Device Tree (DTS), Linux Board Bring up and Kernel version changing on Embedded systems, a review of some lessons learned processors

Schuyler Patton SW Applications Sitara™ Processors
Texas Instruments
Agenda

• Who am I
• Linux bring up on a new derivative board based on an existing board.
• Discuss boot flow and what does Linux need to boot on a board
• Discuss the Board DTS File and its components
• Introduce the “hello world” concept
• Creating and booting a hello_world.dts
• Lifecycle of the LTS kernel
Schuyler Patton

• Have been working on embedded software with Texas Instruments for 20+ years

• Currently a member of the Sitara Processors Linux Applications team and have been for the past 11 years

• Supporting Linux, networking and board porting on TI SOCs
Target audience for this discussion

• Specifically would like to address this presentation to those new to Linux and are doing their first board port.

• Anybody interesting in looking at how Linux is “Bound” to an embedded processor on a board to make an embedded system. I will just call the embedded system the board from here.
Level set of the discussion

• This discussion will be about a method to simplify the bring up process for a new board or kernel.

• This is about getting to a minimal number of elements to get a basic boot to a prompt.

• An applications engineer put this presentation together, not a kernel developer. This distinction is sometimes blurred though.

• Perhaps another time and presentation we can discuss additional strategies that the new board is working as intended.
Device Tree Files

• This discussion will be discussing device tree files. This presentation is written with the assumption that the audience has a little bit of background in what the DTS file is.

• For a background on Device Tree please look at the tutorial presentation from Free Electrons which is awesome

The new derivative board based on an existing board relationship for Linux bring up
New Derivative Board Based On An Existing Board

New Board Based On An Existing Board

New Additional Peripherals on new board

- PHY
- RJ45
- UART
- UART

Remove these Peripherals on new board

- HDMI Framer
- HDMI

Existing Board for basing a new design on

The arch/arm/boot/dts/ directory is where the 32bit board dts files are stored

- Leveraging a known good for the new derivative design
- Several Board Vendors shown here.
- This is just a sample of the boards in the directory.
- Going to start with the DTS file of the reference board

The arch/arm64/boot/dts/ directory is where the 64bit board dts files are stored
Board dts File – How do you start?

- Will the reference DTS just drop in?
- Completely start over?
- Going to start with the DTS file of the reference board

Reasons for hello_world dts vs. full board dts

• Developing a minimal working DTS on a known platform eliminates the “is it the board or the DTS debate”
• Board bring up can be challenging so with a functioning minimal set baseline you can concentrate in small steps and will make the bring up process simpler
• By booting to a minimal interface set allows a base platform to use Linux utilities to help with debug
• Debugging a DTS that is not semantically correct is a pain.

• Upgrading kernels will be easier with a minimal baseline DTS as the processor.dtsi file most likely will have significant changes between releases. Just because the DTS file compiles does not mean the kernel will boot cleanly.
What initial success looks like

- Booting to a prompt on the console with a minimal DTS.
- Not all the peripherals have to be enabled for the developer to prove that Linux can be booted on the new board.
Quick Review, booting Linux

RBL  SPL  U-Boot  Kernel
Elements needed for a board to boot Linux

SOC
- peripheral
- peripheral
- peripheral
- peripheral
- peripheral
- peripheral
- peripheral
- peripheral
- peripheral
- peripheral

processor

Boot Loader
Linux Kernel
Root Filesystem
Board DTB
Board state as the boot loader launches Linux

- For this discussion we are expecting that the boot loader (typically U-Boot) has been ported to the board and run and setup a minimal configuration.

- A minimal configuration is that there is power, the processor is operating at a performance point and that DDR has been configured.

- The boot loader loads the Kernel Image and Board DTB to DDR

Board state as the boot loader launches Linux

DTS file is what “binds” Linux

Devicetree is a data structure describing the hardware components of a particular computer so that the operating system's kernel can use and manage those components, including the CPU or CPUs, the memory, the buses and the peripherals.


