_ERR(val)
FULL DISCLOSURE

The dev_err() error report is present in the real driver.

For the example, I removed the dev_err() to show how important it is to clearly report errors that result in the probe failing.
Problem - driver not bound (3)

PARTIAL FIX and try again

Add precise error message to driver.

Retain the underlying error to show how useful the error message is.

```c
rc = of_property_read_u32(node, "qcom,rset-ohms", &rset);
if (rc) {
    dev_err(chgr->dev, "can't find 'qcom,rset-ohms' in DT block");
    return rc;
}
```
Problem - driver not bound (4)

Showing the real error message!

$ dmesg | grep coin

...  
qcom,pm8941-coincell
   fc4cf000.spmi:pm8941@0:qcom,coincell@2800:
      can't find 'qcom,rset-ohms' in DT block
qcom,pm8941-coincell:
   probe of fc4cf000.spmi:pm8941@0:qcom,coincell:
      failed with error -22
Problem - driver not bound (4)

can't find 'qcom,rset-ohms' in DT block

failed with error -22

The detailed message provides enough information to easily troubleshoot the problem.
Problem - driver not bound (4)

FIX and try again

Add property 'qcom,rset-ohms' to the pm8941_coincell device tree node.
FIXED - driver bound to device

$ dt_node_info coincell
====== devices
/sys/devices/platform/soc/fc4cf000.spmi/spmi-0/0-00/

====== nodes
/soc/spmi@fc4cf000/pm8941@0/qcom,coincell@2800 qcom,

====== nodes bound to a driver
/soc/spmi@fc4cf000/pm8941@0/qcom,coincell@2800 qcom,

====== nodes with a device
/soc/spmi@fc4cf000/pm8941@0/qcom,coincell@2800 qcom,

====== nodes not bound to a driver

====== nodes without a device
More useful data: driver

What bus was the driver registered for?

-----  Target system  -----  

Kernel command line:  
  dyndbg="func bus_add_driver +p"

$ dmesg | grep "add driver"
bus: 'XXX': add driver ZZZ

Examples of bus type on next slide
$ dmesg | grep "add driver"
bus: 'platform': add driver gcc-msm8974
bus: 'i2c': add driver dummy
bus: 'mdio_bus': add driver Generic PHY
bus: 'usb': add driver hub
bus: 'qcom_smd': add driver wcnss_ctrl
bus: 'spmi': add driver pmic-spmi
bus: 'scsi': add driver sd
bus: 'spi': add driver m25p80
bus: 'mmc': add driver mmcblk
bus: 'amba': add driver mmci-pl18x
bus: 'hid': add driver hid-generic
More useful data: driver

Deferred probe issues

-----  Target system  -----  

Kernel command line:
  dyndbg="func deferred_probe_work_func +p"
  dyndbg="func driver_deferred_probe_add +p"
  dyndbg="func driver_deferred_probe_add +p"
  dyndbg="func driver_deferred_probe_del +p"
Typical driver binding patterns

Make these substitutions on the following slides

BUS --- the bus name
DEV --- the device name
DVR --- the driver name
Device Creation ---> probe

create child: NODE
device: 'DEV': device_add
bus: 'BUS': driver_probe_device: matched device DEV with driver DVR
bus: 'BUS': really_probe: probing driver DVR with device DEV

=====  messages from driver probe function  =====
driver: 'DVR': driver_bound: bound to device 'DEV'
bus: 'BUS': really_probe: bound device DEV to driver DVR
Driver Register ---> probe

bus: 'BUS': add driver DVR
bus: 'BUS': driver_probe_device: matched device DEV with driver DVR
bus: 'BUS': really_probe: probing driver DVR with device DEV

===== messages from driver probe function ======

driver: 'DVR': driver_bound: bound to device 'DEV'
bus: 'BUS': really_probe: bound device DEV to driver DVR
Deferred Probe ---> re-probe

bus: 'BUS': add driver DVR
device: 'DEV': device_add
bus: 'BUS': driver_probe_device: matched device DEV with DVR
bus: 'BUS': really_probe: probing driver DVR with device DEV

===== messages from driver probe function =====

BUS DEV: Driver DVR requests probe deferral
BUS DEV: Added to deferred list
BUS DEV: Retrying from deferred list
bus: 'BUS': driver_probe_device: matched DEV with driver DVR
bus: 'BUS': really_probe: probing driver DVR with device DEV

===== messages from driver probe function =====

driver: 'DVR': driver_bound: bound to device 'DEV'
bus: 'BUS': really_probe: bound device DEV to driver DVR
Summary:

dyndbg="func of_platform_bus_create +p"
dyndbg="func bus_add_driver +p"
dyndbg="func device_add +p"
dyndbg="func driver_probe_device +p"
dyndbg="func really_probe +p"
dyndbg="func driver_bound +p"
dyndbg="func deferred_probe_work_func +p"
dyndbg="func driver_deferred_probe_add +p"
dyndbg="func driver_deferred_probe_add +p"
dyndbg="func driver_deferred_probe_del +p"
Takeaway

/proc/device-tree and /sys/devices provide visibility into the state and data of

- Device Tree
- Devices
- Drivers
Takeaway

/proc/device-tree and /sys/devices provide visibility into the state and data of
- Device Tree
- Devices
- Drivers

**dt_stat** combines this information to provide several reports

**dt_node_info** packages the information from dt_stat in an easy to scan summary
Takeaway

kernel command line dyndbg options can provide a lot of information about what is causing device creation and driver binding errors.
Takeaway

Driver authors: if enough information is provided in error messages then DT source errors should be solvable without reading the driver source.
Review

Comparing device trees through the life cycle
- (skipped)
  - transformations during build, boot loader, kernel boot, run-time
  - dtdiff (patches required)

Kernel boot: device creation, driver binding
- dyndbg
- dt_stat
- dtdiff
Review - Why this talk?

At the end of this talk, you will know how to:

- debug some common device tree problems
- access data to support the debug process

Debugging some types of device tree problems will be easier.
Resources

Resources for "Solving Device Tree Issues" talk, LinuxCon Japan - June 4, 2015
  http://elinux.org/Device_Tree_frowand
  More detailed information on how to perform the tasks in this talk

Device Tree For Dummies, Thomas Pettazzoni, ELC 2014

devicetree: Kernel Internals and Practical Troubleshooting
Frank Rowand, ELCE 2014
  http://elinux.org/ELC_Europe_2014_Presentations
THE END

Thank you for your attention...
Questions?
How to get a copy of the slides

1) leave a business card with me
2) frank.rowand@sonymobile.com
3) http://elinux.org/Device_Tree
4) http://events.linuxfoundation.org
Linux Plumbers Conference
Seattle, August 19 - 21

Device Tree Tools, Validation, and Troubleshooting track

This is your chance to participate in shaping and improving device tree tools and processes

Early registration (reduced price) ends Fri June 5

http://linuxplumbersconf.org/2015/
Device Tree Tools, Validation, and Troubleshooting track

Contact frowand.list@gmail.com (Frank Rowand)

- if you plan to attend and:
  - need to avoid conflicts with other tracks
  - want to lead a session
  - want to propose a topic

- if you can not attend, but you have some content, perspective, needs, or other information that you want to be shared at the event