Bootph: A Swiss Army Knife for Boot-Time Optimization

Gokul Praveen

Beleswar Padhi

Keerthy J

About us: TI Processors and Open Source

Upstream FIRST!

- ✔ Upstream FIRST mentality!
- ✔ Decades of contribution and collaboration
- ✔ Focus on long term, sustainable and quality products
- ✔ Upstream & opensource ecosystem in device architecture
- ✔ Ingrained culture to give back to the community

Introduction to the Speakers

- **Gokul Praveen**
  - Software Applications Engineer at Texas Instruments
    - Mainly focused on boot time optimizations, board bring ups with Linux, U-Boot, and handling platform-specific drivers for connectivity peripherals.
- **Beleswar Padhi**
  - Senior Software Engineer at Texas Instruments
    - Mainly focused on Remoteproc, RPMsg, Mailbox, Virtio subsystems, as well as boot-time optimizations.
- **Keerthy Jagadish**
  - Jacinto Apps Manager at Texas Instruments, Bengaluru
    - Mainly focused on boot time optimizations & board bring ups with Linux

Disclaimers

- This is a technology presentation, not product-readiness or roadmap commitment.
- Opinions presented here are that of the speakers and may not reflect that of Texas Instruments Inc.
- All the boot-time profiling is _NOT_ done on production binaries, and only done on default upstream offerings. The aim is to highlight how a upstream codebase can be optimized heavily by the use of appropriate bootph tags. Users who are working on production binaries can also adopt this approach for boot-time optimization.

AGENDA

- **This session will cover the following :**
  - Generic UBOOT Bootflow
  - Intro into Device Tree and how it came up
  - A practical, in-depth look at the bootph (boot phase) device tree tags in U-Boot,(How, when and where)
  - Using Texas Instruments SoCs as a real-world example, we will share concrete boot-time and memory usage statistics demonstrating the impact of bootph tagging.
  - Some Lessons learned and common pitfalls encountered when introducing bootph tags into existing device trees.
  - The session concludes with some modern optimizations on device trees.

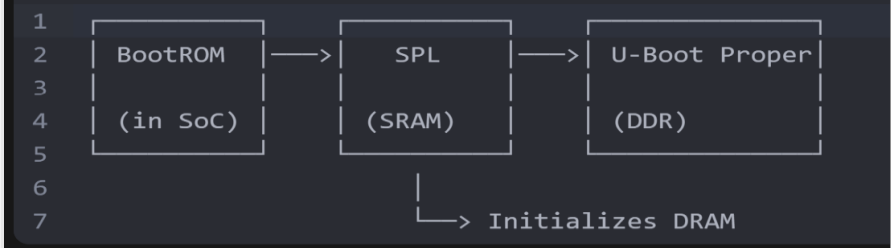
Problem Statement

- **Core Problem :**
  - **Time constraint**
    - With more stringent regulations for automotive use-cases, every millisecond of boot time is critical. Safety features like rear-view camera and surround view must start working quickly to meet regulations.
  - **Memory Constraint**
    - This becomes more important for bootloaders like U-Boot which cannot use the entire device tree in early phases due to memory constraints**.**
- **Solution :**
  - This is where the **bootph tags** come into the picture, by including and initializing only the required hardware in each phase of boot.

Overview of U-Boot boot phases(TPL, VPL, SPL, and U-Boot proper)

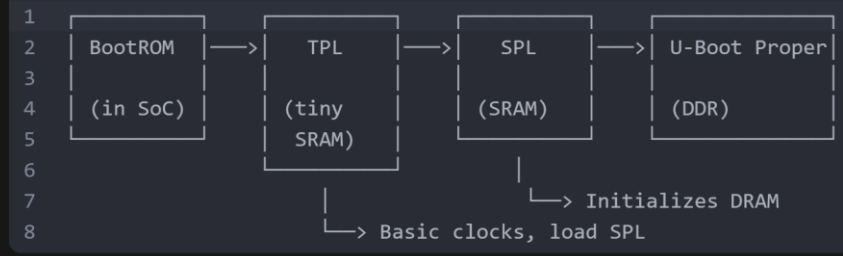
PHASE	TPL	VPL	SPL	U-BOOT proper
TYPICAL SIZE	8-32 KB	16-64 KB	32-128 KB	500 KB - 2 MB+
RUNS FROM	Startup SRAM	Larger SRAM	Larger SRAM	DDR
USE CASE	Very limited SRAM(<64KB) SoCs	Secure/verified boot chains (newer addition to U-Boot)	Most ARM/RISC-V SoCs	All systems (main bootloader)
KEY FUNCTION	• Initialize critical clocks • Setup minimal SRAM • Load SPL from storage • Jump to SPL	• Can select one of several SPL binaries, if A/B verified boot is enabled. • Implementation of the VPL logic is work-in-progress. • For now it just boots into SPL.	• **Initialize DRAM** Stack , gd and Heap can be moved to DRAM(optional). • Initialize clocks fully • Setup pinmux • Load U-Boot proper • It may also load other firmware components. • **Optional**: verify U-Boot signature.	**Pre-reloc** • Set up logging and console • Prepare for relocation • Plan memory layout Reserve memory for various components: • U-Boot code (Upper portion of memory) and data • Stacks and heaps • Global data structures • Set up new global data pointer **Post reloc** • Full hardware initialization. • Load kernel/OS through network or other boot media.

Scenario 1: Simple SoC (Most Common)



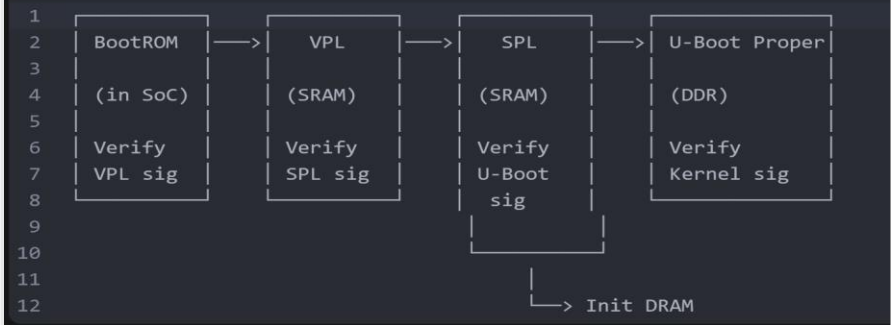
Example: BeagleBone Black, Raspberry Pi variants

Scenario 2: Very Constrained SoC



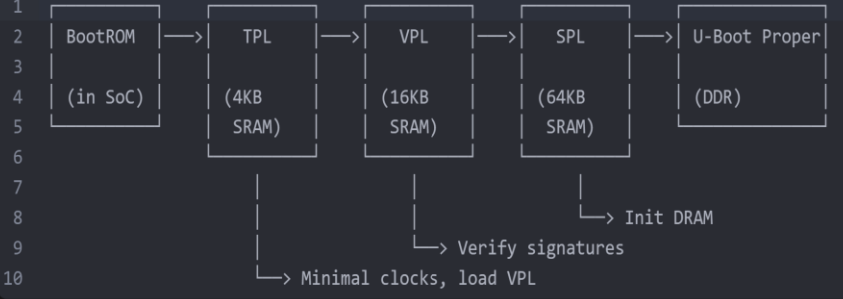
Example: Rockchip RK3399 (with tiny SRAM), some FPGA SoCs

Scenario 3: Secure Boot Chain



Example: Systems requiring verified boot (Android, automotive)