New (Derivative) Board Based On An Existing Board

Board DTS

/dts-v1/;
#include "processor.dtsi"
#include "base_board_common.dtsi"
#include <...>
.
.
#include <...>
/
{
    model = "Board Base Design X1";
.
.
}
**Processor dtsi File**

- Contains the definition for the entire SOC.
- Full entitlement of the SOC, all on chip peripherals defined here.

[Diagram of processor components]

---

```
#include "processor.dtsi"
```

---

```
/model = "Board Base Design X1";
```

---

Processor dtshi File – Processor Architecture

<table>
<thead>
<tr>
<th>ARM® Cortex™-A8</th>
<th>Up to 1 Ghz</th>
</tr>
</thead>
<tbody>
<tr>
<td>32KB and 32KB L1 + SED</td>
<td></td>
</tr>
<tr>
<td>256KB L2 + ECC</td>
<td></td>
</tr>
<tr>
<td>176KB ROM 64KB RAM</td>
<td></td>
</tr>
</tbody>
</table>
Processor dtsi File – SOC internal modules

Processor dtsi File – Board Binding

DTS File – Binding a Peripheral to a board

• What is the binding process?

• All these signals in the block diagram need to be accounted for for the driver to function.

• Is the board port developer responsible to identify “all” the settings? No
DTS File – Binding a Peripheral to a board

• Components of a DTS File

Arch

SOC

Board

am33xx.dtsi

dt-bindings/gpio/gpio.h
dt-bindings/pinctrl/am33xx.h
skeleton.dtsi
am33xx-clocks.dtsi

am335x-boneblack.dts
am335x-bone-common.dtsi
Hello World DTS File
DTS file “binds” Linux to a board

Devicetree is a data structure describing the hardware components of a particular computer so that the operating system's kernel can use and manage those components, including the CPU or CPUs, the memory, the buses and the peripherals.


The Hello World DTS File

```c
main() {
    printf("hello, world\n");
}
```

```
/dts-v1/;
#include "processor.dtsi"
/
    model = "Hello World";
/;
&uart
&mwc...
```

Processor dtisi File – Board Binding

The Hello World DTS File

existing_reference_board.dts

```
/dts-v1/;
#include "processor.dtsi"
#include "board-type-common.dtsi"
#include "base-board-common.dtsi"
#include "another-common.dtsi"

/ {
  model = "Hello World";
}

&uart
&mmc
&emac
&usb
...
```

hello_world.dts

```
/dts-v1/;
#include "processor.dtsi"

/ {
  model = "Hello World";
}

&uart
&mmc
... 
```

Going to draw on reference DTS to make the hello_world.dts
Building the DTS file to a DTB file (blob)

• Using the dtbs build target as part of the kernel make system

```bash
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- dtbs
CALL scripts/checksyscalls.sh
DTC arch/arm/boot/dts/hello_world.dtb
```

• Debug tip, how to “un-compile” the DTS file. The reason is that just because it compiles does not mean it will work. At a minimum this step could reveal incorrect nesting, over written values.

```bash
dtc hello_world.dtb -O dts -o hello_world_as_compiled.dts
```
Adding new file to the DTB build

- Add your file to the Makefile in the config area of the board type you based your board on.

The directory for the Makefile is located here:
(32bit systems) arch/arm/boot/dts
(64bit systems) arch/arm64/boot/dts
Where is the DTB file stored?

- The /boot directory in the root filesystem for the board holds the DTB for the board.

- For this class of SOC U-Boot uses an eeprom to identify the board so the DTB associated with the board can be loaded with the kernel at boot time.

- For this test the hello_world.dtb was copied over the am335x_boneblack.dtb to make the development of the new DTB file.
How to make an Hello World DTS

- Ideal case is to develop the hello_world.dts on the original base board so you are working from a known good.

- Iteratively remove elements until the boot fails...

