National Yang Ming Chiao Tung University
Computer Architecture & System Lab

DeviceTree

IOC5226 Operating System Capstone

Tsung Tai Yeh
Department of Computer Science
National Yang Ming Chiao Tung University

Acknowledgements and Disclaimer

- Slides were developed in the reference with
  - MIT 6.828 Operating system engineering class, 2018
  - MIT 6.004 Operating system, 2018
  - Remzi H. Arpaci-Dusseau etl. , Operating systems: Three easy pieces. WISC

Outline

- Devicetree Structure
- Standard Properties
- Flattened DeviceTree (DTB)

DeviceTree

- What is DeviceTree?
  - Describe system hardware
  - Provides a complete boot program to client programs (OS, bootloader, or hypervisor) interface definition
  - Describes devices which cannot be dynamically detected by a client program, such as PCI host bridge, instead of PCI
- How to use DeviceTree?
  - A boot program loads a devicetree into a client program’s memory
  - Passes a pointer to the devicetree to the client

https://reurl.cc/zQ8Avp

DeviceTree

- When will we use DeviceTree?
  - **Step 7: Launch the kernel (S-mode RISC-V)**
    - U-Boot calls bootm/booti to jump to the kernel entry
    - Passing FDT (Flattened Device Tree), memory map, and bootargs
  - **Step 8: Kernel early boot (mounts rootfs)**
    - Kernel sets up MMU/VM, driver, mount root, pivots to userspace PID 1
  - **Step 9: Userspace & services**
    - Services start (networking, containers, app ..)

https://reurl.cc/A9q6xe

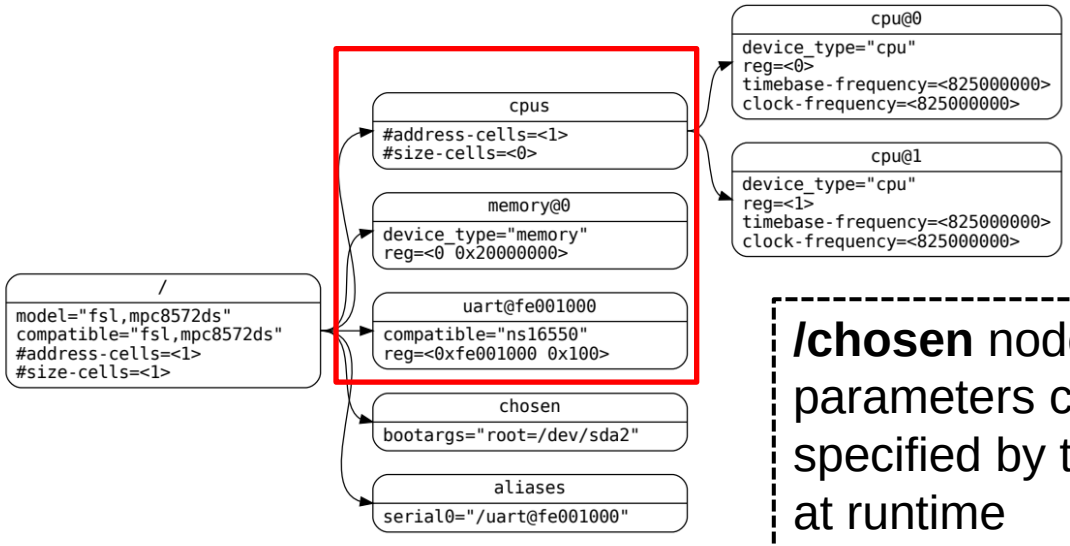
DeviceTree

- DeviceTree is
  - A tree data structure with nodes that describe the devices in a system
  - Each node has property/value pairs
    - Describe the characteristics of the device being represented
    - Has exactly one parent except for the root node
    - A device is an actual hardware device, such as a UART

https://reurl.cc/zQ8Avp

DeviceTree

- DeviceTree Example
  - A platform having CPU, memory, and a single UART
  - Device nodes are shown with properties and values inside each node