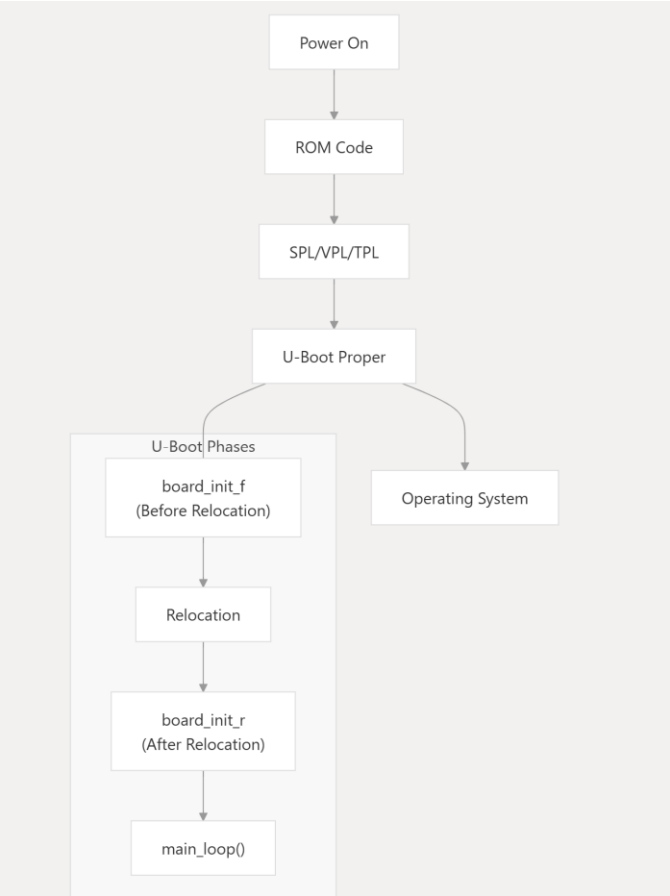
Scenario 4: Complex SoC with All Phases



Example: High-security embedded systems

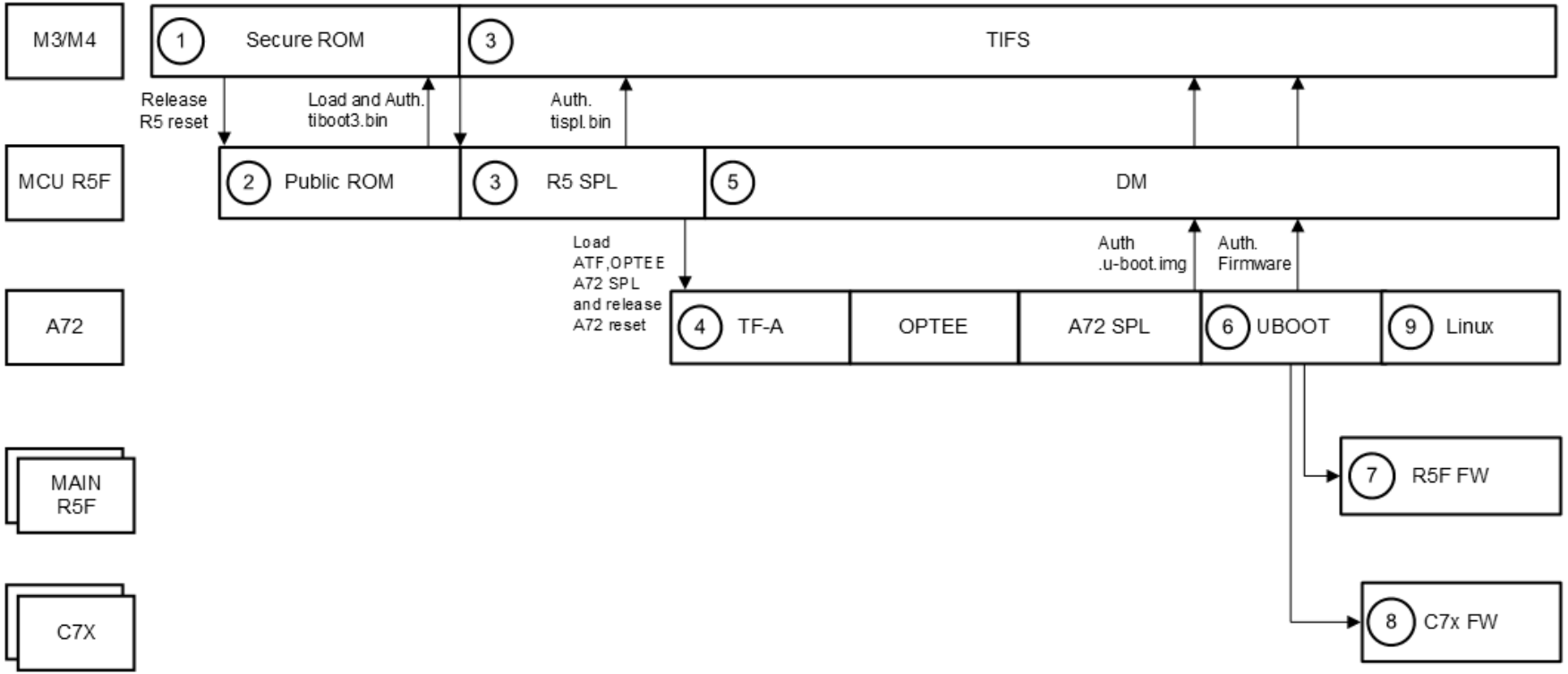
Texas Instruments

Generic Bootflow



Texas Instruments

DRA821(Heterogenous SoC BOOT FLOW)



Texas Instruments

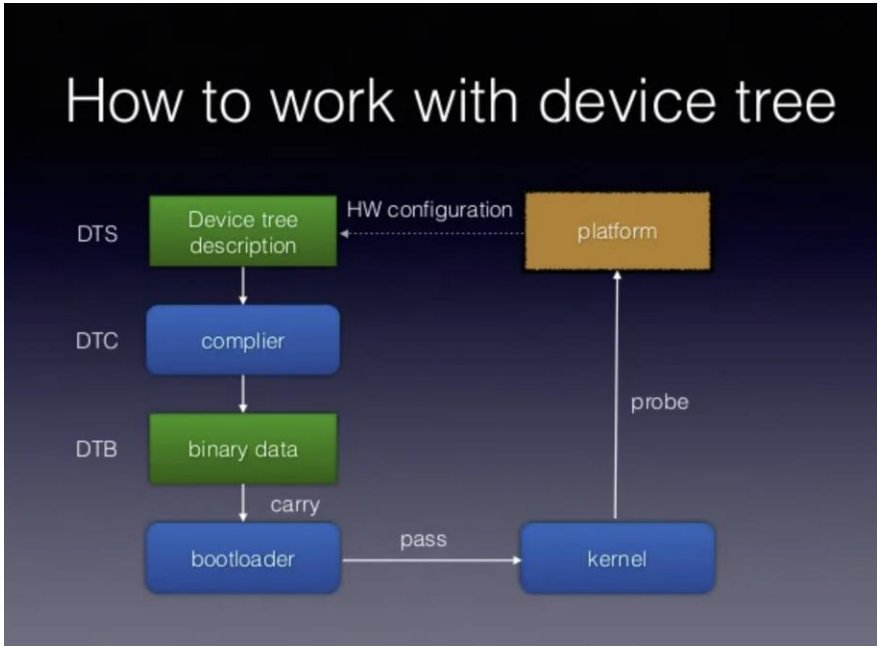
Device tree

What is a device tree ?

- A **devicetree** is a data structure describing the hardware components of a particular computer/board so that the

Flattened device tree(fdt)

- Compiled Binary form of the DT(.dtb format).
- The fdt is a convenient vehicle for implementing run-time configuration for three reasons:
  - A compiler(dtc: device tree compiler) checks the dts file and converts it to a compact binary format,
  - A library is already available in U-Boot (**libfdt**) for reading, writing or modifying .dtb files programmatically.
  - It is extensible since it consists of nodes and properties in a nice hierarchical format.