- Example here used renamed copies of include files.
How to make an Hello World DTS

- Red box shows isolated node that was not needed to get to a basic boot.
- Green shows what was still needed after hello world DTS was created.
- Process was iterative
HW DTS File

- “Hello World” like minimal board DTS File
- Define the model, memory
- Voltage regulator for SD card for root file system
- UART node and supporting pin mux
• mmc1 Pin Mux
• UART node
• mmc1 node

```dts
mmc1_pins: pinmux_mmc1_pins {
  pinctrl-single,pins = <
    AM33XX_IOPAD(0x960, PIN_INPUT | MUX_MODE7) /* spio0_cs1.gpio0_6 */
    AM33XX_IOPAD(0x8fc, PIN_INPUT_PULLUP | MUX_MODE0) /* mmc0_dat0.mmc0_dat0 */
    AM33XX_IOPAD(0x8f8, PIN_INPUT_PULLUP | MUX_MODE0) /* mmc0_dat1.mmc0_dat1 */
    AM33XX_IOPAD(0x8f4, PIN_INPUT_PULLUP | MUX_MODE0) /* mmc0_dat2.mmc0_dat2 */
    AM33XX_IOPAD(0x8f0, PIN_INPUT_PULLUP | MUX_MODE0) /* mmc0_dat3.mmc0_dat3 */
    AM33XX_IOPAD(0x904, PIN_INPUT_PULLUP | MUX_MODE0) /* mmc0_cmd.mmc0_cmd */
    AM33XX_IOPAD(0x900, PIN_INPUT_PULLUP | MUX_MODE0) /* mmc0_clk.mmc0_clk */
>
}

&uart0 {
  pinctrl-names = "default";
  pinctrl-0 = <&uart0_pins>;
  status = "okay";
}

&mmc1 {
  status = "okay";
  bus-width = <0x4>;
  pinctrl-names = "default";
  pinctrl-0 = &mmc1_pins;
  cd-gpios = <&gpio0 6 GPIO_ACTIVE_LOW>;
  vmmc-supply = <&vmmcsd_fixed>;
}
```
Blank Slate, let’s build a “Hello World” like minimal board DTS File, just want to get to a prompt on the console.

```
/dts-v1/
```
HW DTS File

- “Hello World” like minimal board DTS File
- The first is the defining the Arch/SOC for the custom board. What is the processor the board is based on

```
#include "am33xx.dtsi"
```
HW DTS File

• “Hello World” like minimal board DTS File

• Defining the root node model, compatibility, memory....

```c
/dts-v1/;
#include "am33xx.dtsi"

/ {
  model = "TI AM335x Hello World ";
  compatible = "ti,am335x-bone-black", "ti,am335x-bone", "ti,am33xx";

  memory@80000000 {
    device_type = "memory";
    reg = <0x80000000 0x10000000>; /* 256 MB */
  };

  chosen {
    stdout-path = &uart0;
  };

  vmmcsd_fixed: fixedregulator0 {
    compatible = "regulator-fixed";
    regulator-name = "vmmcsd_fixed";
    regulator-min-microvolt = <33000000>;
    regulator-max-microvolt = <33000000>;
  }
};
```
For SOC that use pin muxes - what is a pin mux?

• Most pins on the SOC have a mux must be set to enable a peripheral access
• Each pin name has several signal names that can accessed by a mux mode

• In this example J18 is using mux mode 1 which is the DCAN0 TX signal
• The silicon IP in the DTS has to be connected to the pin
HW DTS File

• “Hello World” like minimal board DTS File

• Defining the pin mux for a peripheral, in this case UART

• On SOCs that use a pin mux for signal routing this is a critical step.

```c
&am33xx_pinmux {
    uart0_pins: pinmux_uart0_pins {
        pinctrl-single,pins = <
            AM33XX_IOPAD(0x970, PIN_INPUT_PULLUP | MUX_MODE0) /*uart0_rxd.uart0_rxd*/
            AM33XX_IOPAD(0x974, PIN_OUTPUT_PULLDOWN | MUX_MODE0) /*uart0_txd.uart0_txd*/
        >;
    };
    mmc1_pins: pinmux_mmc1_pins {
        pinctrl-single,pins = <
            AM33XX_IOPAD(0x960, PIN_INPUT | MUX_MODE7)        /* spio0_cs1.gpio0_6 */
            AM33XX_IOPAD(0x8fc, PIN_INPUT_PULLUP | MUX_MODE0) /* mmc0_dat0.mmc0_dat0 */
            AM33XX_IOPAD(0x8f8, PIN_INPUT_PULLUP | MUX_MODE0) /* mmc0_dat1.mmc0_dat1 */
            AM33XX_IOPAD(0x8f4, PIN_INPUT_PULLUP | MUX_MODE0) /* mmc0_dat2.mmc0_dat2 */
            AM33XX_IOPAD(0x8f0, PIN_INPUT_PULLUP | MUX_MODE0) /* mmc0_dat3.mmc0_dat3 */
            AM33XX_IOPAD(0x904, PIN_INPUT_PULLUP | MUX_MODE0) /* mmc0_cmd.mmc0_cmd */
            AM33XX_IOPAD(0x900, PIN_INPUT_PULLUP | MUX_MODE0) /* mmc0_clk.mmc0_clk */
        >;
    };
};
```
HW DTS File