/chosen node describes parameters chosen or specified by the firmware at runtime

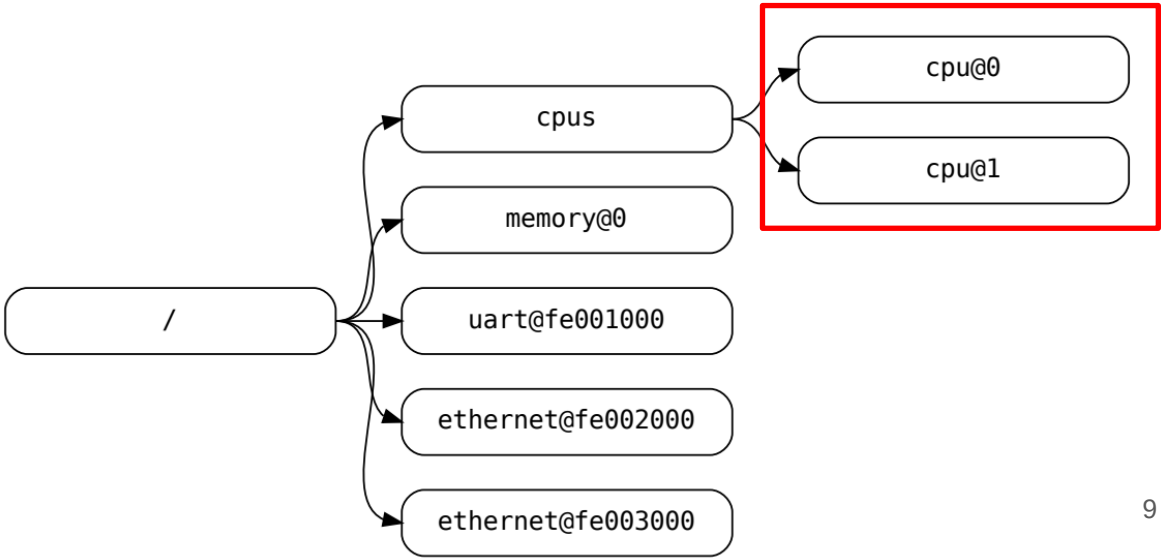
DeviceTree Structure

```
[label:] node-name[@unit-address] {
    [properties definitions]
    [child nodes]
};
```

- Node Names
  - Each node in the devicetree is named according to
    - node-name@unit-address
    - The node-name component specifies the name of the node, 1 to 31 characters in length
    - The unit-address component of the name is specific to the bus type on which the node sits
      - The unit-address must match the first address specified in the reg property of the node

DeviceTree Structure

- Node Names Example
  - In the case of node-name with an @unit-address shall be unique from any property names at the same level in the tree
  - The root node is identified by ‘/’



DeviceTree Structure

- Properties
  - Describe the characteristics of the node (a name/ a value)
  - Nonstandard Property names
    - Should specify a unique string prefix
    - E.g. “linux, network-index”
  - Property values
    - An array of zero or more bytes that contains information associated with the property
    - E.g. <empty>, <u32>, <u64>, <string> …

DeviceTree Structure

- Standard Properties
  - Property name: compatible
  - Value type: <stringlist>
  - The “compatible” property defines the specific programming model for the device
  - E.g. compatible = “fsl, mpc8641”, “ns16550”

DeviceTree Structure

- Standard Properties
  - Property name: phandle
  - Value type: <u32>
  - The “phandle” property
    - specifies a numerical ID for a node that is unique within the devicetree
  - The “phandle” property value
    - Another device node could reference the pic node with a phandle value of 1

```
pic@10000000 {
    phandle = <1>;
    interrupt-controller;
    reg = <0x10000000 0x100>;
};
```

```
another-device-node {
    interrupt-parent = <1>;
};
```

DeviceTree Structure

- Standard Properties
  - Property name: reg
  - Value type: <prop-encoded-array> (address, length) pairs
  - The “reg” property
    - Describes the address of the device’s resource within the address space defined by its parent bus
    - Means the offsets and lengths of memory-mapped IO register blocks (addresses on root node are CPU real addresses)
    - E.g. reg <0x3000 0x20 0xFE00 0x100>
      - A device in a SoC has two blocks of registers, a 32-byte block at offset 0x3000 and a 256-byte block at offset 0xFE00

DeviceTree Structure

- Standard Properties
  - Property name: range
  - Value type: <empty> or <prop-encoded-array>
  - The “range” property
    - Provides a mean of defining a translation between the child-bus-address and parent-bus-address
  - The “range” property value
    - <empty> specifies that the parent and child address space is identical

