Freescale Semiconductor
Application Note

Document Number: AN5125
Rev. 0, 09/2015

Introduction to Device Trees

Contents

1 Introduction

A device tree is a tree structure used to describe the physical hardware in a system. Each node in the tree describes the characteristics of the device being represented. The purpose of the device tree is to describe device information in a system that cannot necessarily be dynamically detected or discovered by a client program. For example, a PCI host may be able to probe and detect attached devices; and so a device tree node describing PCI devices may not be required. However, a device node is required to describe the PCI host bridge in the system, if that cannot be detected by probing.

Before the advent of the device tree, the kernel contained device specific code. A small change, such as the modification of an I2C peripheral's address would force a recompilation of the kernel image to be run.

The boot loader (for example, U-Boot) would load a single binary, the kernel image, and execute it.

Prior to device trees, several attempts were made to reduce complexity and pass small amounts of information to the kernel. The boot loader would typically prepare some additional information and place it in system ram at a location pointed to by a predefined register. This information would contain some basic information, such as memory size and location, and kernel command line information, such as IP address. The goal was to allow the kernel to configure hardware based on parsable information about the hardware rather than hard-coded initialization functions (for example, hard-coded IP addresses).

With device trees, the kernel itself no longer needs specific code for each version of hardware. Instead, the code is located in a separate binary: the device tree blob. This enables us to target different hardware with the same kernel image by simply changing the much simpler, and much smaller, device tree binary.

The device tree can be passed to the kernel either through appending it to the kernel image or through the bootloader. The machine type is now defined in the device tree itself. The bootloader can dynamically add some information (for example, clock frequencies) to the device tree and then passes a pointer to the tree, located in system memory, through r2 (for ARM®) architecture) or r3 (for Power Architecture®). The kernel then unflattens and parses the device tree.

2 Basic device tree

Device trees are well described in the Power.org Standard for Embedded Power Architecture Platform Requirements (ePAPR): https://www.power.org/documentation/epapr-version-1-1/. The ePAPR defines a concept, a device tree, to describe system hardware and separate that description from the kernel image.

The device tree is a tree structure with nodes that describe the physical devices in the system that cannot be dynamically detected by software. The nodes are organized in a hierarchical parent/child relationship.

This figure is a representation of a simple device tree, describing the platform type, CPU and memory. Nodes are organized in a hierarchy as a collection of property and value tokens. Sub-nodes define the relationship of devices within the hierarchy. (e.g. I2C devices are children of an I2C controller node.)

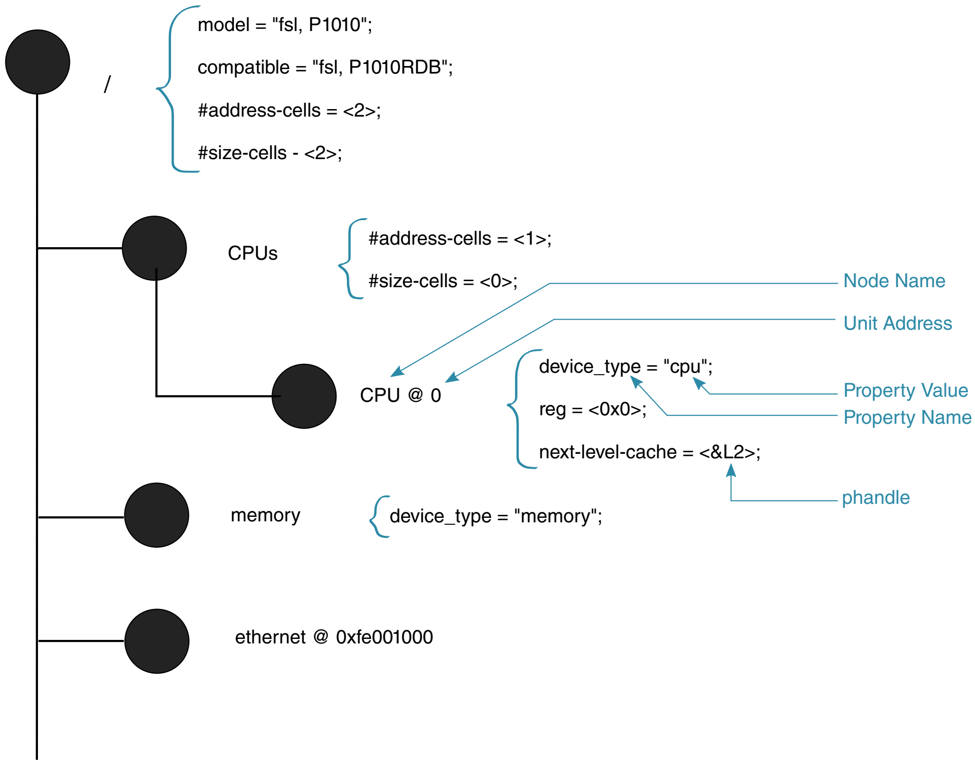

Figure 1. High-level device tree

In Figure 1, we see the definition of a P1010 based system. The compatible keyword specifies the name of the system in the form <**manufacturer**>, <**model**>. This may be used by the operating system to make decisions on how to run on the machine.

Further in the tree, we see a node named **cpus** define one CPU with a unit address of 0. This corresponds to the node's **reg** property and indicates that a single CPU is available.

Further in the tree, the node named Ethernet has a unit-address value of FE001000.

This example is intended as a simple example of portions of a device tree. The following sections delve into more advanced examples, as well as specifics of the syntax used to define nodes in the tree.

3 Syntax

A device tree is simply a tree structure of nodes and properties. Properties are key-value pairs and may contain both properties and child nodes. The following sections review the basic syntax of the device tree nodes, as well as parent/child node relationships.

3.1 Node names

The node name is a label used to identify the node. The unit-address component of the node identifies the base address of the bus on which the node sits. This is the primary address used to access the device.

Child nodes must be uniquely named, but can alternatively be addressed by a "unit name," which is used to differentiate nodes with the same name (for example, multiple I^2C devices in the same SoC) at the same level. Unit names are made of the node names, the "@" symbol, and a unit address (for example, i2c@3000, i2c@4000, and so on).

Multiple definitions of the same node are merged into one by the Device Tree Compiler.

3.2 Properties

A node may contain multiple properties arranged in *name* = *value* format. The *name* consists of a string, while *value* can be an array of strings, bytes, numbers, or phandles, or a mixture of types. For example, *value* can be:

- compatible = "fsl,mpc8610-msi", "fsl,mpic-msi";
- reg = <0 0 0x8000000>;
- interrupt-parent = <&mpic>;

NOTE

Numbers are always 32-bit big-endian in device trees. At times, multiple 32-bit big-endian numbers are used to represent a larger value (for example, 64-bit).

3.3 Phandle

A phandle (pointer handle) is a 32-bit value associated with a node that is used to uniquely identify that node so that the node can be reference from a property in another node. More simply put, it is a property in one node that contains a pointer to another node. A phandle is created either by the device tree compiler or U-Boot for each label.

In the following example, <&label> is converted to the phandle for the labeled node by the DTC.

```
name@address {
    <key> = <&label>;
};
```

```
label: name@adresss {
}
```

It is most commonly used for interrupts. In Listing 1 on page 7, **interrupt-parent** is assigned a phandle to the node with the label **mpic**.

3.4 Aliases

The aliases node is an index of other nodes. The properties of the node are paths within the device tree, not phandles.

```
aliases {
        ethernet0 = &enet0;
        ethernet1 = &enet1;
        ethernet2 = &enet2;
        serial0 = &serial0;
        serial1 = &serial1;
        pci0 = &pci0;
};
```

4 Memory mapping and addressing

Addresses are encoded using the following three properties:

- **reg**
- **#address-cells**
- **#size-cells**

Each addressable device has a **reg** property, which lists the address ranges used by the device through one or more 32-bit integers, called cells. Both address and length are variable in size, so the **#address-cells** and **#size-cells** properties in the parent node define the number of cells in each field.

CPU nodes represent a simple case of addressing. Each CPU is assigned a unique ID, and there is no size associated with CPU IDs.

```
cpus {
     #address-cells = <1>;
     #size-cells = <0>;

     cpu0: PowerPC,e6500@0 {
             device_type = "cpu";
             reg = <0 1>;
             next-level-cache = <&L2>;
     };

     cpu1: PowerPC,e65000@2 {
             device_type = "cpu";
             reg = <2 3>;
             next-level-cache = <&L2>;
     };
     cpu2: PowerPC,e6500@4 {
             device_type = "cpu";
             reg = <4 5>;
             next-level-cache = <&L2>;
     };
```

```
      cpu3: PowerPC,e65000@6 {
              device_type = "cpu";
              reg = <6 7>;
              next-level-cache = <&L2>;
      };
};
```

Memory mapped devices are assigned a range of addresses, rather than a single address value as found in CPU nodes. **#size-cells** of the parent indicates how large (in 32-bit quantities) the length field of each child is. **#address-cells** indicates how many 32-bit address cells are used per child, as well.

```
{
      #address-cells = <0x1>;
      #size-cells = <0x1>;
      compatible = "fsl,p1022-immr", "simple-bus";
      i2c@3100 {
           reg = <0x3100 0x100>;
      };
}
```

In the above example, we see two cells in the **reg** property of the I^{2}C child node. The first cell corresponds to the base address of 0x3100. The second cell is the size. So, the register map of this particular I^{2}C controller is from 0x3100 to 0x31ff.