• “Hello World” like minimal board DTS File

• Finally enable the nodes with information specific to the new board

• Notice “&” for uart0, this means information is being appended to the uart0 node that is defined in the processor dtsi. Recommend to never modify the processor dtsi to add board binding information like what is shown here
HW DTS File – What binding was needed

- UART node in the DTS File
- UART node disassembled from the DTB file, significant difference. DTS only needs to define the pinmux and set status to "okay" in this example.
- Remember the debug tip from earlier about reverse compiling the DTB file.

```c
&uart0 {
    pinctrl-names = "default";
    pinctrl-0 = &uart0_pins;
    status = "okay";
};
```

```c
serial@44e09000 {
    compatible = "ti,am3352-uart", "ti,omap3-uart";
    ti,hwmods = "uart1";
    clock-frequency = <0x2dc6c00>;
    reg = <0x44e09000 0x2000>;
    interrupts = <0x48>;
    status = "okay";
    dma = <0x26 0x1a 0x0 0x26 0x1b 0x0>;
    dma-names = "tx", "rx";
    pinctrl-names = "default";
    pinctrl-0 = <0x2a>;
};
```
HW DTS File – What binding was needed

• UART node in the DTS File

• UART node disassembled from the DTB file, significant difference. DTS only needs to define the pinmux and set status to “okay” in this example.

• Remember the debug tip from earlier about reverse compiling the DTB file.

```c
&uart0 {
    pinctrl-names = "default";
    pinctrl-0 = <&uart0_pins>;
    status = "okay";
};

serial@44e09000 {
    compatible = "ti,am3352-uart", "ti,omap3-uart";
    ti,hwmods = "uart1";
    clock-frequency = <0x2dc6c00>;
    reg = <0x44e09000 0x2000>;
    interrupts = <0x48>;
    status = "okay";
    dmas = <0x26 0x1a 0x0 0x26 0x1b 0x0>;
    dma-names = "tx", "rx";
    pinctrl-names = "default";
    pinctrl-0 = <0x2a>;
};
```
OMAP UART controller

Required properties:
- compatible : should be "ti,j721e-uart", "ti,am654-uart" for J721E controllers
- compatible : should be "ti,am654-uart" for AM654 controllers
- compatible : should be "ti,omap2-uart" for OMAP2 controllers
- compatible : should be "ti,omap3-uart" for OMAP3 controllers
- compatible : should be "ti,omap4-uart" for OMAP4 controllers
- compatible : should be "ti,am4372-uart" for AM437x controllers
- compatible : should be "ti,am3352-uart" for AM335x controllers
- compatible : should be "ti,dra742-uart" for DRA7x controllers
- reg : address and length of the register space
- interrupts or interrupts-extended:
  Should contain the uart interrupt specifier or both the interrupt controller phandle and interrupt specifier.
- ti,hwmods : Must be "uart<n>", n being the instance number (1-based)

Optional properties:
- clock-frequency : frequency of the clock input to the UART
- dmas : DMA specifier, consisting of a phandle to the DMA controller node and a DMA channel number.
- dma-names : "rx" for receive channel, "tx" for transmit channel.
- rs485-rts-delay, rs485-rx-during-tx, linux,
- rs485-enabled-at-boot-time: see rs485.txt
- rs485-rts-active-high: drive RTS high when sending (default is low).
- clocks: phandle to the functional clock as per Documentation/devicetree/bindings/clock/clock/bindings.txt