DeviceTree Structure

- Standard Properties
  - 1024KB range of address space
  - A child node addressed at physical 0x0
  - Maps to parent address 0x0e0000000
  - The serial device node can be addressed by a load or store at address 0xe0004600

```
soc {
    compatible = "simple-bus";
    #address-cells = <1>;
    #size-cells = <1>;
    ranges = <0x0 0xe0000000 0x00100000>;

    serial@4600 {
        device_type = "serial";
        compatible = "ns16550";
        reg = <0x4600 0x100>;
        clock-frequency = <0>;
        interrupts = <0xA 0x8>;
        interrupt-parent = <&ipic>;
    };
};
```

DeviceTree Structure

- Interrupts and Interrupt Mapping
  - A logical interrupt tree (directed acyclic graph) represents the hierarchy and routing of interrupts in the platform hardware
  - **Interrupt-parent property**
    - The physical wiring of an interrupt source to an interrupt controller
  - Each interrupt generating device contains **an interrupt property**
    - Describe one or more interrupt sources (interrupt specifier)for that device

DeviceTree Structure

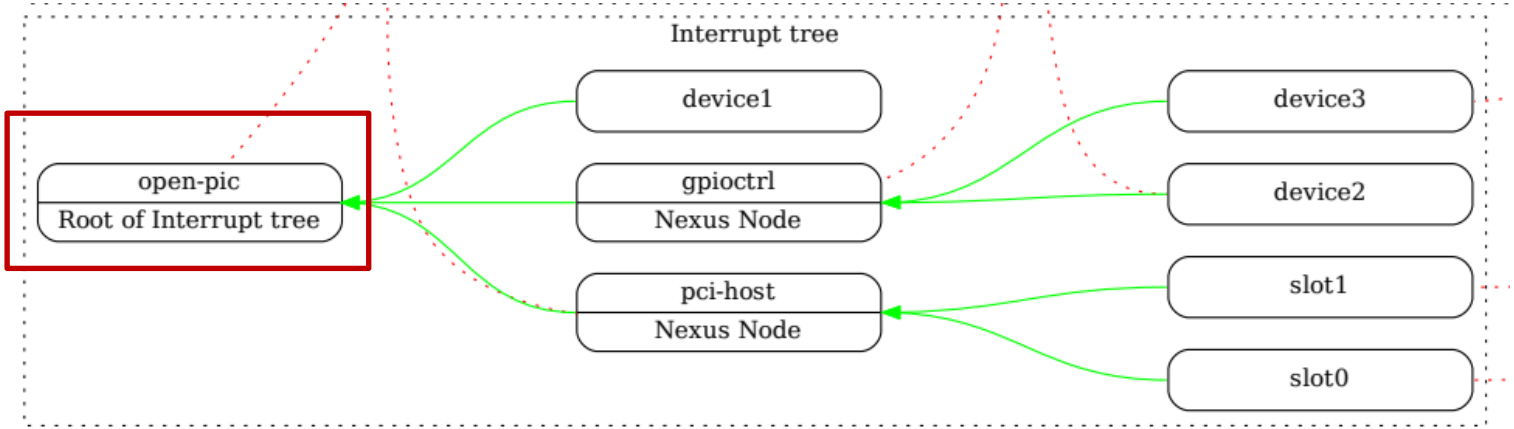
- Interrupts and Interrupt Mapping
  - **The #interrupt-cells property**
    - Used by the root of an interrupt domain
    - Define the number of <u32> values used to encode an interrupt specifier
  - **The interrupt domain**
    - *An interrupt controller*
      - A physical device and will need a driver to handle interrupts routed through it

DeviceTree Structure

- Interrupts and Interrupt Mapping
  - **The interrupt domain**
    - *An interrupt nexus*
      - Defines a translation between one interrupt domain and another
      - The translation is based on both domain/bus-specific information

