设备树

- **1、引进**
  - 驱动程序的编写
    - 1. 数据提取，确定寄存器等
    - 2. 写驱动，主要是file_operations
    - 3. 模块的init和exit
    - 4. 编写测试程序
  - 关键问题
    - **如何确定寄存器？？？**
    - 1. 传统架构：在代码写死（宏）
      - 定义设备结构，包含引脚资源和 cdev
      - 驱动可以直接操作
    - 2. 目标设备驱动模型：在platform_device写死，不同设备有不同的pdev
      - 定义platform_device，其中resource和platform_data 包含引脚资源
      - 驱动从上面获取资源
    - 3. 设备树：提取出来，在设备树定义，驱动通过内核提供的接口获取
      - 定义设备树，编译为dtb，内核启动时解析dtb为device_node，最后编译platform_device，
      - 驱动通过内核提供取资源
  - 设备树文档
    - **内核源码/Documentation/devicetree/bindings**
- **2、格式和编译**
  - DTS
    - 基本语法
      - 节点格式
        - [label:] node name@[unit-address] {
          [properties definitions]
          [child nodes]
        };
      - 属性格式
        - // 布尔型属性
        - [label:] property-name = value;
        - // 无数据属性（相当于一个标志）
        - [label:] property-name;
    - 整体结构
      - /dts-v1/;
        /memreserve/ [address] [length];
        [label:] / {
        [properties definitions]
        [child nodes]
        };
        // 文件开始
        // 内存保留区域
        // 根节点开始
        // 属性定义
        // 子节点
        // 根节点结束
        // 文件结束
    - 重要默认属性
      - compatible：用于和驱动进行匹配
      - model：区分配置相以型号不同的设备
      - #address-cells：子节点的reg，用几个u32描述地址
      - #size-cells：子节点的reg，用几个u32描述大小
      - memory：内存节点
      - chosen：启动参数bootargs节点
      - cpu：cpu节点
    - 节点的引用
      - phandle引用：节点内部使用phandle属性，数值引用
      - label引用：节点起个标签名，&引用
  - DTB
    - DTB编译后的二进制文件，内核启动可以直接读取
    - 存放格式是**大端序**，以四个部分依次排列以
      - DTB的大小
      - **struct boot_param_header（文件头）**
      - **其他部分在DTB的偏移**
      - DTB的版本信息
      - ...等等
    - alignment gap（对齐填充）
      - 以数据按照特定的字节（4字节字）填充空白
    - memory reserve map（内存保留映射表）
      - 记录需在内存里用于DTB的，内核不能动
    - **device-tree structure（设备树结构数据）**
      - 节点的层次关系
        - 0x00000001(节点的开始) + 节点名称
        - 0x00000002(节点的结束)
      - 每个节点的属性和值
        - 0x00000003(属性的开始) + struct fdt_prop_header(属性长度和偏移) + 实际val
      - 节点之间的引用关系
        - 0x00000009(设备树结束)
    - device-tree strings（字符串存储区）
      - 所有字符串（如属性名、节点名）的存放位置，其他的引用这里的偏移即可
  - 转换
    - DTS -> DTB（编译）
      - dtc -I dts -O dtb -o output.dtb input.dts
    - DTB -> DTS（反编译）
      - dtc -I dtb -O dts -o output.dts input.dtb
    - DTC
      - 内核自带工具
      - 你可以自己下载
- **3、内核处理设备树**
  - 设备树数据划分
    - 平台识别(platform identification)
    - 运行时填充(runtime configuration)
    - 设备填充(device population)
  - 整体流程
    - 主要是根据上面的内容来划分整体过程的
    - DTB -> device_node -> platform_device
    - 第一步：head.S的初始阶段早处理
      - bootloader启动内核，设置三个寄存器
      - r0：一般为0
      - **r1：machine id（关键1，设置机器类型）**
      - **r2：ATAGS或DTB的开始地址（关键2，设置DTB地址）**
    - 第二步：平台信息的处理
      - 根节点的compatible属性
      - 内核每个machine_desc的dt_compat
      - 两个比较，确定最终的machine_desc
      - **形**设置当前硬件平台的机器信息
    - 第三步：运行时配置信息处理
      - 主要是扫描运行时配置信息
      - chosen节点，获取bootargs
      - address-cells和size-cells节点，定义子节点地址和大小
      - memory节点，解析系统物理内存的起始地址和大小
      - 1、初始化内存保留区域，设置内存保留区域（memreserve）
      - 2、扁平整个FDT，分配内存
      - 3、为设备节点分配device_node，设置内容
        - /* drivers/of/fdt.c */
        - void * __init unflatten_dt(void *blob,
        - int (*mk_node)(const void *blob,
        - int offset,
        - const char *name,
        - int depth,
        - void *data),
        - void (*dt_alloc)(void *data,
        - unsigned long size,
        - unsigned long align));
        - EXPORT_SYMBOL_GPL(unflatten_dt);
        - /* drivers/of/fdt.c */
        - void * __init unflatten_device_tree(void);
        - EXPORT_SYMBOL_GPL(unflatten_device_tree);
      - 4、为节点的属性分配property，设置属性
        - /* drivers/of/fdt.c */