Texas Instruments

Device Tree : Configuration in UBOOT

Configuration

- U-Boot rather maintains a Git subtree as *dts/upstream/* sub-directory. It is regularly synced with the Linux kernel and hence no need for manual devicetree sync.
- You may find that the *dts/upstream/* already has a suitable devicetree file for your board. Look in *dts/upstream/src/<arch>/<vendor>*.
- SoC/board maintainers are encouraged to migrate to use synced copies from *dts/upstream/src/<arch>/<vendor>*.
- To do that:
  - add ***imply OF_UPSTREAM*** for the SoC being used via Kconfig
  - set ***DEFAULT_DEVICE_TREE***=*<vendor>/<name>* when prompted by Kconfig.

Adding tweaks for U-Boot

- In addition to the *dts/upstream* Git subtree, U-Boot automatically looks for and includes a file with updates to the standard devicetree for your board, searching for them in *arch/<arch>/dts/* in this order.
- Only one of these is selected, but, of course, you can #include another one within that file, to create a hierarchy of shared files.

```
<orig_filename>-u-boot.dtsi
<CONFIG_SYS_SOC>-u-boot.dtsi
<CONFIG_SYS_CPU>-u-boot.dtsi
<CONFIG_SYS_VENDOR>-u-boot.dtsi
u-boot.dtsi
```

Why are these tweaks needed?

- U-Boot is, of course, a very different project from Linux, e.g. it operates under much more restrictive memory and code-size constraints.
- For SPL, constraints are even more extreme and the devicetree is shrunk to remove unwanted nodes, or even turned into C code to avoid access overhead.

How the idea of Device tree came up?

Earlier, the hardware configuration was passed to the drivers via C structures compiled into UBOOT/kernel to tell the drivers about platform-specific settings like the address of its registers, bus speed, etc.

Why device tree won over the earlier method?

- **Reduced Maintenance overhead**
  - **Before:** Bloated U-Boot binary containing hardware info of all platforms.
  - **After**: One U-Boot binary + different DTBs = support multiple hardware configs
  - **Impact**: Dramatically reduced build/maintenance overhead for SoC vendors
- **Runtime Hardware Detection and modification**
  - **Before: Required recompilation of bootloader/kernel in cased of modifications.**
  - **After :**
    - U-Boot can detect hardware at runtime and select/modify appropriate DTB
    - **Example**: Detect RAM size, board revision, attached peripherals
    - Load different DTBs based on the detected configuration.
- **Reduced Code Duplication**
  - **Before**: Hardware config duplicated in U-Boot AND kernel
  - **After**:
    - Single source of truth (DTB) shared between both
    - Avoids Bloating of bootloader and kernel
    - Less code to maintain and fewer places for bugs

Old Way (Platform Data):

c

```c
// board/vendor/myboard/myboard.c
struct uart_platform_data uart_config = {
    .base = 0x44e09000,
    .clock = 48000000,
};
// Recompile U-Boot for any change!
```

New Way (Device Tree):

dts

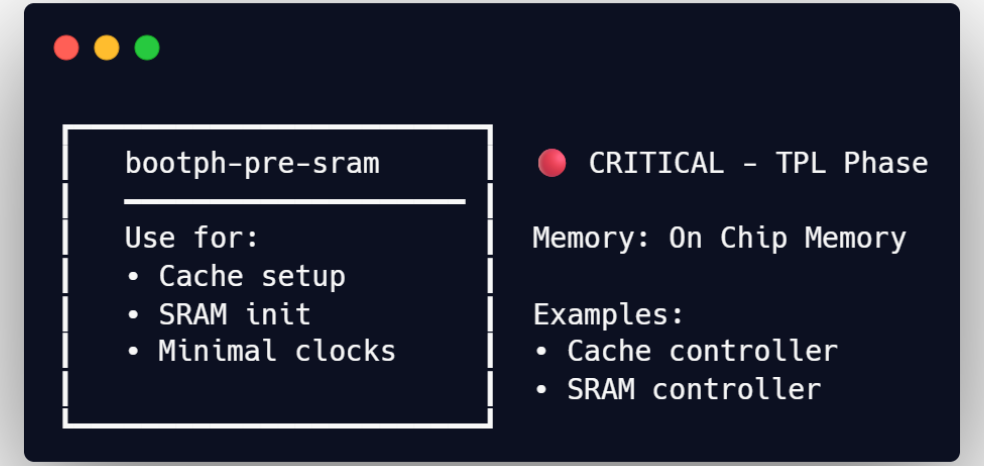
```dts
// myboard.dts
uart0: serial@44e09000 {
    compatible = "ti,omap3-uart";
    reg = <0x44e09000 0x1000>;
    clock-frequency = <48000000>;
};
// Just recompile DTB (seconds vs minutes)
```

What are bootph tags?

- **Bootph tags** are used in device trees t o manage which device nodes should b e included during the boot process.
- They help in limiting the nodes that are present in different phases of boot, ensuring that only necessary nodes are included based on the available memory.

Usage, Meaning and How they affect each boot phase

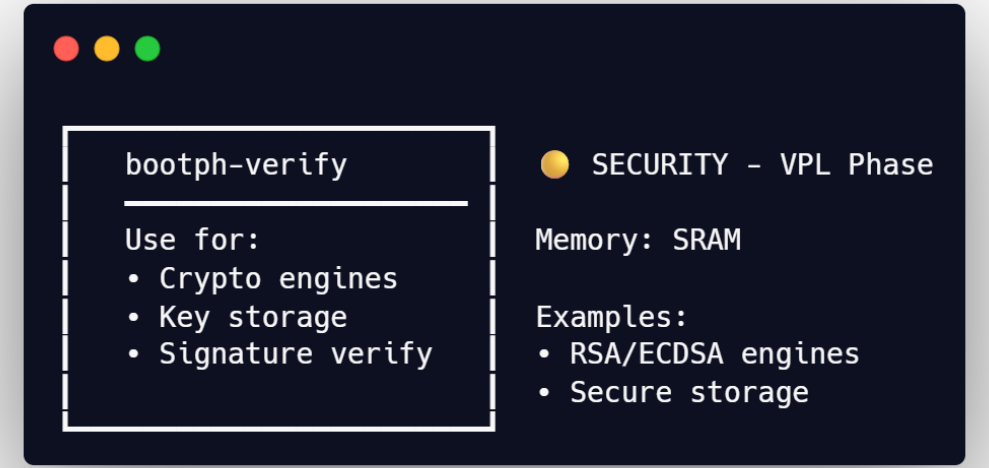
- Note: All of the bootph-tags includes the nodes in the final phase(U-boot Proper post-reloc) phase as well



Texas Instruments

Usage, Meaning and How they affect each boot phase

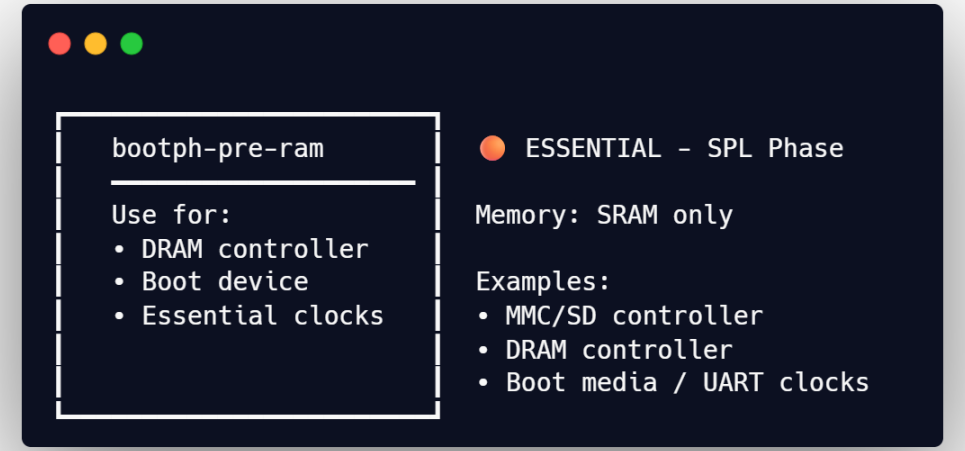
- Note: All of the bootph-tags includes the nodes in the final phase(U-boot Proper post-reloc) phase as well