Memory mapped devices may also include a **ranges** property in order to translate a range of addresses from parent to child devices.

The root node describes the CPU's address space. Child nodes of the root use the CPU's address domain and do not need explicit mapping. However, nodes that are not direct children of the root node do not use the CPU's address domain. The device tree must specify how to translate addresses from one domain to another. Through the ranges property, such translation is performed and a non-direct mapped child may obtain a memory mapped address.

```
/ {
    #address-cells = <0x2>;
    #size-cells = <0x2>;
    soc@fffe00000 {
        ranges = <0x0 0xf 0xffe00000 0x100000>;
        #address-cells = <0x1>;
        #size-cells = <0x1>;
        compatible = "fsl,p1022-immr", "simple-bus";
        i2c@3100 {
            #address-cells = <0x1>;
            #size-cells = <0x0>;
            cell-index = <0x1>;
            compatible = "fsl-i2c";
            reg = <0x3100 0x100>;
            codec@1a {
                compatible = "wlf,wm8776";
                reg = <0x1a>;
            };
        };
```

The **ranges** property defines a range of addresses for the child devices in this format: **<bus-address parent-bus-address size>**

- **bus-address** — bus base address, using **#address-size** of this bus node
- **parent-bus-address** — base address in the parent's address space, using **#address-size** of the parent node
- **size** — size of mapping, using **#address-size** of this node

Note that an empty **ranges** property indicates that the translation from parent to child address space is an identity mapping only, meaning that the parent bus address space is the same as the child bus address space. The absence of a **ranges** property is not the same as an empty ranges property. The absence of a **ranges** property means that translation is not possible (for example, with CPU nodes).

In the above example, the SoC has a range defined that maps to:

- Bus address = 0x0 (using the **#address-size** of the SoC node)
- Parent address = 0x0F_FFE0_0000

NOTE

Numbers are represented as 32-bit, big-endian in the device tree. However, because the **#address-size** of the parent node is set to 2, we concatenate two cells into a 64-bit address of 0x0000_000F_FFE0_0000.

In this example, the SoC node is defined at this address. This corresponds to the CCSR base address (or the internal register map base address) on the QorIQ P1022 device.

- Size = 0x100000 (using **#address-size** of the child node)

These essentially map address 0x0 of children to 0xF_FFE0_0000, which is the base address of the SoC. So, for example, the I^2C controller defined is at address 0x3100, which corresponds to an offset of 0x3100 from the base, or an absolute SoC address of 0xF_FFE0_3100.

Finally, there are devices that are not memory mapped on the processor bus. They may have indirect addresses that are not directly accessible by the CPU. Instead, the parent device's driver would be responsible for bus accesses.

```
i2c@3000 {
               gpio3: gpio@21 {
                   compatible = "nxp,pca9555";
                   reg = <0x21>;
                   #gpio-cells = <2>;
                   gpio-controller;
                   polarity = <0x00>;
               };
```

For example, the above I^2C controller taken from PSC9131rdb.dts shows an I^2C device assigned an address, 0x21, but no length or range associated with it.

PCI address space is completely separate from the CPU address space, and address translation is handled slightly differently. This is still performed using the **range**, **#address-cells**, and **#size-cells** properties.

```
pci1: pcie@ffe09000 {
        reg = <0 0xffe09000 0 0x1000>;
        ranges = <0x2000000 0x0 0xa0000000 0 0xa0000000 0x0 0x20000000
              0x1000000 0x0 0x00000000 0 0xffc10000 0x0 0x10000>;
```

PCI child addresses use three cells labeled phys.hi, phys.mid, and phys.low. The first of these, phys.hi, encodes information about the space. Most interesting may be the space coding, which indicates configuration space, I/O space, or 32-/64-bit memory space.

The PCI child address is followed by CPU address space and size. The size of these are determined by the parent's definition of **#address-cells** and **#size-cells**.

In the above example, we have two address spaces defined:

- A 32-bit memory region beginning at PCI address 0xa0000000, mapped to CPU address 0xa000000, with size = 0x20000000
- An I/O region beginning at PCI address 0x0, mapped to CPU address 0xffc10000, with size = 0x10000

4.1 Partitions

Many times, flash partitions are described in the device tree (see TABLE 1). This would, for example, correspond to a partition on an mtd device seen by the kernel. However, partitions typically are not based on a hardware description and are instead an arbitrary partitioning by the device tree author and should be discouraged.

5 Interrupts

Interrupts differ from addresses translations and do not follow the nature structure of the tree. Instead, interrupt signals can originate from and terminate anywhere in the machine. Interrupt signals are expressed as links between nodes, instead of naturally in tree form. Interrupt connections can be described using the following properties:

- **interrupt-controller**
- **#interrupt-cells**
- **interrupt-parent**
- **interrupts**

The **interrupt-controller** property is an empty property, declaring a node as a device that receives interrupt signals.

The **#interrupt-cells** property is a property of the interrupt controller node. It is used to define how many cells are in an interrupt specifier for the interrupt controller.

The **interrupt-parent** property is a property of a device node containing a phandle to the interrupt controller to which it is attached. Nodes without an interrupt-parent property can inherit the property from their parent node.

Finally, the **interrupts** property is a property of a device node containing a list of interrupt specifiers; one for each interrupt output signal.

The following two nodes show interrupts connections on a QorIQ P1010 device.

Listing 1. Example: Interrupt connections on a QorIQ P1010 device

```
mpic: pic@40000 {
                        interrupt-controller;
                        #address-cells = <0>;
                        #interrupt-cells = <4>;
                        reg = <0x40000 0x40000>;
                        compatible = "fsl,mpic";
                        device_type = "open-pic";
                };
serial0: serial@4500 {
                        cell-index = <0>;
                        device_type = "serial";
                        compatible = “fsl,ns16550”,"ns16550";
                        reg = <0x4500 0x100>;
                        clock-frequency = <0>;
                        interrupts = <42 2 0 0>;
                        interrupt-parent = <&mpic>;
                };
```

In Listing 1 on page 7, the interrupt controller is defined as **pic**, which is at address offset 0x40000. The label **mpic** was added to the interrupt controller node to assign a phandle to the **interrupt-parent** property in the root node.

For the MPIC, the **interrupt** property has either two or four values. The first cell always specifies the index of the xIVPR register of that interrupt. The first 16 are external interrupts; the remaining are internal interrupts. Therefore, internal interrupts have a value 16 larger than documented in the reference manuals. **#interrupt-cells** in the pic node, above, is defined as four, indicating each interrupt specifier has four cells. From the above example, the **interrupt** number was 42. 42 - 16 = 26, which, according to the P1010 reference manual, corresponds to the DUART interrupt.

The second value represents level sense. For MPIC, level sense is defined as follows:

- 0 = low-to-high edge sensitive type enabled
- 1 = active-low level sensitive type enabled
- 2 = active-high level sensitive type enabled
- 3 = high-to-low edge sensitive type enabled

If there is a third and fourth value, they represent **interrupt-type** and **type-info**. For MPIC, **interrupt-type** is defined as follows:

- 0 = normal
- 1 = error interrupt
- 2 = MPIC inter-processor interrupt
- 3 = MPIC timer interrupt

In the case of an error interrupt, **type-info** is the error interrupt number. **type-info** would also be valid for IPIs and timers.

The complete description of MPIC bindings can be found in Documentation/devicetree/bindings/powerpc/fsl/mpic.txt.

NOTE

In Listing 1 on page 7, **device_type** is deprecated and should not be used. Also, using **#cell-index** is discouraged. If used, the binding needs to be specific about what it corresponds to in the programming model, and alternatively, a more specific named property should be considered.

For the ARM GIC, the bindings are similar but different. The first cell defines the interrupt type:

- 0 = SPI interrupts
- 1 = PPI interrupts

The second cell contains the interrupt number. SPI interrupts number 0-987, while PPI interrupts number 0-15.