        - static void * unflatten_dt_alloc(void *data,
        - unsigned long size,
        - unsigned long align);
    - 第四步：dtb转换为device_node（unflatten）
    - 第五步：device_node转换为pdev
      - 核心问题：
        - 1、那些device_node可以转换为platform_device？
        - 2、怎么转换？
      - 问题1
        - 根节点下 含有compatible属性的子节点**可转**
        - 节点的compatible和simple-bus、simple-mfd、isa、arm,amba-bus、其子节点有compatible**也可以转**
        - Or、isa和该节点下的子节点 **不能转**
      - 问题2
        - reg、interrupts属性 ---> **platform_device.resource**
        - 其他属性 ---> **platform_device.dev.of_node**
      - 内核处理流程
        - 设置初始化：
        - 1. 注册platform_bus_type
        - 2. 注册platform_bus_type的匹配函数
        - 3. 注册platform_bus_type的probe函数
        - 4. 注册platform_bus_type的uevent函数
        - 5. 注册platform_bus_type的dev_groups
        - 6. 注册platform_bus_type的pm_ops
        - 7. 注册platform_bus_type的shutdown函数
        - 8. 注册platform_bus_type的dev_attrs
        - 9. 注册platform_bus_type的bus_attrs
        - 10. 注册platform_bus_type的drivers_kset
        - 11. 注册platform_bus_type的devices_kset
        - 12. 注册platform_bus_type的drivers_autoprobe
        - 13. 注册platform_bus_type的match_table
        - 14. 注册platform_bus_type的dev_release
        - 15. 注册platform_bus_type的dev_uevent
        - 16. 注册platform_bus_type的dev_attr_driver_name
        - 17. 注册platform_bus_type的dev_attr_modalias
        - 18. 注册platform_bus_type的dev_attr_numa_node
        - 19. 注册platform_bus_type的dev_attr_driver_probe
        - 20. 注册platform_bus_type的dev_attr_driver_probe_all
        - 21. 注册platform_bus_type的dev_attr_driver_probe_preferred
        - 22. 注册platform_bus_type的dev_attr_driver_override
        - 23. 注册platform_bus_type的dev_attr_modalias_show
        - 24. 注册platform_bus_type的dev_attr_driver_name_show
        - 25. 注册platform_bus_type的dev_attr_numa_node_show
        - 26. 注册platform_bus_type的dev_attr_driver_probe_show
        - 27. 注册platform_bus_type的dev_attr_driver_probe_all_show
        - 28. 注册platform_bus_type的dev_attr_driver_probe_preferred_show
        - 29. 注册platform_bus_type的dev_attr_driver_override_show
        - 30. 注册platform_bus_type的dev_attr_modalias_store
        - 31. 注册platform_bus_type的dev_attr_driver_probe_store
        - 32. 注册platform_bus_type的dev_attr_driver_probe_all_store
        - 33. 注册platform_bus_type的dev_attr_driver_probe_preferred_store
        - 34. 注册platform_bus_type的dev_attr_driver_override_store
        - 35. 注册platform_bus_type的dev_attr_numa_node_store
        - 36. 注册platform_bus_type的dev_attr_driver_probe_store
        - 37. 注册platform_bus_type的dev_attr_driver_probe_all_store
        - 38. 注册platform_bus_type的dev_attr_driver_probe_preferred_store
        - 39. 注册platform_bus_type的dev_attr_driver_override_store
        - 40. 注册platform_bus_type的dev_attr_modalias_store
        - 41. 注册platform_bus_type的dev_attr_numa_node_store
        - 42. 注册platform_bus_type的dev_attr_driver_probe_store
        - 43. 注册platform_bus_type的dev_attr_driver_probe_all_store
        - 44. 注册platform_bus_type的dev_attr_driver_probe_preferred_store
        - 45. 注册platform_bus_type的dev_attr_driver_override_store
        - 46. 注册platform_bus_type的dev_attr_modalias_store
        - 47. 注册platform_bus_type的dev_attr_numa_node_store
        - 48. 注册platform_bus_type的dev_attr_driver_probe_store
        - 49. 注册platform_bus_type的dev_attr_driver_probe_all_store
        - 50. 注册platform_bus_type的dev_attr_driver_probe_preferred_store
        - 51. 注册platform_bus_type的dev_attr_driver_override_store
        - 52. 注册platform_bus_type的dev_attr_modalias_store
        -