Texas Instruments

Usage, Meaning and How they affect each boot phase

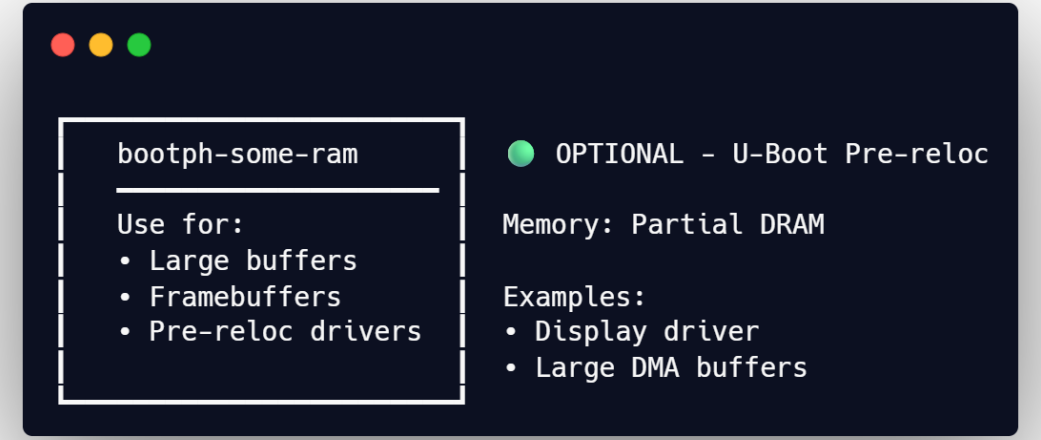
- Note: All of the bootph-tags includes the nodes in the final phase(U-boot Proper post-reloc) phase as well



Texas Instruments

Usage, Meaning and How they affect each boot phase

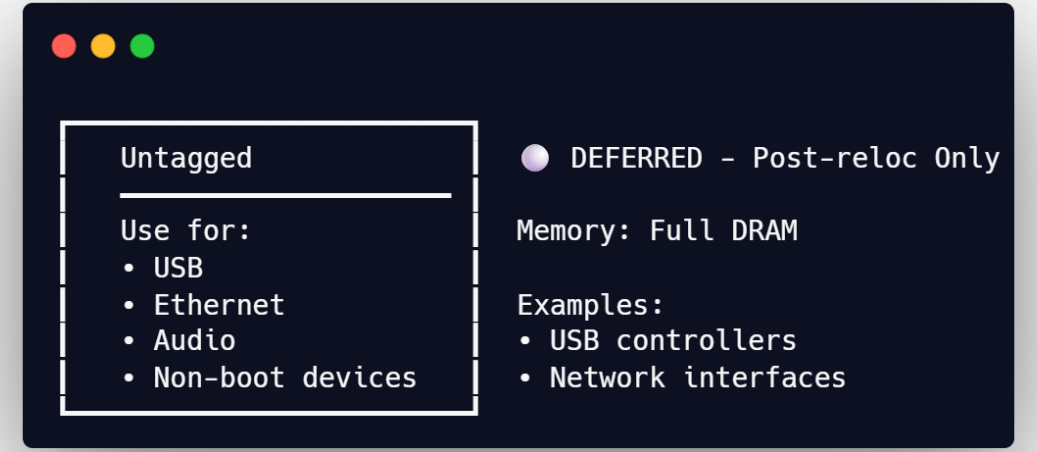
- Note: All of the bootph-tags includes the nodes in the final phase(U-boot Proper post-reloc) phase as well



Texas Instruments

Usage, Meaning and How they affect each boot phase

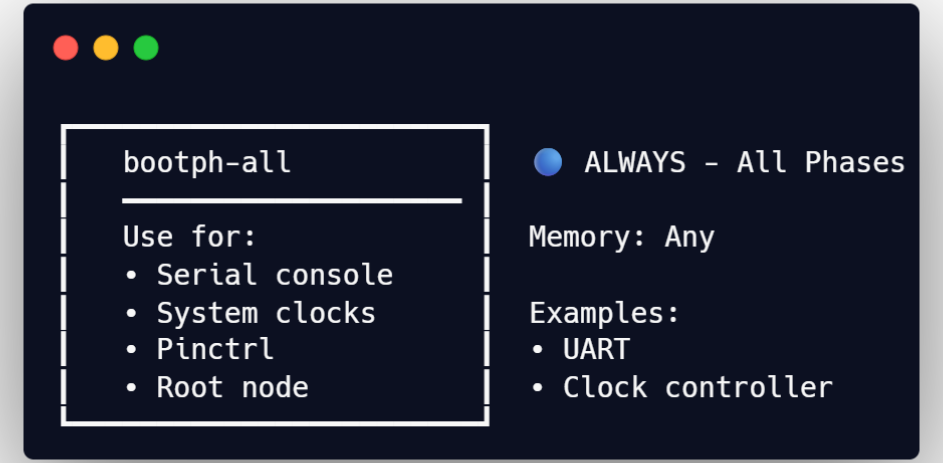
- Note: All of the bootph-tags includes the nodes in the final phase(U-boot Proper post-reloc) phase as well



Texas Instruments

Usage, Meaning and How they affect each boot phase

- Note: All of the bootph-tags includes the nodes in the final phase(U-boot Proper post-reloc) phase as well



Evolution of bootph-tags over time

Old driver model tags	New driver model tags
u-boot,dm-tpl	bootph-pre-sram
None	bootph-verify
u-boot,dm-spl	bootph-pre-ram
u-boot,dm-pre-proper	bootph-some-ram
u-boot,dm-pre-reloc	bootph-all

Why was this done?

- Since Linux accepted these bindings later on, it became non-specific to U-Boot even though U-Boot introduced it.
- Modern practice is to have single device tree source that both the bootloader and Linux kernel use.
- This means that now any project that makes use of DT can use these tags.
  - Eg: The Zephyr bootloader MCUboot can also make use of this property
- Hence, these tags were renamed to be generic.

Processing of bootph tags

- **Fdtgrep** is a utility within UBoot that allows users to perform a **greplike search** of a fla ttened device tree (fdt) file.
- The fdtgrep tools filter the U-Boot device tree during the build process to generate a much smaller device tree used in SPL (**spl/u-boot-spl.dtb**) with:
  - The mandatory nodes (/alias, /chosen, /config)
  - The nodes with one pre-relocation property: ‘bootph-all’ or ‘bootph-pre-ram’)
    - Uses the **-b** flag to find these properties.
    - Uses **-u** to ensure parent nodes are included too
- All the pre-relocation properties (‘bootph-all’, ‘bootph-pre-ram’ (SPL), ‘bootph-pre-sram’ (TPL) and ‘bootph-verify’ (VPL))
- fdtgrep is also used to remove:
  - The properties defined in **CONFIG_OF_SPL_REMOVE_PROPS**

```
# configs/am335x_evm_defconfig

CONFIG_OF_SPL_REMOVE_PROPS="interrupts interrupt-parent dmas dma-names"
```

```
Build Process:
═══════════════════════════════════════

1. Full Device Tree (Linux DT)
   arch/arm/dts/am335x-bone.dts
   ↓
   [Compile]
   ↓
   am335x-bone.dtb (100KB - all nodes)

2. U-Boot Build Process
   ↓
   [fdtgrep tool - strips nodes without bootph tags]
   ↓
   u-boot-spl.dtb (8KB - only tagged nodes)
   ↓
   Embedded in SPL binary

3. Full U-Boot DT
   u-boot.dtb (100KB - all nodes)
   ↓
   Embedded in U-Boot proper binary
```


Texas Instruments

Core code logic of bootph : TPL/SPL

```
    if (pre_reloc_only) {
        if (!ofnode_pre_reloc(node) &&
            !(entry->flags & DM_FLAG_PRE_RELOC)) {
            log_debug("Skipping device pre-relocation\n");
            return 0;
        }
    }
```