The third cell represents level sense:

- 1 = low-to-high edge sensitive
- 2 = high-to-low edge sensitive
- 4 = active-high level sensitive
- 8 = active-low level-sensitive

The complete description of GIC bindings can be found in Documentation/devicetree/bindings/arm/gic.txt.

For alternate interrupt controllers, we would have to examine the specific bindings for a complete explanation of the two cells, but these are typically defined with the first cell specifying interrupt number and the second specifying interrupt flags (such as edge/level triggering, active-high, active-low, and so on).

6 Example: Device tree node

Below is an example node of an I^2C controller, with two devices on the I^2C interface.

```
i2c@3000 {
    #address-cells = <1>;
    #size-cells = <0>;
    cell-index = <0>;
    compatible = "fsl-i2c";
    reg = <0x3000 0x100>;
```

```
    interrupts = <43 2>;
    interrupt-parent = <&mpic>;
    dfsrr;

    dtt@48 {
        compatible = "national,lm75";
        reg = <0x48>;
    };

    rtc@68 {
        compatible = "dallas,ds1337";
        reg = <0x68>;
    };
};
```

Using the syntax described above, we can make the following observations about this example node:

- The I^2C controller is located at offset 0x3000 from its parent.
- The driver for the I^2C controller is fsl-i2c.
- The first child is named dtt, at offset 0x48 from its parent; the driver is national lm75.
- The second child is named rtc, at offset 0x68 from its parent; the driver is Dallas ds1337.
- The interrupt parent is the mpic, and interrupt number 0x43 is used. Because this is OpenPIC, an offset of 16 is added to the interrupt number for internal interrupts. 43 - 16 = 27, so this is actually SoC interrupt 0x27.

7 Device tree inclusion

Device trees can be split into several files. As an example, the device tree for the QorIQ Qonverge product, the BSC9131 is split into two files.

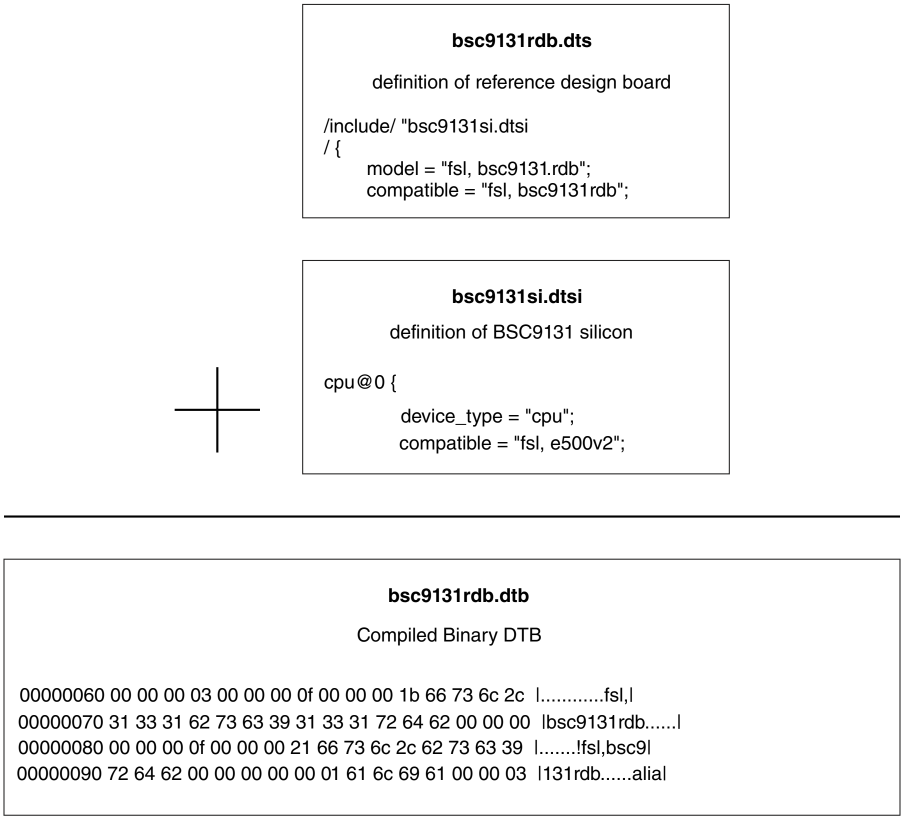

Figure 2. Device tree inclusion

.dts files are board level definitions. The .dts extension denotes “device tree source”.

.dtsi files are files included by the .dts files and generally contain SoC-level definitions. Device tree files do not have to be monolithic; instead, they can be split into several files, including each other. By convention, .dtsi files are included files, containing definitions of SoC-level information, while .dts files are final device trees containing board-level information. The .dtsi extension denotes “device tree source include”.

The inclusion works by overlaying the tree of the including file over the tree of the included file, producing a combined compiled binary.

As another example, the P1022 processor uses multiple include files for different SoC-specific nodes:

- p1022ds.dtsi — board definitions common to all addresses sizes
- p1022ds_32b.dts — main 32-bit DTS for the P1022 development system
- p1022ds_36b.dts — main 36-bit DTS for the P1022 development system
- fsl/p1022si-pre.dtsi — aliases and CPU nodes
- fsl/p1022si-post.dtsi — updates/overrides to SoC-specific nodes
- fsl/pq3-*.dtsi — common PowerQUICC III SoC devices
- fsl/qoriq-*.dtsi — common QorIQ SoC devices

```
/include/ "pq3-i2c-0.dtsi"
/include/ "pq3-i2c-1.dtsi"
/include/ "pq3-duart-0.dtsi"
/include/ "pq3-espi-0.dtsi"
spi@7000 {
        fsl,espi-num-chipselects = <4>;
};
```

8 Device tree compiler

The Device Tree Compiler (DTC) is the tool that is used to compile the source into a binary form. Source code for the DTC is located in scripts/dtc.

The output of the device tree compiler is a device tree blob (DTB), which is a binary form that gets loaded by the boot loader and parsed by the Linux kernel at boot.

On ARM® and ARM® 64-bit architectures, DTBs to be generated at build time are listed in arch/../boot/dts/Makefile, although they can be manually compiled by the DTC at any time.

The basic syntax of the DTC command line is: dtc [options] <input filename>

The most common options include:

```
-I <input format>
-O <output format>
-b <boot CPU>
      set the physical boot cpu
```

The input format could be .dts, .dtb, or .fs (.fs would read from the current file systems /proc/device-tree). The output format could be .dts, .dtb, or .asm. There are many other options, to pad bytes and so on (-R, -S, -P). As an example, to compile the above mentioned bsc9131rdb.dts file: dtc -I dts -O dtb bsc9131rdb.dts > bsc9131rdb.dtb

The DTC can also be used to reverse compile DTBs and make them human-readable again: dtc -I dtb -O dts bsc9131rdb.dtb > bsc9131rdb_output.dts

9 U-Boot

U-Boot updates the flattened device tree (FDT) with platform-specific information, such as the information derived from the reset configuration word (RCW), environment variables, and hardware configuration. The most common areas that U-Boot touches are related to frequency, MAC addresses, LIODNs (Peripheral MMU settings), and memory size — although the actual fix-ups are board specific and are not documented in any place other than the U-Boot code. Within U-Boot, the main function where this all occurs is ft_board_setup().

U-Boot itself does not use the device tree on current Freescale platforms, although it has several commands that allow you to view and manipulate the FDT itself:

- bootm has FDT-related subcommands:
  - bootm fdt — relocates the flattened device tree
  - bootm go — performs fix-up actions and boots the operating system
- fdt manipulates the FDT:
  - fdt addr <addr> [<length>] — sets the FDT location to <addr>
  - fdt boardsetup — performs board-specific setup
  - fdt move <fdt> <newaddr> <length> — copies the FDT to <addr> and makes it active
  - fdt resize — resizes the FDT to size + padding to 4 K address
  - fdt print <path> [<prop>] — recursive print starting at <path>
  - fdt set <path> <prop> [<val>] — sets <property> [to <val>]
  - fdt mknode <path> <node> — creates a new node after <path>
  - fdt rm <path> [<prop>] — deletes the node or <property>
  - fdt header — displays header information
  - fdt chosen [<start> <end>] — adds/updates the /chosen branch in the tree
    - <start>/<end> — initrd the start/end address

10 Linux

10.1 Reading the flattened device tree (FDT)

If CONFIG_PROC_DEVICETREE is set in the kernel configuration options, you can view the actual device tree parsed by the kernel from within the /proc file system after booting.

