

基于MTK6762平台Android适配多机型方案

V 1.0.0

**百富计算机技术(深圳)有限公司**

**PAX Computer Technology（Shenzhen）Co.,Ltd.**

版权声明

Copyright © 2000-2015百富计算机技术（深圳）有限公司保留所有权利。未经百富计算机技术（深圳）有限公司书面许可,不得重新制作或散发本文档的任何内容。尽管百富计算机技术（深圳）有限公司努力确保本文档内容的准确性，但仍可能包含错误或遗漏。本文档所包含的信息如有更改，恕不另行通知。本文中的事例和样例程序仅用作说明演示目的，可能并不适用于您的要求，在您的软件投入商业使用前，请您仔细验证事例或样例程序的适用性。

修改记录

|  |  |  |  |
| --- | --- | --- | --- |
| 日期 | 版本号 | 备注 | 作者 |
| 2021-09-28 | V1.0 | 初稿 | 李斌 |
|  |  |  |  |

目录

[概述 4](#_Toc83888472)

[1.1 编写目的 4](#_Toc83888473)

[1.2 设计背景 4](#_Toc83888474)

[1.3 设计目标 5](#_Toc83888475)

[1.4 硬件主板信息 6](#_Toc83888476)

[2 Linux DT设备树 7](#_Toc83888477)

[3 Android DTS Overlay 8](#_Toc83888478)

[3.1 设备树overlay规则 8](#_Toc83888479)

[3.2 调试手段 9](#_Toc83888480)

[4 Android dtbo 10](#_Toc83888481)

[4.1 分割DT 10](#_Toc83888482)

[4.2 编译主DT和叠加DT 10](#_Toc83888483)

[4.3 对DT进行分区 11](#_Toc83888484)

[4.4 在引导加载程序中运行 11](#_Toc83888485)

[5 MTK6762兼容方案 13](#_Toc83888486)

# 概述

## 编写目的

在MTK 产品设计过程中，会出现多个产品公用同一种MTK 芯片，他们产品使用同一套代码，但是由于硬件上的设计差异，导致重新建立分支代码，更改驱动的DTS配置信息，重新编译程序，下载不同的刷机包。对于开发和维护的工作量造成指数级增长，基于MTK6762平台Android适配多机型方案能有效解决这种问题，一套软件，一个刷机包，可以适配不同的硬件设备。从而大大减轻工作量，减少维护成本。

## 设计背景

在设计过程中，有两个项目同时使用MTK6762 Android 11。考虑到维护难度问题题，我们设计了两个软件的兼容问题，但是两个项目在硬件形态上有所区别(如图1-1所示)：

1.项目A中的GPIO口，而在项目B中被当成UART口使用

2.项目A中spi口，在项目B中被当成i2c口使用。

当以前系统内核kernel设备树配置中，只有一种设备树配置，而兼容方案的目的，就是配置多个设备树将其加载到系统，然后根据硬件差异，让启动器去选择正确的设备树进行配置。

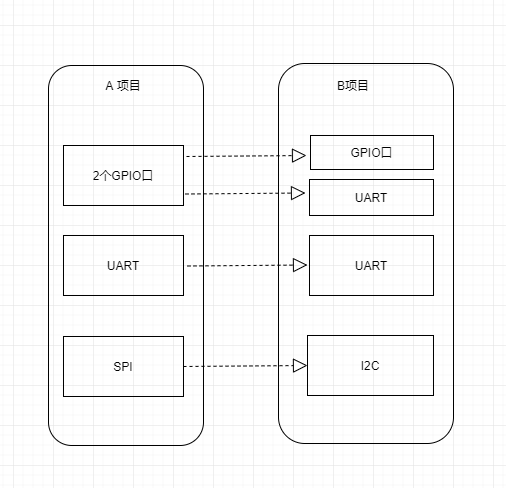


图1‑1 MTK SoC启动流程

## 设计目标

* 分析MTK Android 11源代码编译原理；
* 分析MTK DTS编译原理；
* 分析MTK 开机流程；
* MTK lk模块的dts加载；
* 编码语言：C、Python、Makefile。

## 硬件主板信息

* 处理器：MT6762；
* 内存：3GB DDR3；
* 存储空间：64GB EMMC Flash；
* USB接口：1ports；
* 电源输入：5V；
* SD卡：1 port。
* Camera：2 port；
* Display：1 port;
* 通信协议口：UART；