Example:
uart4: serial@49042000 {
  compatible = "ti,omap3-uart";
  reg = <0x49042000 0x400>;
  interrupts = <80>;
  dmas = <&sdma 81 &sdma 82>;
  dma-names = "tx", "rx";
  ti,hwmods = "uart4";
  clock-frequency = <48000000>;
};

Documentation/devicetree/bindings/serial/omap_serial.txt

46
OMAP UART controller

Required properties:
- compatible : should be "ti,j721e-uart", "ti,am654-uart" for J721E controllers
- compatible : should be "ti,am654-uart" for AM654 controllers
- compatible : should be "ti,omap2-uart" for OMAP2 controllers
- compatible : should be "ti,omap3-uart" for OMAP3 controllers
- compatible : should be "ti,omap4-uart" for OMAP4 controllers
- compatible : should be "ti,am4372-uart" for AM437x controllers
- compatible : should be "ti,am3352-uart" for AM335x controllers
- compatible : should be "ti,dra742-uart" for DRA7x controllers
- reg : address and length of the register space
- interrupts or interrupts-extended :
  Should contain the uart interrupt specifier or both the interrupt controller phandle and interrupt specifier.
- ti,hwmods : Must be "uart<n>", n being the instance number (1-based)

Optional properties:
- clock-frequency : frequency of the clock input to the UART
- dmas : DMA specifier, consisting of a phandle to the DMA controller node and a DMA channel number.
- dma-names : "rx" for receive channel, "tx" for transmit channel.
- rs485-rts-delay, rs485-rx-during-tx, linux,
- rs485-enabled-at-boot-time: see rs485.txt
- rs485-rts-active-high: drive RTS high when sending (default is low).
- clocks: phandle to the functional clock as per Documentation/devicetree/bindings/clock/clock-bindings.txt

Example:
uart4: serial@49042000 {
  compatible = "ti,omap3-uart";
  reg = <0x49042000 0x400>;
  interrupts = <80>;
  dmas = <&sdma 81 &sdma 82>;
  dma-names = "tx", "rx";
  ti,hwmods = "uart4";
  clock-frequency = <48000000>;
};
Did the hello world work?

Starting kernel ...

[ 0.000000] Booting Linux on physical CPU 0x0
[ 0.000000] Linux version 4.19.59-g5f8c1c6121 (oe-user@oe-host) (gcc version 8.3.0 (GNU Toolchain for the A-profile Architecture 8.3-2019.03 (arm-rel-8.36))) #1 PREEMPT Sat Oct 19 17:17:25 UTC 2019
[ 0.000000] CPU: ARMv7 Processor [413fc082] revision 2 (ARMv7), cr=10c5387d
[ 0.000000] OF: fdt: Machine model: TI AM335x Hello World
[ 0.000000] Built 1 zonelists, mobility grouping on. Total pages: 129920
[ 0.000000] Kernel command line: console=ttyO0,115200n8 root=PARTUUID=00000000-02 rw rootfstype=ext4 rootwait
Did the hello world work?

Starting kernel ...

[ 0.000000] Booting Linux on physical CPU 0x0
[ 0.000000] Linux version 4.19.59-g5f8c1c6121 (oe-user@oe-host) (gcc version 8.3.0 (GNU Toolchain for the A-profile Architecture 8.3-2019.03 (arm-rel-8.36))) #1 PREEMPT Sat Oct 19 17:17:25 UTC 2019
[ 0.000000] CPU: ARMv7 Processor [413fc082] revision 2 (ARMv7), cr=10c5387d

[ 0.000000] OF: fdt: Machine model: TI AM335x Hello World

[ 0.000000] Built 1 zonelists, mobility grouping on. Total pages: 129920
[ 0.000000] Kernel command line: console=tty00,115200n8 root=PARTUUID=00000000-02 rw rootfstype=ext4 rootwait
Did the hello world work?

Starting kernel ...