For example, you can execute a find for all nodes under /proc/device-tree:

```
[root@p4080ds]# cd /proc/device-tree
[root@p4080ds]# find
.
./name
[...]
./model
./fsl,dpaa/ethernet@0/fsl,qman-channel
[...]
./soc@ffe000000/fman@500000/ethernet@f0000/phy-connection-type
[...]

./soc@ffe000000/dma@100300/dma-channel@100/interrupts
[...]
./chosen/linux,initrd-start
```

You may also use the dtc tool to compile the /proc/device-tree into a DTS file:

```
[root@p4080DS]# dtc -I fs -O dts /proc/device-tree/
[...]
chosen {
bootargs = "root=/dev/ram rw console=ttyS0,115200 ramdisk_size=128000";
linux,stdout-path = "/soc@ffe000000/serial@11c500";
linux,initrd-start = <0x2f320000>;
linux,initrd-end = <0x2ffffd15>;
};
```

10.2 Device tree bindings

Device tree bindings describe the syntax used to describe specific types and classes of devices. The compatible property of a device node describes the specific binding, or bindings, to which the node complies. Device tree bindings recognized by the kernel are documented in Documentation/devicetree/bindings.

Each document describes which properties are accepted, with which values, as well as which properties are mandatory or optional. The latest device tree bindings can be found upstream.

As an example, below is the documentation for the IFC binding, located in Documentation/devicetree/bindings/powerpc/fsl/ifc.txt.

```
Integrated Flash Controller

Properties:
- name : Should be ifc
- compatible : should contain "fsl,ifc". The version of the integrated
               flash controller can be found in the IFC_REV register at
               offset zero.

- #address-cells : Should be either two or three.  The first cell is the
                   chipselect number, and the remaining cells are the
                   offset into the chipselect.
- #size-cells : Either one or two, depending on how large each chipselect
                can be.
- reg : Offset and length of the register set for the device
- interrupts : IFC has two interrupts. The first one is the "common"
               interrupt(CM_EVTER_STAT), and second is the NAND interrupt
               (NAND_EVTER_STAT).

- ranges : Each range corresponds to a single chipselect, and covers
           the entire access window as configured.

Child device nodes describe the devices connected to IFC such as NOR
(e.g.
cfi-flash) and NAND (fsl,ifc-nand). There might be board specific devices
like FPGAs, CPLDs, etc.

Example:

        ifc@ffe1e000 {
                compatible = "fsl,ifc", "simple-bus";
                #address-cells = <2>;
                #size-cells = <1>;
                reg = <0x0 0xffe1e000 0 0x2000>;
                interrupts = <16 2 19 2>;
```

Figure 3. IFC binding documentation

10.2.1 Manually parsing bindings

Occasionally, more often for modules, device tree bindings are undocumented. Because the kernel source is open, it is possible to go through the code and identify exactly how the node is used and by what driver code.

The compatible string is used to bind a device with a driver. Below is an example of an SPI node in the bsc9131rdb.dts file from the Freescale Wireless SDK Release 1.5:

```
spi@6000 {
        rfphy0: ad9361_phy@0{
            compatible = "fsl,espi-ad_phy","ad9361";
            reg = <0>;
            spi-max-frequency = <20000000>;
            spi-cpha;
            band_group1 = <1 7>;
            band_group2 = <41>;
```

```
            band_group3 = <>;
            band_group4 = <13 40>;
            band_group_sniff1 = <>;
            band_group_sniff2 = <13>;
            band_group_sniff3 = <1 7>;
            band_group_sniff4 = <>;
            band_group_config1 = <&pa_en 0 &lna_en 0>;
            band_group_config2 = <&pa_en 0 &lna_en 1>;
            band_group_config3 = <&pa_en 1 &lna_en 0>;
            band_group_config4 = <&pa_en 1 &lna_en 1>;


            reset: reset {
                label = "reset";
                gpios = <&gpio1 2 1>;
            };
            pa_en: pa_en {
                #num-val = <1>;
                label = "pa_en";
                gpios = <&gpio 18 1>;
            };
            lna_en: lna_en {
                #num-val = <1>;
                label = "lna_en";
                gpios = <&gpio 17 1>;
            };
        };
);
```

From the bindings in the node, we can see that the hardware is compatible with fsl, espi-ad_phy, and ad9361. This compatible property is used by the kernel to identify the hardware and match a driver that is registered in the kernel.

Looking through the source, we can see that espi-ad_phy is aliased to ad9361_phy (in file drivers/of/base.c). Further searching finds the driver for ad9361_phy is located in drivers/rf/phy/ad9361.c.

```
#define DRV_NAME "ad9361_phy"
static struct spi_driver ad_phy_driver = {
        .driver = {
                .name = DRV_NAME,
                .bus = &spi_bus_type,
                .owner = THIS_MODULE,
                },
        .probe = ad_phy_probe,
        .remove = __devexit_p(ad_phy_remove),
};
```

The driver name is registered with the kernel as ad9361_phy, which is why this particular driver is used.

Probe is defined as ad_phy_probe, which indicates the function used to parse the device tree. We can examine this function to see exactly where and how the properties in the device tree node for this RF module are used.

As another example, we can look at the T1040 device tree from the QorIQ SDK 1.6. The following is from t1040rdb.dts:

```
ucc@2200{
    compatible = "fsl,ucc_hdlc";
    rx-clock-name = "brg2";
    tx-clock-name = "brg2";
    fsl,rx-sync-clock = "none";
    fsl,tx-sync-clock = "none";
    fsl,tx-timeslot = <0xfffffffe>;
    fsl,rx-timeslot = <0xfffffffe>;
    fsl,tdm-framer-type = "e1";
    fsl,tdm-mode = "normal";
    fsl,tdm-id = <1>;
    fsl,siram-entry-id = <2>;
```

```
    fsl,inter-loopback;
};
```

In this example, the hardware is compatible with fsl,ucc_hdlc. We see that the driver for the hardware is located at drivers/net/wan/fsl_ucc.hdlc.c

```
#define DRV_DESC "Freescale QE UCC HDLC Driver"
#define DRV_NAME "ucc_hdlc"

static struct platform_driver_ucc_hdlc_driver = {
     .probe = ucc_hdlc_probe,
     .remove = ucc_hdlc_remove,
     .driver = {
          .owner = THIS_MODULE,
          .name = DRV_NAME,
          .pm = HDLC_PM_OPS,
          .of_match_table = fsl_ucc_hdlc_of_match,
     },
};
```

In this case, **probe** is defined as ucc_hdlc_probe, which indicates the function used to parse the device tree.

11 Examples

On Power Architecture®, for example, device trees are located in arch/powerpc/boot/dts. On ARM® architecture, device trees are for now located in arch/arm/boot/dts.

The following sections are commented examples of DTS and DTSI files for two different Freescale products — P2020 and LS1021A-TWR.

NOTE

For brevity, only certain sections are outlined below.

11.1 P2020 example

Below are example sections of a device tree for the P2020 RDB. This specific DTS file makes use of multiple DTSI include files.

11.1.1 P2020rdb.dts

This table shows the P2020rdb.dts file, which describes the P2020 board.

Table 1. P2020rdb.dts

DTS file	Comments
/include/ "fsl/p2020si-pre.dtsi"	Include file fsl/p2020si-pre.dtsi
/ {	Root node is identified with a forward slash
model = "fsl,P2020RDB";	Defines the manufacturer (fsl) and model number (P2020RDB) of the device

Table continues on the next page...

Table 1. P2020rdb.dts (continued)

DTS file	Comments
compatible = "fsl,P2020RDB";	Describes specific board
aliases {	Each property of the aliases node defines an index of other nodes
ethernet0 = &enet0;	
ethernet1 = &enet1;	
ethernet2 = &enet2;	
serial0 = &serial0;	
serial1 = &serial1;	
pci0 = &pci0;	
pci1 = &pci1;	
};	
memory {	Memory node
device_type = "memory";	Defines device type as memory
};	
lbc: localbus@ffe05000 {	Node localbus, starting at address 0xFFFE05000
reg = <0 0xffe05000 0 0x1000>;	First instance of **reg** must be equal to the address of the localbus node. Because **address-cells** = 2 (at the root node, defined in the DTSI file): • Address (64-bit quantity) = 0x0_FFE0_5000 • Size (at the root node) = 2 So, the size here is a 64-bit quantity (represented by two <u32> values) equal to 0x1000.
/* NOR and NAND Flashes */	The **ranges** property maps translation between the address space of the bus (child) and the address space of the parent. The first cell indicates chip select followed by offset into the chip-select address and size.
ranges = <0x0 0x0 0x0 0xef000000 0x01000000	CS0, offset 0xef000000, size 0x1000000
0x1 0x0 0x0 0xffa00000 0x00040000	CS1, offset 0xffa00000, size 0x40000
0x2 0x0 0x0 0xffb00000 0x00020000>;	CS2, offset 0xffb00000, size 0x20000
nor@0,0 {	This is the first child of the local bus. CS = 0, address offset is 0x0
#address-cells = <1>;	Addresses of children (for example, partitions) are 32 bits
#size-cells = <1>;	Size of children are 32 bits
compatible = "cfi-flash";	Hardware for NOR is indicated as cfi-flash
reg = <0x0 0x0 0x1000000>;	The local bus defines the first cell as chip select, followed by address and size. CS = 0, address offset 0x0000_0000, size = 0x1000000
bank-width = <2>;	16-bit device on the local bus
device-width = <1>;	A binding specific to cfi-flash
partition@0 {	First child of NOR
/* This location must not be altered */	
/* 256KB for Vitesse 7385 Switch firmware */	

Table continues on the next page...