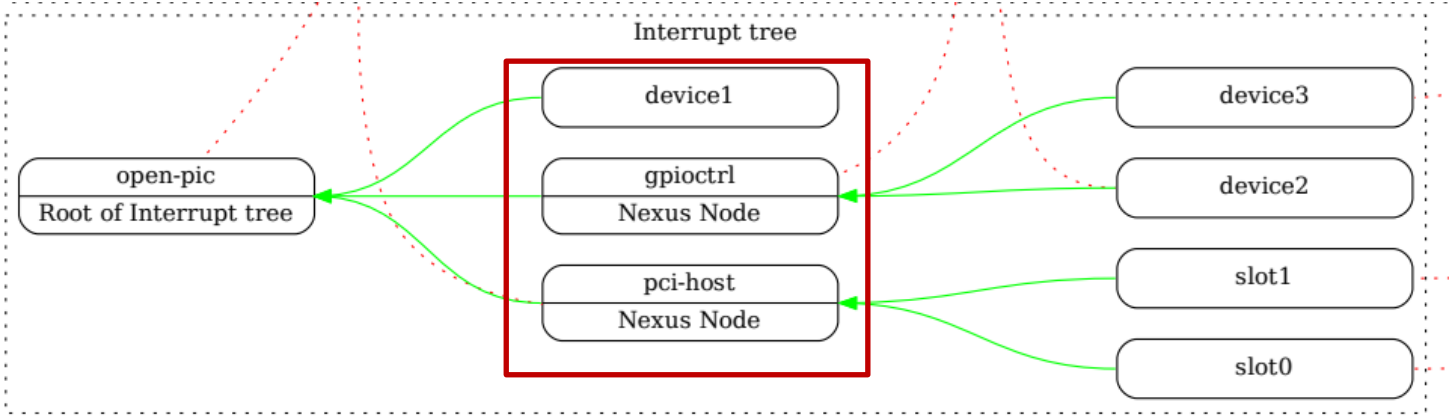
DeviceTree Structure

- Interrupts and Interrupt Mapping
  - The open-pic interrupt controller is the root of the interrupt tree
  - The interrupt tree root has three children
    - device 1, PCI host controller, GPIO controller



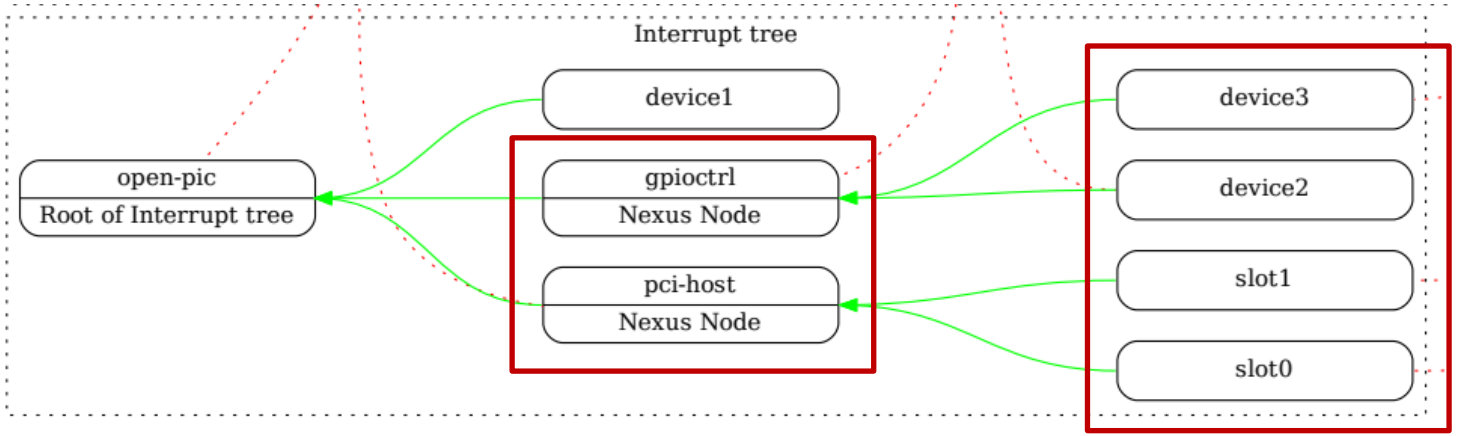
DeviceTree Structure

- Interrupts and Interrupt Mapping
  - The interrupt domains exist
    - One rooted at the open-pic node
    - One at the PCI host bridge node
    - One at the GPIO controller node



DeviceTree Structure

- Interrupts and Interrupt Mapping
  - There are two nexus nodes
    - One at the PCI host bridge
    - One at the GPIO controller