# Linux DT设备树

DTS即Device Tree Source设备树源码，Device Tree是一种描述硬件的数据结构，它起源于OpenFirmware (OF)。在Linux 2.6中，ARM架构的板极硬件细节过多地被硬编码在arch/arm/plat-xxx和arch/arm/mach-xxx，比如板上的platform设备、resource、i2c\_board\_info、spi\_board\_info以及各种硬件的platform\_data，这些细节的实现对于内核来说只是业务相关的内容，而且变动太大，完全可以抽离出来。而采用Device Tree后，硬件的配置可以动态的加载到 Linux，而不再需要在kernel中进行大量的程序。

在做内核开发过程中，linus团队在linux kernel release之后都会有两周时间内，kernel各个部分的维护者都会提交各自的patch， 将各个分支的代码合入kernel的主分支，这个时候linus作为主要负责人，需要合并分支代码到kernel source tree中。造成合并代码很慢，于是团队就引入设备树的概念。

Linux内核从3.x开始引入设备树的概念，用于实现驱动代码与设备信息相分离。在设备树出现以前，所有关于设备的具体信息都要写在驱动里，一旦外围设备变化，驱动代码就要重写。引入了设备树之后，驱动代码只负责处理驱动的逻辑，而关于设备的具体信息存放到设备树文件中，相对与变化比较少的驱动逻辑，用户只需更新变动较大的驱动信息更新，驱动开发者只需要修改设备树文件信息，不需要改写驱动代码。比如在ARM Linux内，一个.dts(device tree source)文件对应一个ARM的machine，一般放置在内核的"arch/arm/boot/dts/"目录内，比如MTK板级设备树文件就是"arch/arm64/boot/dts/mediatek"目录。这个文件可以通过$make dtbs命令编译成二进制的.dtb文件供内核驱动使用。

基于同样的软件分层设计的思想，由于一个SoC可能对应多个machine，如果每个machine的设备树都写成一个完全独立的.dts文件，那么势必相当一些.dts文件有重复的部分，为了解决这个问题，Linux设备树目录把一个SoC公用的部分或者多个machine共同的部分提炼为相应的.dtsi文件。这样每个.dts就只有自己差异的部分，公有的部分只需要"include"相应的.dtsi文件, 这样就是整个设备树的管理更加有序。

# Android DTS Overlay

在传统开发模式中，管理一些硬件资源的分配和模块的加载，从而缓解多个驱动程序争用系统资源的问题，这个机制通常是由设备树来完成的：在开发之前根据项目需求，提前确定系统中所有用到的硬件设备。在设备树中把所有的外围设备信息以设备树特定的语法进行描述，在设备树被编译为dtb文件后，被linux内核加载使用。

可以看到，在传统开发过程，一旦硬件资源发生变化，就要重新修改、编译、下载设备树。比较极端的情况是：当项目中要支持多种的硬件模块，而不同模块间往往会共用某些系统资源(如IO引脚)。一旦系统要兼容模块任意组合使用，那么随着模块数量增加，需要编译的设备树数量将爆炸增长。

因此，使用传统设备树是不利于项目的维护和扩展的。内核为了解决这个提出了一套新的解决方案，那就是Device Tree Overlays，中文上可理解为”设备树插件”。 它的核心原理是，通过扩展传统的设备树语法，使得各个硬件模块的信息可以独立地用新的设备树语法来描述。这样一来，传统的主设备树中只需要保留最基础的硬件信息(主要是cpu和内存)，其他模块单独编译成”设备树插件”。在系统实际使用时，根据实际应用情景，需要用到哪些硬件模块就把对应的设备树插件加入到主设备树即可。

“设备树插件”无疑提高了系统的可维护性和减少了大量的重复工作，目前，我们已经把常见的硬件模块都编译成了“设备树插件”,比如LCD、HDMI、WiFi等等。

## 设备树overlay规则

规则1：对于同一个节点的设置情况，dts中的配置会覆盖dtsi中的配置；

规则2：对于节点的修改，先引用后修改。例如：需要在reserved-memory节点中添加一个新的节点或者直接修改reserved-memory节点的属性，都需要先引用reserved\_memory节点（注意节点的引用名与节点名可以不一致），引用reserved-memory节点，并删除了ranges属性，删除了hyp\_mem节点，新增了kboot\_uboot\_logmem节点；