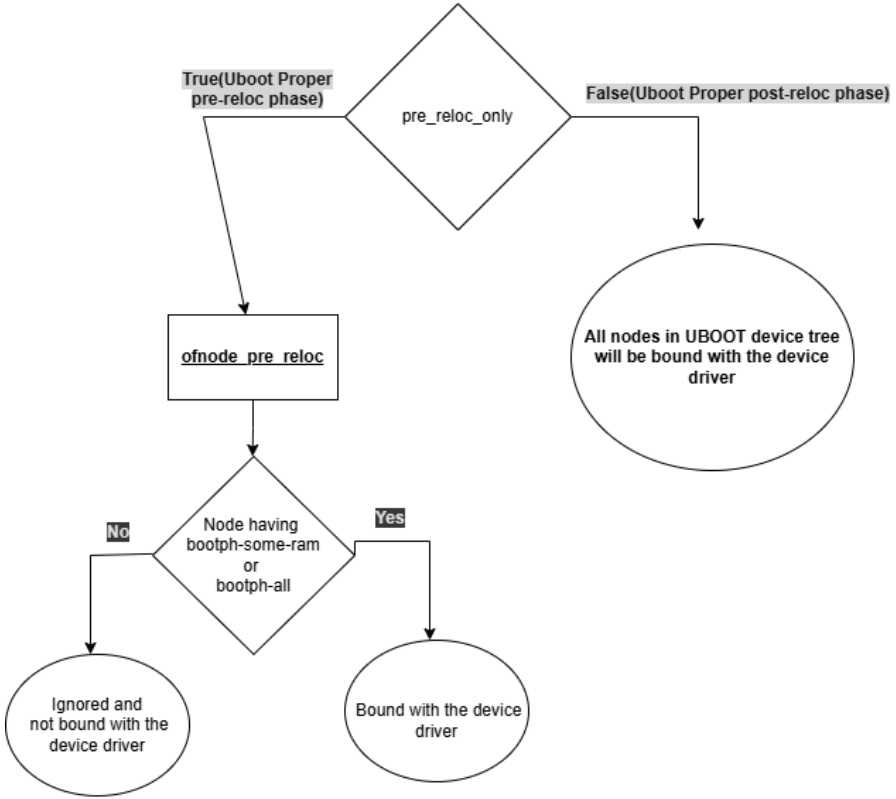
```
bool ofnode_pre_reloc(ofnode node)
{
#if defined(CONFIG_XPL_BUILD) || defined(CONFIG_TPL_BUILD)
    /* for SPL and TPL the remaining nodes after the fdtgrep 1st pass
     * had property bootph-all or bootph-pre-sram/bootph-pre-ram.
     * They are removed in final dtb (fdtgrep 2nd pass)
     */
    return true;
#else
    if (ofnode_read_bool(node, "bootph-all"))
        return true;
    if (ofnode_read_bool(node, "bootph-some-ram"))
        return true;

    /*
     * In regular builds individual spl and tpl handling both
     * count as handled pre-relocation for later second init.
     */
    if (ofnode_read_bool(node, "bootph-pre-ram") ||
        ofnode_read_bool(node, "bootph-pre-sram"))
        return gd->flags & GD_FLG_RELOC;

    return false;
#endif
}
```

- Caller of **ofnode_pre_reloc** function
- Indirectly called by board_init_f function.

- **ofnode_pre_reloc()** returns true for all nodes in TPL/SPL phase
- Why?
  - The device tree has already been filtered at build time by the **fdtgrep** tool
  - fdtgrep removes all nodes that don't have bootph-pre-ram, bootph-pre-sram, or bootph-all.
  - So by the time SPL runs, only the relevant nodes remain in the TPL/SPL DTB

Texas Instruments

Core code logic of bootph : UBOOT Proper



- **UBOOT Proper pre-relocation phase:**
  - Only Nodes with **bootph-all** and **bootph-some-ram** are included.
  - Why?
    - GD_FLAG_RELOC flag is not set
- **UBOOT Proper post-relocation phase:**
  - All nodes are bound.
  - Why?
    - **pre_reloc_only** flag is false.
      - Hence ofnode_pre_reloc function is not called.
      - Hence, device_bind_with_driver_data is called for each node which binds the node to the driver.

Texas Instruments

DRA821 : Boot time and Memory footprint comparison

BOOTFLOW	BOOT MEDIA	MEMORY(DTB)	BOOT TIME	Patch
SPL(BOOTPH-ALL)	SD	55 KB	4.4 sec	PATCH File
SPL(Optimized)	SD	9.4 KB(85% reduced)	2.34 sec(~50% reduced)	PATCH File PATCH File
SPL(BOOTPH-ALL)	eMMC	55 KB	3.46 sec	PATCH File
SPL(Optimized)	eMMC	9.3 KB(85% reduced)	1.79 sec(~50% reduced)	PATCH File PATCH File
SPL(BOOTPH-ALL)	OSPI NOR	55 KB	3.42 sec	PATCH File
SPL(Optimized)	OSPI NOR	9.6 KB(85% reduced)	1.75 sec(~50% reduced)	PATCH File PATCH File