Table 1. P2020rdb.dts (continued)

DTS file	Comments
reg = <0x0 0x00040000>;	reg = 0, which indicates this starts at the top of NOR (which was defined by parent = 0x0_EF00_0000) and goes for size = 0x40000
label = "NOR (RO) Vitesse-7385 Firmware";	Label and read-only are bindings specific to the driver. Label is a human readable string defining the device.
read-only;	
};	
partition@40000 {	Second child of NOR, starting at offset 0x40000
/* 256KB for DTB Image */	
reg = <0x00040000 0x00040000>;	Start address = 0x40000 and size = 0x40000
label = "NOR (RO) DTB Image";	
read-only;	
};	
……	
nand@1,0 {	Nand child of local bus, chip select = 1, address offset = 0x0 from parent
#address-cells = <1>;	
#size-cells = <1>;	
compatible = "fsl,p2020-fcm-nand",	The NAND node is defined as compatible with two different drivers.
"fsl,elbc-fcm-nand";	
reg = <0x1 0x0 0x40000>;	Local bus defines the first cell as chip select, followed by address and size. CS = 1, address offset 0x0000_0000, size = 0x4_0000
partition@0 {	
/* This location must not be altered */	
/* 1MB for u-boot Boot loader Image */	
reg = <0x0 0x00100000>;	First child of NAND; resides at the top of the NAND address range
label = "NAND (RO) U-Boot Image";	
read-only;	
};	
partition@100000 {	Second child of NAND, with address offset of 0x10_0000
/* 1MB for DTB Image */	
reg = <0x00100000 0x00100000>;	Address is offset 0x10_0000 from the top of NAND
label = "NAND (RO) DTB Image";	
read-only;	
};	
……	
soc: soc@ffe00000 {	Label is for SoC at address 0xFE00_0000.

Table continues on the next page...

Table 1. P2020rdb.dts (continued)

DTS file	Comments
	NOTE: This is >32bit, because the root node's **#address-cells** property is set to 2 (actual **#address-cells** mapping in the dtsi file). This node maps internal SoC registers to address 0x0FFE000000.
ranges = <0x0 0x0 0xffe00000 0x100000>;	Maps translation between the address space of the bus and the address space of the parent. Because address size of child nodes are 1, and address size of this node is 2. This maps address 0x0 of child nodes to 0x0_FFE0_0000, with a size of 0x100000.
……	
i2c@3000 {	I^2C node at address 0x3000
rtc@68 {	Child RTC at offset 0x68 from parent
compatible = "dallas,ds1339";	Hardware is Dallas, ds1339
reg = <0x68>;	Address 0x68
};	
};	
……	
spi@7000 {	SPI node at offset 0x7000
flash@0 {	SPI child at offset 0, labeled flash
#address-cells = <1>;	Child nodes use one <u32> address
#size-cells = <1>;	Child nodes use one <u32> for size
compatible = "spansion,s25sl12801";	Hardware is Spansion, s25sl12801
reg = <0>;	
spi-max-frequency = <40000000>;	Additional info parse-able by the SPI driver for max frequency allowable by SPI port
partition@0 {	Child of flash at offset 0
/* 512KB for u-boot Boot loader Image */	
reg = <0x0 0x00080000>;	Offset 0, size 0x80000
label = "SPI (RO) U-Boot Image";	Label used for the partition
read-only;	Attribute is parsable by flash driver
};	
partition@80000 {	Second partition of child under flash
/* 512KB for DTB Image */	
reg = <0x00080000 0x00080000>;	Offset 0x80000, size 0x80000
label = "SPI (RO) DTB Image";	Label used for the partition
read-only;	Attribute is parsable by flash driver
};	
partition@100000 {	Partition child of flash at offset 0x100000
/* 4MB for Linux Kernel Image */	
reg = <0x00100000 0x00400000>;	Offset 0x100000, size 0x400000
label = "SPI (RO) Linux Kernel Image";	Label used for the partition
read-only;	Attribute is parsable by flash driver

Table continues on the next page...

Table 1. P2020rdb.dts (continued)

DTS file	Comments
};	
partition@500000 {	Child partition of flash at offset 0x500000
/* 4MB for Compressed RFS Image */	
reg = <0x00500000 0x00400000>;	Offset 0x500000, size 0x400000
label = "SPI (RO) Compressed RFS Image";	Label used for the partition
read-only;	Attribute is parsable by flash driver
};	
partition@900000 {	Child partition of flash at offset 0x900000
/* 7MB for JFFS2 based RFS */	
reg = <0x00900000 0x00700000>;	Offset 0x900000, size 0x700000
label = "SPI (RW) JFFS2 RFS";	Label used for the partition
};	
};	
};	
usb@22000 {	USB at offset 0x22000 from the SoC
phy_type = "ulpi";	Attribute is parsable by the driver, indicating ULPI interface
dr_mode = "host";	Attribute is parsable by the driver, indicating USB host
};	
mdio@24520 {	MDIO port at offset 0x24520
phy0: ethernet-phy@0 {	PHY0 child at offset 0x0 from MDIO
interrupts = <3 1 0 0>;	xIVPR=3, active-low, normal
reg = <0x0>;	
};	
enet0: ethernet@24000 {	Ethernet 0 at offset 0x2400 from SoC
fixed-link = <1 1 1000 0 0>;	Attribute is parsable by driver, fixed link
phy-connection-type = "rgmii-id";	Attribute is parsable by driver, indicating PHY connection = RGMII
};	
enet1: ethernet@25000 {	Ethernet 1 at offset 0x2500 from the SoC
tbi-handle = <&tbi0>;	Pointer to TBI0
phy-handle = <&phy0>;	Pointer to Ethernet PHY0 (phandle)
phy-connection-type = "sgmii";	Attribute is parsable by driver, indicating PHY connection = SGMII
};	
pci0: pcie@ffe08000 {	
reg = <0 0xffe08000 0 0x1000>;	PCIe at offset 0xffe08000 from the root node
status = "disabled";	Currently set to disabled
};	
pci1: pcie@ffe09000 {	PCIe at offset 0xffe09000 from the root node
reg = <0 0xffe09000 0 0x1000>;	PCI1 registers at 0xffe090000.

Table continues on the next page...

Table 1. P2020rdb.dts (continued)

DTS file	Comments
	Because **address-cells** = 2 (at the root node, defined in the DTSI file): • Address (64-bit quantity) = 0x0_ffe09000 • Size = 0x1000
ranges = <0x2000000 0x0 0xa0000000 0 0xa0000000 0x0 0x20000000	PCI child addresses use three cells (phys.hi, phys.mid, and phys.low) phys.hi = 0x2000000, which is a field defined in bindings for things such as prefetchable, configuration space, memory space, and so on. In this case, it maps to a 32-bit memory space. PCI address = 0x0_a0000000 Translated to space 0x0_a0000000 from root node (using address-size from root space) Size of window = 0x20000000
0x1000000 0x0 0x00000000 0 0xffc10000 0x0 0x10000>;	phys.hi = 0x2000000, which is a field defined in bindings for things such as prefetchable, configuration space, memory space, and so on. In this case, it maps to the I/O space. PCI address = 0x0_00000000 Translated to space 0x0_ffc10000 from root node (using address-size from root space) Size of window = 0x10000
pcie@0 {	Child PCIe at offset 0x0 from parent (0xffe09000 from root)
ranges = <0x2000000 0x0 0xa0000000	PCIe (in p2020rdb-post.si) is defined with three <u32> address cells for the child, size = 2 <u32> phys.hi=0x2000000 = 32-bit memory space PCI address = 0x0_a0000000 Translated to space 0x2000000 (phys.hi), 0x0 (phys.mid), 0xa0000000 (phys.low) Size of window = 0x20000000
0x2000000 0x0 0xa0000000	
0x0 0x20000000	
0x1000000 0x0 0x0	phys.hi=0x1000000 = I/O space PCI address = 0x0_00000000 Translated to space 0x1000000 (phys.hi), 0x0 (phys.mid), 0x0 (phys.low) Size of window = 0x100000
0x1000000 0x0 0x0	
0x0 0x100000>;	
};	
};	
/include/ "fsl/p2020si-post.dtsi"	Include file fsl/p2020si-post.dtsi

11.1.2 P2020si-pre.dtsi

This table provides a device tree included in the p2020si-pre.dtsi file, which also includes the e500vs_power_isa.dtsi file.