DeviceTree Structure

- Interrupt Mapping Example
  - The interrupt routing for two PCI slots
    - IDSEL 0x11, 0x12
    - INTA/B/C/D pins for slots 1 and 2 are wired to the Open PIC interrupt controller

```
open-pic {
    clock-frequency = <0>;
    interrupt-controller;
    #address-cells = <0>;
    #interrupt-cells = <2>;
};

pci {
    #interrupt-cells = <1>;
    #size-cells = <2>;
```

```
#address-cells = <3>;
interrupt-map-mask = <0xf800 0 0 7>;
interrupt-map = <
    /* IDSEL 0x11 - PCI slot 1 */
    0x8800 0 0 1 &open-pic 2 1 /* INTA */
    0x8800 0 0 2 &open-pic 3 1 /* INTB */
    0x8800 0 0 3 &open-pic 4 1 /* INTC */
    0x8800 0 0 4 &open-pic 1 1 /* INTD */
    /* IDSEL 0x12 - PCI slot 2 */
    0x9000 0 0 1 &open-pic 3 1 /* INTA */
    0x9000 0 0 2 &open-pic 4 1 /* INTB */
    0x9000 0 0 3 &open-pic 1 1 /* INTC */
    0x9000 0 0 4 &open-pic 2 1 /* INTD */
```

DeviceTree Structure

- Interrupt Mapping Example
  - The first row of the interrupt-map table
    - Child unit address: 0x8800 0 0
    - Child interrupt specifier: 1
    - Interrupt parent: &open-pic
    - Parent unit address: <empty>
      #address-cells = <0>
    - Parent interrupt specifier: 2 1

```
#address-cells = <3>;
interrupt-map-mask = <0xf800 0 0 7>;
interrupt-map = <
    /* IDSEL 0x11 - PCI slot 1 */
    0x8800 0 0 1 &open-pic 2 1 /* INTA */
    0x8800 0 0 2 &open-pic 3 1 /* INTB */
    0x8800 0 0 3 &open-pic 4 1 /* INTC */
    0x8800 0 0 4 &open-pic 1 1 /* INTD */
    /* IDSEL 0x12 - PCI slot 2 */
    0x9000 0 0 1 &open-pic 3 1 /* INTA */
    0x9000 0 0 2 &open-pic 4 1 /* INTB */
    0x9000 0 0 3 &open-pic 1 1 /* INTC */
    0x9000 0 0 4 &open-pic 2 1 /* INTD */
```

DeviceTree Structure

- Device node
  - One /cpus node
  - At least one /memory node
    - Describes the physical memory layout for the system
      - RAM: starting address 0x0, length 0x80000000 (2GB)
      - RAM: starting address 0x000000001, length 0x100000000 (4GB)

```
memory@0 {
    device_type = "memory";
    reg = <0x000000000 0x00000000 0x00000000 0x80000000
           0x000000001 0x00000000 0x00000001 0x00000000>;
};
```

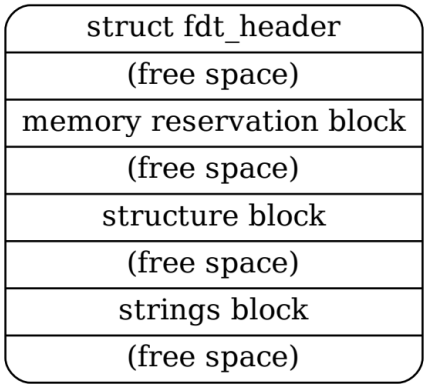
DeviceTree Structure

- Device node
  - An example of a /cpus node with one child cpu node

```
cpus {
    #address-cells = <1>;
    #size-cells = <0>;
    cpu@0 {
        device_type = "cpu";
        reg = <0>;
        d-cache-block-size = <32>; // L1 - 32 bytes
        i-cache-block-size = <32>; // L1 - 32 bytes
        d-cache-size = <0x8000>; // L1, 32K
        i-cache-size = <0x8000>; // L1, 32K
        timebase-frequency = <82500000>; // 82.5 MHz
        clock-frequency = <825000000>; // 825 MHz
    };
};
```

Flattened DeviceTree (DTB) Format

- The Devicetree Blob (DTB) format
  - A flat binary encoding of devicetree data (pointerless data structure)
  - Used to exchange devicetree data between software program
  - E.g. firmware will pass DTB to the OS kernel
  - Locate at an 8-byte-aligned address