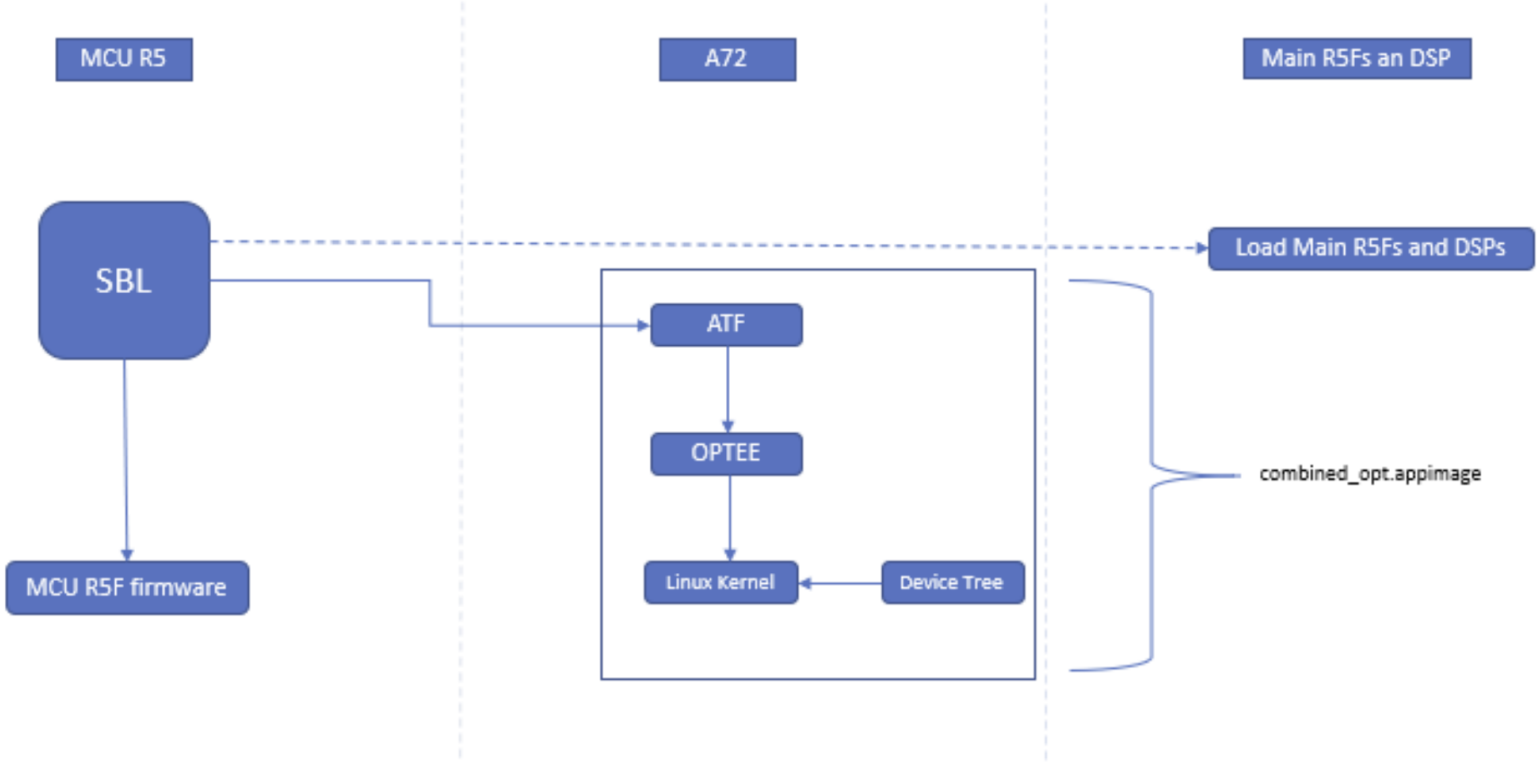
SHA ID: U-Boot 2026.04- 0a7e2a90ad4df

DRA821U : How this optimality was achieved?

- Bootph-pre-ram
  - MCU Domain Timer
  - MCU Domain Secure Proxy
  - MCU Domain Temperature Sensor
  - MCU Domain ESM
  - Boot media from which UBOOT is booted
- Bootph-all
  - Clock controller
  - Power controller
  - Pinmux
  - Serial console
- Bootph-some-ram
  - Display and Remote-proc driver nodes
- Rest of the nodes untagged(included only in final phase of UBOOT(uboot post-reloc phase)

Texas Instruments

RTOS BOOTLOADER BOOTFLOW



Texas Instruments

DRA821 : RTOS vs UBOOT bootloader :Boot time comparison

BOOTLOADER	BOOT MEDIA	BOOT TIME
RTOS BOOTLOADER	eMMC	1.981 sec
UBOOT(default upstream)	eMMC	2.14 sec
UBOOT(Optimized)	eMMC	1.79 sec

SHA ID: U-Boot 2026.04- 0a7e2a90ad4df

Texas Instruments

Lessons Learned: Parent-Child Dependency for bootph.

– Parent and child have a different scope

```
i2c_emul: emul {
   bootph-pre-ram;
   reg = <0xff>;
   compatible = "sandbox,i2c-emul-parent";
   emul0: emul0 {
      bootph-all;
      compatible = "sandbox,i2c-rtc-emul";
      #emul-cells = <0>;
   };
};
```

```
Node '/i2c@0/emul/emul0' requires parent node '/i2c@0/emul' but it is not in
   the valid list
```


Texas Instruments

Parent-Child Dependency for bootph.

- Child has a scope but parent does not
  - The bootph of child gets propagated to the parent.

```
ethernet@46000000 {
        compatible = "ti,j721e-cpsw-nuss";
        #address-cells = <0x02>;
        #size-cells = <0x02>;
        reg = <0x00 0x46000000 0x00 0x200000>;
        reg-names = "cpsw_nuss";
        ranges = <0x00 0x00 0x00 0x46000000 0x00 0x200000>;
        dma-coherent;
        clocks = <0x07 0x12 0x15>;
        clock-names = "fck";
        power-domains = <0x08 0x12 0x01>;
        dmas = <0x10 0xf000 0x10 0xf001 0x10 0xf002 0x10 0xf003 0x10 0xf004 0x10 0xf
005 0x10 0xf006 0x10 0xf007 0x10 0x7000>;
        dma-names = "tx0\0tx1\0tx2\0tx3\0tx4\0tx5\0tx6\0tx7\0rx";
        pinctrl-names = "default";
        pinctrl-0 = <0x11 0x12>;
        phandle = <0x85>;

        ethernet-ports {
                #address-cells = <0x01>;
                #size-cells = <0x00>;
        };
};
```

Decompiled SPL DTB

```
mcu_cpsw: ethernet@46000000 {
    compatible = "ti,j721e-cpsw-nuss";
    #address-cells = <2>;
    #size-cells = <2>;
    reg = <0x00 0x46000000 0x00 0x200000>;
    reg-names = "cpsw_nuss";
    ranges = <0x00 0x00 0x00 0x46000000 0x00 0x200000>;
    dma-coherent;
    clocks = <&k3_clks 18 21>;
    clock-names = "fck";
    power-domains = <&k3_pds 18 TI_SCI_PD_EXCLUSIVE>;

    dmas = <&mcu_udmap 0xf000>,
           <&mcu_udmap 0xf001>,
           <&mcu_udmap 0xf002>,
           <&mcu_udmap 0xf003>,
           <&mcu_udmap 0xf004>,
           <&mcu_udmap 0xf005>,
           <&mcu_udmap 0xf006>,
           <&mcu_udmap 0xf007>,
           <&mcu_udmap 0x7000>;
    dma-names = "tx0", "tx1", "tx2", "tx3",
            "tx4", "tx5", "tx6", "tx7",
            "rx";

    ethernet-ports {
        #address-cells = <1>;
        #size-cells = <0>;
        bootph-all;
        cpsw_port1: port@1 {
            reg = <1>;
            ti,mac-only;
            label = "port1";
            ti,syscon-efuse = <&mcu_conf 0x200>;
            phys = <&phy_gmii_sel 1>;
        };
    };

    davinci_mdio: mdio@f00 {
        compatible = "ti,cpsw-mdio","ti,davinci_mdio";
        reg = <0x00 0xf00 0x00 0x100>;
        #address-cells = <1>;
        #size-cells = <0>;
```

Device tree source(dts)

Texas Instruments

Parent-Child Dependency for bootph.

- Parent has a scope but child does not
  - Child gets excluded

```
ethernet@46000000 {
        compatible = "ti,j721e-cpsw-nuss";
        #address-cells = <0x02>;
        #size-cells = <0x02>;
        reg = <0x00 0x46000000 0x00 0x200000>;
        reg-names = "cpsw_nuss";
        ranges = <0x00 0x00 0x00 0x46000000 0x00 0x200000>;
        dma-coherent;
        clocks = <0x07 0x12 0x15>;
        clock-names = "fck";
        power-domains = <0x08 0x12 0x01>;
        dmas = <0x10 0xf000 0x10 0xf001 0x10 0xf002 0x10 0xf003 0x10 0xf004 0x10 0xf00
xf006 0x10 0xf007 0x10 0x7000>;
        dma-names = "tx0\0tx1\0tx2\0tx3\0tx4\0tx5\0tx6\0tx7\0rx";
        pinctrl-names = "default";
        pinctrl-0 = <0x11 0x12>;
        phandle = <0x85>;
};
```

Decompiled SPL DTB

```
mcu_cpsw: ethernet@46000000 {
    compatible = "ti,j721e-cpsw-nuss";
    #address-cells = <2>;
    #size-cells = <2>;
    reg = <s0x00 0x46000000 0x00 0x200000>;      You, now • Uncommitted changes
    reg-names = "cpsw_nuss";
    ranges = <0x00 0x00 0x00 0x46000000 0x00 0x200000>;
    dma-coherent;
    clocks = <&k3_clks 18 21>;
    clock-names = "fck";
    power-domains = <&k3_pds 18 TI_SCI_PD_EXCLUSIVE>;
    bootph-all;

    dmas = <&mcu_udmap 0xf000>,
           <&mcu_udmap 0xf001>,
           <&mcu_udmap 0xf002>,
           <&mcu_udmap 0xf003>,
           <&mcu_udmap 0xf004>,
           <&mcu_udmap 0xf005>,
           <&mcu_udmap 0xf006>,
           <&mcu_udmap 0xf007>,
           <&mcu_udmap 0x7000>;
    dma-names = "tx0", "tx1", "tx2", "tx3",
            "tx4", "tx5", "tx6", "tx7",
            "rx";

    ethernet-ports {
        #address-cells = <1>;
        #size-cells = <0>;

        cpsw_port1: port@1 {
            reg = <1>;
            ti,mac-only;
            label = "port1";
            ti,syscon-efuse = <&cpsw_mac_syscon 0x0>;
            phys = <&phy_gmii_sel 1>;
        };
    };
```

Device tree source(dts)

Texas Instruments

Parent-Child Dependency for bootph.