Table 2. P2020si-pre.dtsi

DTS file	Comments
/dts-v1/;	Indication that this DTS file conforms with DTS version 1
/include/ "e500v2_power_isa.dtsi"	Include file e500v2_power_isa.dtsi
/ {	Root node is identified with a forward slash
compatible = "fsl,P2020";	Can be used by a program for device driver selection (for example, by an operating system to select platform-specific code)
#address-cells = <2>;	Defines the number of <u32> cells used to encode address by children as 2
#size-cells = <2>;	Root node defines size as two <u32>
interrupt-parent = <&mpic>;	Interrupts are directed to MPIC
aliases {	Each property of the aliases node defines an index of other nodes
serial0 = &serial0;	
serial1 = &serial1;	
ethernet0 = &enet0;	
ethernet1 = &enet1;	
ethernet2 = &enet2;	
pci0 = &pci0;	
pci1 = &pci1;	
pci2 = &pci2;	
};	
cpus {	CPU node
#address-cells = <1>;	Defines the number of <u32> cells used to encode the address field in child nodes **reg** property
#size-cells = <0>;	Defines the number of <u32> cells used to encode the size field in a child node's **reg** property. Because this is 0, children are not expected to have a size field in the **reg** property.
PowerPC,P2020@0 {	Node is a labeled PowerPC, P2020
device_type = "cpu";	Indicates this is a CPU node
reg = <0x0>;	Indicates CPU 0
next-level-cache = <&L2>;	Pointer to the next level of cache
};	
PowerPC,P2020@1 {	Node is a labeled PowerPC, P2020
device_type = "cpu";	Indicates this is a CPU node
reg = <0x1>;	Indicates CPU 1
next-level-cache = <&L2>;	Pointer to the next level of cache
};	

Table continues on the next page...

Table 2. P2020si-pre.dtsi (continued)

DTS file	Comments
};	
};	

11.1.3 P2020si-post.dtsi

The post DTSI file contains definitions for the peripherals on the SoC, such as local bus and PCI. Many of these are referenced to as phandles in the main p2020rdb.dts file. This file, in turn, includes many other DTSI files defining the specific peripherals. Many uses would not need to touch these files because they are SoC specific.

As an example, here is the local bus controller definition from the post.dtsi file:

```
&lbc {
    #address-cells = <2>;
    #size-cells = <1>;
    compatible = "fsl,p2020-elbc", "fsl,elbc", "simple-bus";
    interrupts = <19 2 0 0>;
};
```

This is a typical node that defines children to have two address cells and one size cell. The LBC hardware is compatible with "fsl, p2020-elbc", "fsl, elbc", and "simple-bus". **Interrupt** is defined at #19 and set to active-high level sensitive. **&lbc** is a label to the node path.

11.2 LS1021A example

Below is an example of a device tree for the LS1021A-TWR board. This DTS file describes the LS1021A-TWR board, and includes other DTSI files, as shown in the following figure.

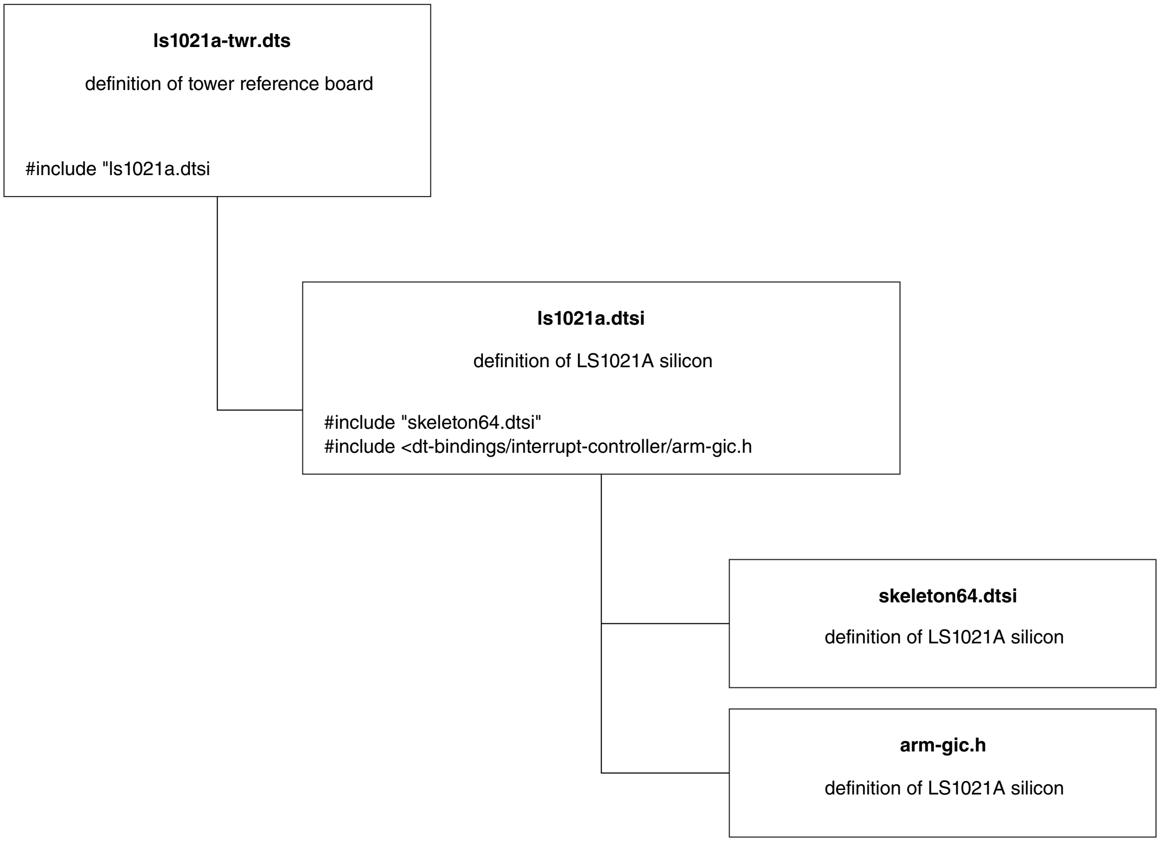

Figure 4. LS1021A DTS and DTSI structure

11.2.1 ls1021a-twr.dts

This table shows the ls1021a-twr.dts file, which describes the LS1021A-TWR board.

Table 3. ls1021a-twr.dts

DTS file	Comments
/dts-v1/;	Indicates that this DTS file conforms with DTS version 1
#include "ls1021a.dtsi"	Include file ls1021a.dts
/ {	Root node is identified with a forward slash
model = "LS1021A TWR Board";	Defines the model number of the device
aliases {	Each property of the aliases node defines an index of other nodes
enet2_rgmii_phy = &rgmii_phy1;	
enet0_sgmii_phy = &sgmii_phy2;	
enet1_sgmii_phy = &sgmii_phy0;	
};	
clocks {	Clocks node;
sys_mclk: clock {	Definition of phandle sys_mclk

Table continues on the next page...

Table 3. ls1021a-twr.dts (continued)

DTS file	Comments
compatible = "fixed-clock";	Defined as a fixed-frequency clock, using fixed-clock bindings
#clock-cells = <0>;	Single clock output
clock-frequency = <24576000>;	Frequency of the clock
};	
};	
regulators {	Regulators node;
compatible = "simple-bus";	Memory mapped with no specific driver; child nodes are registered as platform devices.
#address-cells = <1>;	Defines the number of <u32> cells used to encode address by children as 1
#size-cells = <0>;	Child cells use no size encoding
reg_3p3v: regulator@0 {	Defines phandle reg_3pvp to regulator at address 0
compatible = "regulator-fixed";	Hardware is compatible with "regulator-fixed"; can be used by the operating system or the device driver.
reg = <0>;	Regulator is assigned a single address of 0, which matches address at initialization of node (reg_3p3v: regulator@0)
regulator-name = "3P3V";	Name for the regulator output
regulator-min-microvolt = <3300000>;	Smallest voltage allowed
regulator-max-microvolt = <3300000>;	Largest voltage allowed
regulator-always-on;	Regulator should never be disabled
};	
};	
sound {	Sound node
compatible = "fsl,vf610-sgtl5000";	Specific driver used for sound
simple-audio-card,name = "FSL-VF610-TWR-BOARD";	
simple-audio-card,routing =	No connections between audio components
"MIC_IN", "Microphone Jack",	
"Microphone Jack", "Mic Bias",	
"LINE_IN", "Line In Jack",	
"Headphone Jack", "HP_OUT",	
"Speaker Ext", "LINE_OUT";	
simple-audio-card,cpu = <&sai1>;	Points to phandle &sai1, defined in .dtsi
simple-audio-card,codec = <&codec>;	Points to phandle &codec, defined later as sgtl5000
};	
};	
&dcu0 {	Points to phandle dcu0;
display = <&display>;	Points to phandle display
status = "okay";	Device is enabled
display: display@0 {	Phandle display defined at address 0
bits-per-pixel = <24>;	Should be 24 for RGB888

Table continues on the next page...