Flattened DeviceTree (DTB) Format

- Flattened Devicetree Header Fields
  - All the header fields are 32-bit integers (big-endian format)

```
struct fdt_header {
    uint32_t magic;
    uint32_t totalsize;
    uint32_t off_dt_struct;
    uint32_t off_dt_strings;
    uint32_t off_mem_rsvmap;
    uint32_t version;
    uint32_t last_comp_version;
    uint32_t boot_cpuid_phys;
    uint32_t size_dt_strings;
    uint32_t size_dt_struct;
};
```

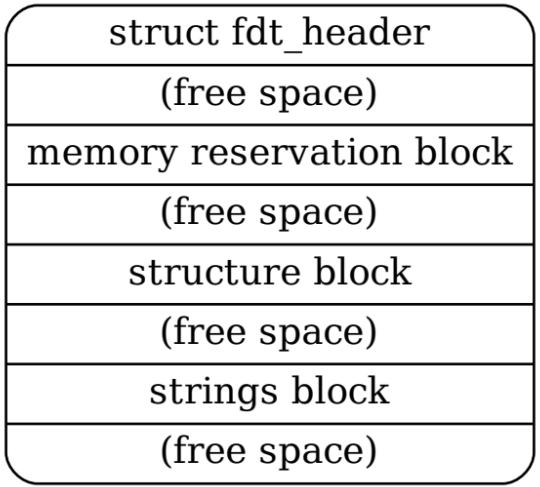
Contain the value 0xd00dffeed

Total size of devicetree data structure

Physical ID of the system's boot CPU

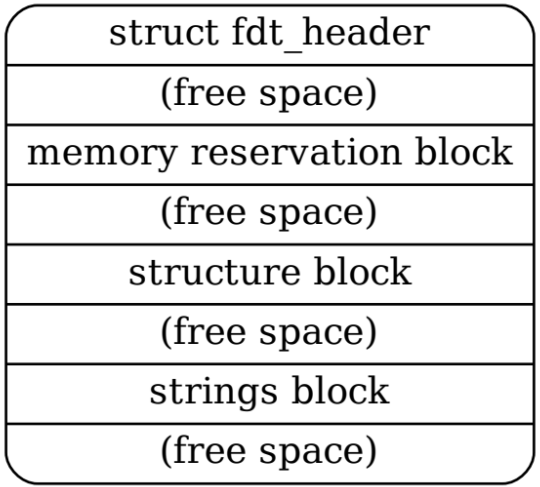
Flattened DeviceTree (DTB) Format

- The Devicetree Blob (DTB) format
  - **Memory reservation block**
    - Used to protect vital data structures from being overwritten by the client program
    - E.g. Translation control entry (TCE) tables initialized by a DTSpec boot program



Flattened DeviceTree (DTB) Format

- The Devicetree Blob (DTB) format
  - **Structure Block**
    - Describe the structure and contents of devicetree itself
    - Has a sequence of tokens with data
    - Each token is located at a 4-byte aligned offset from the beginning of devicetree blob



Flattened DeviceTree (DTB) Format

- The Devicetree Blob (DTB) format
  - **Structure Block**
    - The structure block is composed of a sequence of pieces, each beginning with a token (32-bit integer)
    - Five tokens types
      - FDT_BEGIN_NODE (0x00000001)
        - Marks the beginning of a node’s representation
      - FDT_END_NODE (0x00000002)
        - Marks the end of a node’s representation

Flattened DeviceTree (DTB) Format

- The Devicetree Blob (DTB) format
  - **Structure Block**
    - Five tokens types
      - FDT_PROP(0x00000003)
        - Marks the beginning of one property in the devicetree
      - FDT_NOP (0x00000004)
        - Will be ignored by any program parsing the devicetree
      - FDT_END (0x00000009)
        - Mark the end of the structure block

Flattened DeviceTree (DTB) Format

- The Devicetree Blob (DTB) format
  - **Structure Block**
    - Tree structure
      - The devicetree structure is represented as a linear tree
      - The representation of each node begins with an FDT_BEGIN_NODE token and ends with an FDT_END_NODE token

Takeaway Questions

- Which mode locates the devicetree?
  - (A) M-mode
  - (B) U-mode
  - (C) S-mode
- Which devices are described by the devicetree?
  - (A) GPIO controllers
  - (B) PCI
  - (C) UARTs