- Both parent and child nodes is untagged
- Parent and child node will only be included in the post relocation phase of U-BOOT-proper.

```
mcu_cpsw: ethernet@46000000 {
    compatible = "ti,j721e-cpsw-nuss";
    #address-cells = <2>;
    #size-cells = <2>;
    reg = <0x00 0x46000000 0x00 0x200000>;
    reg-names = "cpsw_nuss";
    ranges = <0x00 0x00 0x00 0x46000000 0x00 0x200000>;
    dma-coherent;
    clocks = <&k3_clks 18 21>;
    clock-names = "fck";
    power-domains = <&k3_pds 18 TI_SCI_PD_EXCLUSIVE>;

    dmas = <&mcu_udmap 0xf000>,
           <&mcu_udmap 0xf001>,
           <&mcu_udmap 0xf002>,
           <&mcu_udmap 0xf003>,
           <&mcu_udmap 0xf004>,
           <&mcu_udmap 0xf005>,
           <&mcu_udmap 0xf006>,
           <&mcu_udmap 0xf007>,
           <&mcu_udmap 0x7000>;
    dma-names = "tx0", "tx1", "tx2", "tx3",
            "tx4", "tx5", "tx6", "tx7",
            "rx";

    ethernet-ports {
        #address-cells = <1>;
        #size-cells = <0>;
        cpsw_port1: port@1 {
            reg = <1>;
            ti,mac-only;
            label = "port1";
            ti,syscon-efuse = <&mcu_conf 0x200>;
            phys = <&phy_gmii_sel 1>;
        };
    };

    davinci_mdio: mdio@f00 {
        compatible = "ti,cpsw-mdio","ti,davinci_mdio";
        reg = <0x00 0xf00 0x00 0x100>;
        #address-cells = <1>;
        #size-cells = <0>;
        clocks = <&k3_clks 18 21>;
```

Device tree source(dts)

Texas Instruments

Common Pitfalls

- Adding tags in the wrong way causing it to get overridden
  - **Case 1(Same dts file)**: Adding different tags to the same node one after the other in the same file.
- **Output** : Set Union of both tags

```
278  &mcu_uart0 {
279      status = "okay";
280      pinctrl-names = "default";
281      pinctrl-0 = <&mcu_uart0_pins_default>;
282      bootph-some-ram;
283  };
284
285  &mcu_uart0{
286      bootph-all;
287  };
288      Tom Rini, 2 years ago • Squashed 'dts/upstream/' content
```

Example 1(dts)

```
serial@40a00000 {
        compatible = "ti,j721e-uart\0ti,am654-uart";
        reg = <0x00 0x40a00000 0x00 0x100>;
        clock-frequency = <0x5b8d800>;
        power-domains = <0x08 0x95 0x01>;
        clocks = <0x07 0x95 0x02>;
        clock-names = "fclk";
        status = "okay";
        pinctrl-names = "default";
        pinctrl-0 = <0x0a>;
        phandle = <0x81>;
};
```

Decompiled SPL DTB

```
&mcu_uart0 {
    status = "okay";
    pinctrl-names = "default";
    pinctrl-0 = <&mcu_uart0_pins_default>;
    bootph-some-ram;
};

&mcu_uart0{
    bootph-pre-ram;
};      Tom Rini, 2 years ago • Squashed 'dts/upstream/' content fro
```

Example 2(dts)

Texas Instruments

Common Pitfalls

- Adding tags in the wrong device tree file causing it to get overridden
  - **Case 2(Different file)**: Adding one tag in the dtsi and another tag in the dts causing it to be overridden
- **Output** : Set Union of both tags

<board>-u-boot.dtsi

```
k3-j7200-r5.dtsi | k3-j7200-common-proc-board-u-boot.dtsi M
h > arm > dts > k3-j7200-common-proc-board-u-boot.dtsi
    &main_i2c0 {
    };

    &main_i2c0_pins_default {
        bootph-all;
    };

    &exp1 {
        bootph-all;
    };

    &exp2 {
        bootph-all;
    };

    &mcu_uart0{
        bootph-some-ram;
    };
```

mcu_uart0 Aa ab .* 1 of 2

```
serial@40a00000 {
        compatible = "ti,j721e-uart\0ti,am654-uart";
        reg = <0x00 0x40a00000 0x00 0x100>;
        clock-frequency = <0x5b8d800>;
        power-domains = <0x08 0x95 0x01>;
        clocks = <0x07 0x95 0x02>;
        clock-names = "fclk";
        status = "okay";
        pinctrl-names = "default";
        pinctrl-0 = <0x0a>;
        phandle = <0x81>;
```

Decompiled SPL DTB

k3-j7200-common-proc-board.dts | k3-j7200-som-p0.dtsi | k3-j7200-

dts > upstream > src > arm64 > ti > k3-j7200-common-proc-board.dts

```
276     };
277
278     &mcu_uart0 {
279         status = "okay";
280         pinctrl-names = "default";
281         pinctrl-0 = <&mcu_uart0_pins_default>;
282         bootph-all;
283     };
284         Tom Rini, 2 years ago • Squashed 'dts/upstream/'
```

<board>.dts

Texas Instruments

Common Pitfalls

- **Ignoring tags**
  - Without tags, nodes are included only in the final phase, where all memory is available.
- **Using bootph-all tags everywhere**

DTB	SIZE(KB)
UPSTREAM DRA821 SPL DTB	15.14
DRA821 SPL DTB (BOOTPH-ALL)	54.96
UPSTREAM DRA821 U-BOOT DTB	77.14
DRA821 UBOOT DTB (BOOTPH-ALL)	79.86

Generic steps the vendor can use to profile/audit their bootph usage

- Identify the device tree nodes required in each phase of the bootloader and accordingly tag them.
- Avoid over tagging a device tree node.
- Ensure that you understand the parent-child dependency

Texas Instruments

Other possible Optimizations : Compiled-in Device Tree

- This feature converts the devicetree contents into C code at build time , which can be compiled into the SPL binary.
- **How does it optimize?**
  - No need to change the source code.
  - Avoids runtime parsing of device tree.
  - This saves the overhead of storing **device tree code** and the **libfdt library code**(~3-10KB), thus leading to more efficient storage of data.
  - Device is instantiated at runtime without converting the device tree data(in the form of C structures) to platform data in the driver, thus reducing the code overhead of that conversion logic as well. The *of_to_plat()* method should therefore do nothing in such a driver.

Texas Instruments

Other possible Optimizations : Compiled-in Device Tree

- **How it works?**
  - The feature is enabled by CONFIG OF_PLATDATA
    - **This feature is only available in SPL/TPL**
  - A tool called '**dtoc**' converts a devicetree file into the following:
    - A set of struct declarations, one for each compatible node
    - The actual platform data for each device.
    - A set of ***U_BOOT_DRVINFO()*** declarations for each node.
  - The device is then instantiated at run-time and the platform data can be accessed using:
  - The platform data is then matched with a driver at runtime, the 'name' property of the ***U_BOOT_DRVINFO()*** declaration has to match a driver declared via ***U_BOOT_DRIVER()/ DM_DRIVER_ALIAS()****macro*.