Table 3. ls1021a-twr.dts (continued)

DTS file	Comments
display-timings {	See the binding document display-timing.txt for bindings
native-mode = <&timing0>;	
timing0: nl4827hc19 {	
clock-frequency = <10870000>;	
hactive = <480>;	
vactive = <272>;	
hback-porch = <2>;	
hfront-porch = <2>;	
vback-porch = <2>;	
vfront-porch = <2>;	
hsync-len = <41>;	
vsync-len = <4>;	
hsync-active = <1>;	
vsync-active = <1>;	
};	
};	
};	
};	
&duart0 {	Merges with the expanded path of node with duart0:label from the DTSI file.
status = "okay";	Device is enabled
};	
&duart1 {	Merges with the expanded path of node with duart1:label from the DTSI file.
astatus = "okay";	Device is enabled
};	
&enet0 {	Merges with the expanded path of node with enet0:label from the DTSI file.
tbi-handle = <&tbi1>;	Phandle of TBI interface for this MAC
phy-handle = <&sgmii_phy2>;	Phandle of PHY connected to this controller
phy-connection-type = "sgmii";	Controller/PHY interface is SGMII
status = "okay";	Device is enabled
};	
&enet1 {	Merges with the expanded path of node with enet1:label from the DTSI file.
tbi-handle = <&tbi1>;	Phandle of TBI interface for this MAC
phy-handle = <&sgmii_phy0>;	Phandle of PHY connected to this controller
phy-connection-type = "sgmii";	Controller/PHY interface is SGMII
status = "okay";	Device is enabled
};	

Table continues on the next page...

Table 3. ls1021a-twr.dts (continued)

DTS file	Comments
&enet2 {	Merges with the expanded path of node with enet2:label from the DTSI file.
phy-handle = <&rgmii_phy1>;	Phandle of PHY connected to this controller
phy-connection-type = "rgmii-id";	Controller/PHY interface is RGMII
status = "okay";	Device is enabled
};	
&i2c0 {	Merges with the expanded path of node with i2c0:label from the DTSI file.
status = "okay";	Device is enabled
};	
&i2c1 {	Merges with the expanded path of node with i2c1:label from the DTSI file.
status = "okay";	Device is enabled
codec: sgtl5000@a {	
compatible = "fsl,sgtl5000";	Use audio driver sgtl5000
reg = <0x0a>;	
VDDA-supply = <&reg_3p3v>;	
VDDIO-supply = <&reg_3p3v>;	
clocks = <&sys_mclk 1>;	
};	
hdmi: sii9022a@39 {	Define phandle HDMI, pointing to node sii9022a at address 0x39
compatible = "fsl,sii902x";	Use sii902x driver
reg = <0x39>;	I^{2}C address of the device
interrupts = <GIC_SPI 167 IRQ_TYPE_EDGE_RISING>;	Interrupts to the CPU
};	
};	
&i2c2 {	Label to node path for I^{2}C2
status = "okay";	Device is enabled
monitor: ltc2945@67 {	
reg = <0x67>;	
};	
};	
&ifc {	Merges with the expanded path of node with ifc:label from the DTSI file.
status = "okay";	Device is enabled
#address-cells = <2>;	Defines the number of <u32> cells used to encode address by children as 2
#size-cells = <1>;	Defines the number of <u32> cells used to encode size by children as 1
/* NOR, and CPLD on board */	The **ranges** property maps translation between the address space of the bus (child) and the address space of the parent.

Table continues on the next page...

Table 3. ls1021a-twr.dts (continued)

DTS file	Comments
	Because **address-cells** = 2, all addressing is via two <u32> values.
ranges = <0x0 0x0 0x0 0x60000000 0x08000000	Maps 0x0 of child to 0x60000000 of parent, for a size of 0x08000000
0x2 0x0 0x0 0x7fb00000 0x00000100>;	Maps 0x2_0000_0000 of child to 0x7fb00000 of parent, for a size of 0x00000100
nor@0,0 {	First child of IFC, address of 0 (which is translated to address 0x60000000 of parent
compatible = "cfi-flash";	Driver to use for NOR flash
#address-cells = <1>;	Defines the number of <u32> cells used to encode address by children as 1
#size-cells = <1>;	Defines the number of <u32> cells used to encode size by children as 1
reg = <0x0 0x0 0x8000000>;	Memory space at address 0x0 for a size of 0x8000000
bank-width = <2>;	Width, in bytes, of flash interface
device-width = <1>;	Width, in bytes, of single flash device
partition@0 {	First mtd partition of NOR flash
/* 128KB for rcw */	
reg = <0x00000000 0x0020000>;	Memory space at address 0x0 for a size of 0x0020000
label = "NOR bank0 RCW Image";	Label for mtd driver
};	
partition@20000 {	Second partition, at address 20000
/* 1MB for DTB */	
reg = <0x00020000 0x00100000>;	Memory space at address 0x00020000 (note that address matches address in node definition) for a size of 0x00100000
label = "NOR DTB Image";	Label for mtd driver
};	
partition@120000 {	mtd partition of NOR flash
/* 8 MB for Linux Kernel Image */	
reg = <0x00120000 0x00800000>;	Memory space at address 0x00120000 for a size of 0x00800000
label = "NOR Linux Kernel Image";	Label for mtd driver
};	
partition@920000 {	mtd partition of NOR flash
/* 56MB for Ramdisk Root File System */	
reg = <0x00920000 0x03600000>;	Memory space at address 0x00920000 for a size of 0x03600000
label = "NOR Ramdisk Root File System Image";	Label for mtd driver
};	
partition@3f80000 {	mtd partition of NOR flash
/* 512KB for bank4 u-boot Image */	
reg = <0x03f80000 0x80000>;	Memory space at address 0x03f80000 for a size of 0x80000

Table continues on the next page...

Table 3. ls1021a-twr.dts (continued)

DTS file	Comments
label = "NOR bank4 u-boot Image";	Label for mtd driver
};	
partition@4000000 {	mtd partition of NOR flash
/* 128KB for bank4 RCW Image */	
reg = <0x04000000 0x20000>;	Memory space at address 0x04000000 for a size of 0x20000
label = "NOR bank4 RCW Image";	Label for mtd driver
};	
partition@4020000 {	mtd partition of NOR flash
/* 63MB JFFS2 ROOT File System Image */	
reg = <0x04020000 0x3f00000>;	Memory space at address 0x04020000 for a size of 0x3f00000
label = "NOR JFFS2 ROOT File System Image";	Label for mtd driver
};	
partition@7f80000 {	mtd partition of NOR flash
/* 512KB for bank0 u-boot Image */	
reg = <0x07f80000 0x80000>;	Memory space at address 0x07f80000 for a size of 0x80000
label = "NOR bank0 u-boot Image";	Label for mtd driver
};	
};	
};	
&lpuart0 {	Merges with the expanded path of node with lpuart0:label from the DTSI file.
status = "okay";	Device is enabled
};	
&mdio0 {	Merges with the expanded path of node with mdio0:label from the DTSI file.
sgmii_phy0: ethernet-phy@0 {	Defines the PHY and address
reg = <0x0>;	Offset of the register set for this device
};	
rgmii_phy1: ethernet-phy@1 {	Defines the PHY and address
reg = <0x1>;	Offset of the register set for this device
};	
sgmii_phy2: ethernet-phy@2 {	Defines the PHY and address
reg = <0x2>;	Offset of the register set for this device
};	
tbi1: tbi-phy@1f {	Defines TBI PHY at address 0x1f
reg = <0x1f>;	Offset of the register set for this device
device_type = "tbi-phy";	
};	
};	

Table continues on the next page...