[ 0.000000] Booting Linux on physical CPU 0x0
[ 0.000000] Linux version 4.19.59-g5f8c1c6121 (oe-user@oe-host) (gcc version 8.3.0 (GNU Toolchain for the A-profile Architecture 8.3-2019.03 (arm-rel-8.36))) #1 PREEMPT Sat Oct 19 17:17:25 UTC 2019
[ 0.000000] CPU: ARMv7 Processor [413fc082] revision 2 (ARMv7), cr=10c5387d
[ 0.000000] OF: fdt: Machine model: TI AM335x Hello World
[ 0.000000] Built 1 zonelists, mobility grouping on. Total pages: 
[ 0.000000] Kernel command line: console=ttyO0,115200n8 root=PARTUUID=00000000-02 rootfstype=ext4 rootwait

/model = "TI AM335x Hello World ";
Did the hello world work?

```
[ 0.980750] sdhci: Secure Digital Host Controller Interface driver
[ 0.987072] sdhci: Copyright(c) Pierre Ossman
[ 0.992211] omap_gpio 44e07000.gpio: Could not set line 6 debounce to 200000 microseconds (-22)
[ 1.001011] omap_hsmmc 48060000.mmc: Got CD GPIO
[ 1.006261] omap_hsmmc 48060000.mmc: Linked as a consumer to regulator.1
[ 1.102726] mmc0: host does not support reading read-only switch, assuming write-enable
[ 1.122632] mmc0: new high speed SDHC card at address 0007
[ 1.136358] mmcblk0: mmc0:0007 SD08G 7.42 GiB
[ 1.143832] mmcblk0: p1 p2
[ 1.182949] EXT4-fs (mmcblk0p2): mounted filesystem with ordered data mode. Opts: (null)
[ 1.217988] Run /sbin/init as init process
```

```c
&mmc1 {
    status = "okay";
    bus-width = <0x4>;
    pinctrl-names = "default";
    pinctrl-0 = <&mmc1_pins>;
    cd-gpios = <&gpio0 6 GPIO_ACTIVE_LOW>;
    vmmc-supply = <&vmmcsd_fixed>;
};
```
Did the hello world work?

[ OK ] Started telnetd.service.
    Starting busybox-udhcpd.service...
    Starting thttpd.service...
[ OK ] Started Matrix GUI.
[ OK ] Started busybox-udhcpd.service.
[ OK ] Started thttpd.service.

Arago Project

Arago Project http://arago-project.org am335x-evm ttyS0
Arago 2019.07 am335x-evm ttyS0
am335x-evm login: root
root@am335x-evm:~#
LTS Kernel Lifecycle and the new Board
Linux Board Port – SDK Lifecycle

- **Initial board port**
- **LTS Release Date**
- **Time**
- **Production**
- **Issue detected during testing fixed in later Kernel**

- **Linux Kernel Current Annual LTS**
- **Linux Kernel LTS + 1**
- **Linux Kernel LTS + 2**
- **Linux Kernel LTS + ...**

- Upgrade/Port ?
- Backport ?

- [Image: Diagram showing the lifecycle of a Linux board port, including initial board port, LTS release dates, time, production, and issue detection in later kernels.]
How Processor dtsi files change over time

• In one year there significant changes to am335x-bone-common.dtsi (one of include files for am335x-boneblack.dts) and more importantly the AM335x processor am33xx.dtsi file

• When a board dts is not updated or maintained to kernel version it mostly likely will fail to at least compile.

• Recommend keeping the board hello_world.dts up to date.

• Never modify the processor dtsi file, makes upgrades that much more difficult
Keep at least the Hello World file current

- Reasons for recommending to at least keeping the board hello_world.dts up to date with the latest kernel.
  - Board Design Refresh
    - (components EOL)
  - Revision of board design
    - (debating upgrading kernels as well)
  - Looking to see if a later driver might fix an issue, don’t have to do a full upgrade.
Building the root filesystem into the Linux Kernel
Building the root filesystem into the kernel

- If there is an issue with the board for the interface that would be used for the root filesystem this technique can be used to bypass the interface, boot the kernel and perhaps debug the problematic interface.

- Need to find a source for the root filesystem, needs to be small.

- Make sure the linux utilities needed for debug are in the filesystem.
Building the root filesystem into the kernel
Building the root filesystem into the kernel

- This increases the size of the kernel so take that into account as the kernel is read into DDR by U-Boot.
Thank You!