Texas Instruments

```
sdmmc: dwmmc@ff0c0000 {
        compatible = "rockchip,rk3288-dw-mshc";
        clock-freq-min-max = <400000 150000000>;
        clocks = <&cru HCLK_SDMMC>, <&cru SCLK_SDMMC>,
                 <&cru SCLK_SDMMC_DRV>, <&cru SCLK_SDMMC_SAMPLE>;
        clock-names = "biu", "ciu", "ciu_drv", "ciu_sample";
        fifo-depth = <0x100>;
        interrupts = <GIC_SPI 32 IRQ_TYPE_LEVEL_HIGH>;
        reg = <0xff0c0000 0x4000>;
        bus-width = <4>;
        cap-mmc-highspeed;
        cap-sd-highspeed;
        card-detect-delay = <200>;
        disable-wp;
        num-slots = <1>;
        pinctrl-names = "default";
        pinctrl-0 = <&sdmmc_clk>, <&sdmmc_cmd>, <&sdmmc_cd>, <&sdmmc_bus4>;
            vmmc-supply = <&vcc_sd>;
            status = "okay";
            bootph-all;
    };
```

```
struct dtd_rockchip_rk3288_dw_mshc {
        fdt32_t         bus_width;
        bool            cap_mmc_highspeed;
        bool            cap_sd_highspeed;
        fdt32_t         card_detect_delay;
        fdt32_t         clock_freq_min_max[2];
        struct phandle_1_arg clocks[4];
        bool            disable_wp;
        fdt32_t         fifo_depth;
        fdt32_t         interrupts[3];
        fdt32_t         num_slots;
        fdt32_t         reg[2];
        fdt32_t         vmmc_supply;
};
```

```
/* Node /dwmmc@ff0c0000 index 2 */
static struct dtd_rockchip_rk3288_dw_mshc dtv_dwmmc_at_ff0c0000 = {
        .fifo_depth             = 0x100,
        .cap_sd_highspeed       = true,
        .interrupts             = {0x0, 0x20, 0x4},
        .clock_freq_min_max     = {0x61a80, 0x8f0d180},
        .vmmc_supply            = 0xb,
        .num_slots              = 0x1,
        .clocks                 = {{0, 456},
                                   {0, 68},
                                   {0, 114},
                                   {0, 118}},
        .cap_mmc_highspeed      = true,
        .disable_wp             = true,
        .bus_width              = 0x4,
        .u_boot_dm_pre_reloc    = true,
        .reg                    = {0xff0c0000, 0x4000},
        .card_detect_delay      = 0xc8,
};
```

```
U_BOOT_DRVINFO(dwmmc_at_ff0c0000) = {
        .name           = "rockchip_rk3288_dw_mshc",
        .plat       = &dtv_dwmmc_at_ff0c0000,
        .plat_size  = sizeof(dtv_dwmmc_at_ff0c0000),
        .parent_idx     = -1,
};
```


Texas Instruments

Other possible Optimizations : Live Tree

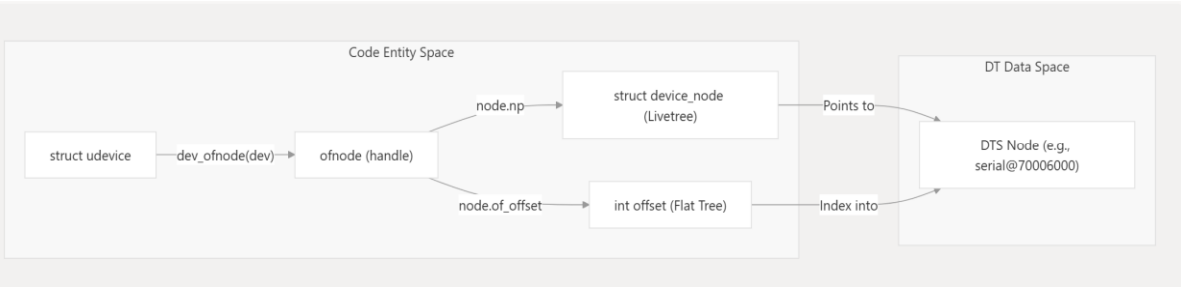
- An alternative to Flat device tree in UBOOT proper post relocation phase
- An unpacked, hierarchical pointer-based structure (similar to Linux).
- This is enabled via **CONFIG_OF_LIVE**
- **Use case**:
  - Run time modification of device tree

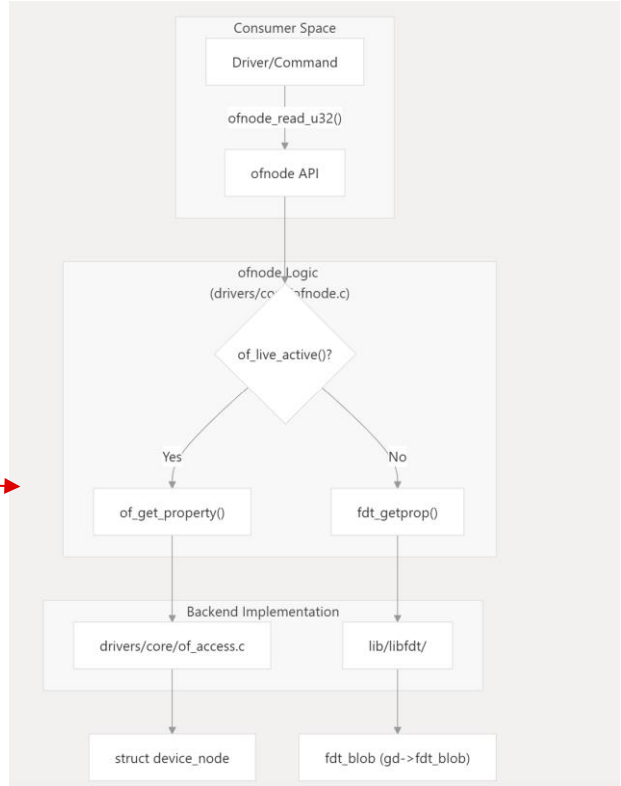
```
struct udevice *bus;
const void *blob = gd->fdt_blob;
int node = dev_of_offset(bus);

i2c_bus->regs = (struct i2c_ctlr *)devfdt_get_addr(dev);
plat->frequency = fdtdec_get_int(blob, node, "spi-max-frequency", 500000);
```

```
struct udevice *bus;

i2c_bus->regs = dev_read_addr_ptr(dev);
plat->frequency = dev_read_u32_default(bus, "spi-max-frequency", 500000);
```

Other possible Optimizations : Live Tree

```
struct udevice *bus;

i2c_bus->regs = dev_read_addr_ptr(dev);
plat->frequency = dev_read_u32_default(bus, "spi-max-frequency", 500000);
```





Texas Instruments

Other possible Optimizations : Live Tree

- How this feature helps:
  - Adding new properties can involve copying large amounts of data around to make room.
  - The overall tree has a fixed maximum size so sometimes the tree must be rebuilt in a new location to create more space.
  - Even if not adding new properties or nodes, scanning the tree can be slow.
    - For example, finding the parent of a node is a slow process. Reading from nodes involves a small amount parsing which takes a little time.

Texas Instruments

Reference links

- **UBOOT Bootflow:**
  - https://docs.u-boot.org/en/stable/usage/spl_boot.html
  - https://deepwiki.com/u-boot/u-boot/3-boot-process
  - Memory Management — Das U-Boot unknown version documentation
- **Device tree:**
  - https://docs.u-boot.org/en/latest/develop/driver-model/design.html#:~:text=tree)%20and%20probe.-,Platform%20Data,%EF%83%81,-Note%3A%20platform%20data
  - https://docs.u-boot.org/en/latest/develop/driver-model/design.html#device-tree
  - Devicetree Control in U-Boot — Das U-Boot unknown version documentation
- **Compiled in Device tree**
  - Compiled-in Device Tree / Platform Data — Das U-Boot unknown version documentation

Texas Instruments

Credits and Acknowledgement

- Texas Instruments Inc.
- The Linux Foundation.

E-mail:

- Keerthy J <j-keerthy@ti.com>
- Gokul Praveen <g-praveen@ti.com>
- Beleswar Padhi <b-padhi@ti.com>