Table 3. ls1021a-twr.dts (continued)

DTS file	Comments
&uqe {	Merges with the expanded path of node with uqe:label from the DTSI file.
tdma: ucc@2000 {	
compatible = "fsl,ucc-tdm";	
rx-clock-name = "clk8";	
tx-clock-name = "clk9";	
fsl,rx-sync-clock = "rsync_pin";	
fsl,tx-sync-clock = "tsync_pin";	
fsl,tx-timeslot = <0xfffffffe>;	
fsl,rx-timeslot = <0xfffffffe>;	
fsl,tdm-framer-type = "e1";	
fsl,tdm-mode = "normal";	
fsl,tdm-id = <0>;	
fsl,siram-entry-id = <0>;	
};	
serial: ucc@2200 {	Node UCC at address 0x2200
device_type = "serial";	Defines the device type for UCC
compatible = "ucc_uart";	Driver for UCC Uart
port-number = <1>;	Corresponds to /dev/ttyQE device
rx-clock-name = "brg2";	UCC Rx clock source
tx-clock-name = "brg2";	UCC Tx clock source
};	
};	
&pwm6 {	Merges with the expanded path of node with pwm6:label from the DTSI file.
status = "okay";	Device is enabled
};	
&pwm7 {	Merges with the expanded path of node with pwm7:label from the DTSI file.
status = "okay";	Device is enabled
};	
&qspi {	Merges with the expanded path of node with qspi:label from the DTSI file.
num-cs = <2>;	Number of chip selects for QSPI
status = "okay";	Device is enabled
qflash0: n25q128a13@0 {	Define the phandle of QFLASH0 pointing to node n25q128a13 at address 0
compatible = "micron,n25q128a13";	Driver used is micron n25q128a13
#address-cells = <1>;	Child nodes use one address cell
#size-cells = <1>;	Child nodes use one size cell
spi-max-frequency = <20000000>;	Max frequency

Table continues on the next page...

Table 3. ls1021a-twr.dts (continued)

DTS file	Comments
reg = <0>;	
};	
};	
&sai1 {	Merges with the expanded path of node with sai1:label from the DTSI file.
status = "okay";	Device is enabled
};	

11.2.2 ls1021a.dtsi

The following sections use information from the ls1021a.dtsi file, which describes the LS1021A SoC hardware and is included by the tower board device tree. Two additional files are included from within this one, neither of which are commented upon from within this document.

Table 4. ls1021a.dtsi

DTS file	Comments
#include "skeleton64.dtsi"	
#include <dt-bindings/interrupt-controller/arm-gic.h>	Include file arm-gic.h, for interrupt controller definition
/ {	Root node is identified with a forward slash
compatible = "fsl,ls1021a";	Can be used by a program for device driver selection (for example, by an operating system to select platform specific code)
interrupt-parent = <&gic>;	Interrupt controller points to phandle GIC (defined in arm-gic.h)
#address-cells = <2>;	Defines the number of <u32> cells used to encode address by children as 2
#size-cells = <2>;	Defines the number of <u32> cells used to encode size by children as 2
aliases {	Each property of the aliases node defines an index of other nodes
serial0 = &lpuart0;	
serial1 = &lpuart1;	
serial2 = &lpuart2;	
serial3 = &lpuart3;	
serial4 = &lpuart4;	
serial5 = &lpuart5;	
ethernet0 = &enet0;	
ethernet1 = &enet1;	
ethernet2 = &enet2;	
sysclk = &sysclk;	

Table continues on the next page...

Table 4. ls1021a.dtsi (continued)

DTS file	Comments
gpio0 = &gpio1;	
gpio1 = &gpio2;	
gpio2 = &gpio3;	
gpio3 = &gpio4;	
crypto = &crypto;	
};	
memory@80000000 {	Memory is located at address 0x80000000
device_type = "memory";	
reg = <0x0 0x80000000 0x0 0x20000000>;	Size (defined in parent was 2, and address was 2), address = 0x80000000 and size = 0x20000000
};	
cpus {	CPU node
#address-cells = <1>;	Defines the number of <u32> cells used to encode the address field in child nodes **reg** property
#size-cells = <0>;	Defines the number of <u32> cells used to encode the size field in a child node's **reg** property. Because this is 0, children are not expected to have a size field in the **reg** property.
cpu@f00 {	Defines the node as ARM, Cortex®-A7, address f00 (for example, CPU f00)
compatible = "arm,cortex-a7";	Can be used by a program for device driver selection (for example, by an operating system to select platform-specific code)
device_type = "cpu";	Indicates this is a CPU node
reg = <0xf00>;	**reg** defines the CPU ID and must match the address of the CPU node.
clocks = <&cluster1_clk>;	
};	
cpu@f01 {	Defines the node as ARM, Cortex-A7, address f01 (for example, CPU f01)
compatible = "arm,cortex-a7";	
device_type = "cpu";	Indicates this is a CPU node
reg = <0xf01>;	**reg** defines the CPU ID and must match the address of the CPU node.
clocks = <&cluster1_clk>;	
};	
};	
soc {	High-level node that defines the SoC
compatible = "simple-bus";	Memory mapped with no specific driver. Child nodes are registered as platform devices.
#address-cells = <2>;	Defines the number of <u32> cells used to encode address by children as 2
#size-cells = <2>;	Defines the number of <u32> cells used to encode size by children as 2

Table continues on the next page...

Table 4. ls1021a.dtsi (continued)

DTS file	Comments
device_type = "soc";	Sets the device type
interrupt-parent = <&gic>;	Sets interrupts to go to phandle GIC
ranges;	Empty ranges indicates identity mapping is used
gic: interrupt-controller@1400000 {	Phandle GIC, pointing to interrupt-controller at address 0x1400000
compatible = "arm,cortex-a15-gic";	Can be used by a program for device driver selection (for example, by an OS to select platform-specific code)
#interrupt-cells = <3>;	Number of cells needed to encode an interrupt
interrupt-controller;	Defines the node as interrupt controller
reg = <0x0 0x1401000 0x0 0x1000>,	Registers at location 0x1401000, size 0x1000
<0x0 0x1402000 0x0 0x1000>,	Registers at location 0x1402000, size 0x1000
<0x0 0x1404000 0x0 0x2000>,	Registers at location 0x1404000, size 0x2000
<0x0 0x1406000 0x0 0x2000>;	Registers at location 0x1406000, size 0x2000
interrupts = <GIC_PPI 9 (GIC_CPU_MASK_SIMPLE(2) | IRQ_TYPE_LEVEL_HIGH)>;	Definition of interrupt from GIC
};	
ifc: ifc@1530000 {	IFC node at address 0x1530000
compatible = "fsl,ifc", "simple-bus";	Compatible with the "fsl,ifc" driver
reg = <0x0 0x1530000 0x0 0x10000>;	Registers at location 0x1530000, size 0x10000
interrupts = <GIC_SPI 75 IRQ_TYPE_LEVEL_HIGH>;	Definition of interrupt from IFC
};	
qspi: quadspi@1550000 {	QSPI node at address 0x1550000
compatible = "fsl,ls1-qspi";	Can be used by a program for device driver selection (for example, by an OS to select platform-specific code)
#address-cells = <1>;	Defines the number of <u32> cells used to encode address by children as 1
#size-cells = <0>;	Defines the number of <u32> cells used to encode size by children as 1
reg = <0x0 0x1550000 0x0 0x10000>,	Memory mapped registers defined at 0x1550000, size 0x10000
<0x0 0x40000000 0x0 0x4000000>;	Memory mapped registers defined at 0x40000000 size 0x40000000
reg-names = "QuadSPI", "QuadSPI-memory";	Indicates first block of registers is named QuadSPI and second block is QuadSPI-memory
interrupts = <GIC_SPI 131 IRQ_TYPE_LEVEL_HIGH>;	Definition of QSPI interrupt
clock-names = "qspi_en", "qspi";	Clock names for QSPI
clocks = <&platform_clk 1>, <&platform_clk 1>;	Corresponding clocks for QSPI
big-endian;	Indicates that the peripheral is big-endian
amba-base = <0x40000000>;	Indicates that the AMBA® base address is 0x40000000
status = "disabled";	Indicates that the QSPI block is disabled
};	
i2c0: i2c@2180000 {	I^{2}C node at address 0x2180000

Table continues on the next page...

Table 4. ls1021a.dtsi (continued)

DTS file	Comments
compatible = "fsl,vf610-i2c";	Hardware is described by vf610-i2c
#address-cells = <1>;	Defines the number of <u32> cells used to encode address by children as 1
#size-cells = <0>;	Children have no size encodings
reg = <0x0 0x2180000 0x0 0x10000>;	Registers at 0x2180000, size 0x10000
interrupts = <GIC_SPI 88 IRQ_TYPE_LEVEL_HIGH>;	Definition of interrupt for I^2C0
clock-names = "i2c";	Label for the clock
clocks = <&platform_clk 1>;	Clocks = phandle platform_clk
status = "disabled";	Indicates that the device is not presently operational, but may become operational at a later time
};	

12 Revision history

This table provides a revision history for this application note.

Table 5. Document revision history

Rev. number	Date	Description
0	09/2015	Initial public release

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number AN5125
Revision 0, 09/2015