规则3：只有引用申明的节点，在dtsi中“&节点名”才会生效，否则引用点将不生效；例如：firmware节点下fstab 节点的定义如下，对于同一个节点的设置情况，dts文件中的内容会覆盖dtsi中的。

## 调试手段

在调试的过程中，没有达到预期时，需要先确定修改有没有编译到对应的dtbo.img中，就需要反编译dtbo.img：

* 反编译工具：反编译工具代码中自带，只需要初始化一下环境变量就可以使用。初始化指令如下：
* 反编译dtb.img：dtc-I dtb -O dts dtb.img -o dtsi.txt
* 反编译dtbo.img：
* mkdtimg dump dtbo.img -b dtbo
* dtc -I dtb -O dts dtbo.00 -o dtsi.txt

# Android dtbo

在做Android8时，发现在dts中引入了dto。设备树叠加层（DTO）可让主要的设备树（DTB）叠加到设备树上。使用DTO的引导程序可以维护系统芯片（SOC）DT,并动态叠加针对特定设备的DT，从而向树中添加节点并对先用树中的属性进行更改。也就是SOC的设备节点作为DTB，其他设备作为DTO，DTO可以对DTB中的节点进行引用和修改。实现DTO包括分割设备树，编译，分区和运行。

## 分割DT

首先将设备树分割成两部分：

* 主DT。由SOC供应商提供的SOC公用部分和默认配置。
* 叠加DT。由原始设计制造商(ODM)/原始设备制造商(OEM)提供的设备专属配置。

## 编译主DT和叠加DT

* 要编译主DT，请执行以下操作：
  + 将主dts编译为.dtb文件。
  + 将.dtb文件刷写到引导加载程序在运行时可访问的分区。
* 要编译叠加DT，请执行以下操作：
  + 将叠加DT .dts编译为.dtbo文件。虽然文件格式与已格式化为扁平化设备树的.dtb文件相同，但是用不同的文件扩展名可以将其与主DT分开来。
  + 将.dtbo文件刷写到引导加载程序在运行时可访问的分区。

## 对DT进行分区

在闪存中确定加载程序在运行时可访问和可信的位置以放入.dtb和.dtbo。主DT的实例位置：

* 引导分区的一部分，已附加到内核(image.gz)
* 单独的DT blob(.dtb)，位于专用的(dtb)中。

## 在引导加载程序中运行

* 将 .dtb 从存储加载到内存中
* 将 .dtbo 从存储加载到内存中
* 用 .dtbo 叠加 .dtb 以形成合并的 DT
* 启动内核（已给定合并 DT 的内存地址）

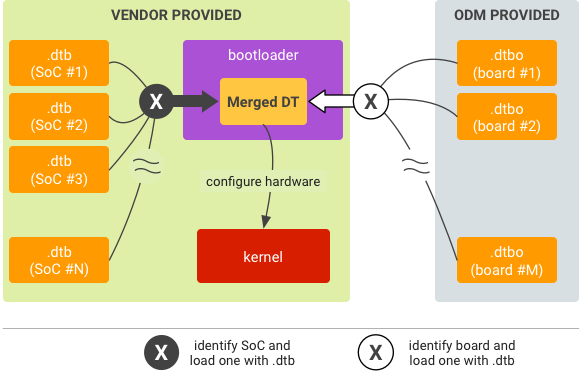


图 4‑1 MTK dtbo去选择流

# MTK6762兼容方案

基于MTK SoC芯片MTK6762，我们设计了M8、M50两款硬件产品，运行Android 11操作系统，为了降低产品软件维护成本，需要实现一套Android 11操作系统同时运行在M8、M50两款硬件产品上，于是我们设计了采用Device Tree Overlays的策略，在系统开机阶段通过读取硬件产品上的ID脚，以此来判定当前系统是M8、M50，进而加载对应的dtbo，从而实现一套系统兼容多套硬件的方案。

硬件兼容实现流程(图5-1，图5-2)：

* 编译过程中将多个硬件设备的设备树打包dtbo
* 硬件通过8个GPIO总线定义不同硬件产品
* MTK LK模块启动过程中读取GPIO状态，判断当前硬件机型
* 根据机型选择对应机型的dtbo，在cmdline 中增加机型信息，将机型参数传入kernel
* Kernel 加载init进程，init根据机器类型，加载不同的init.rc配置文件
* Init根据配置，加载不同硬件服务，设置屏幕的分辨率，加载设备驱动

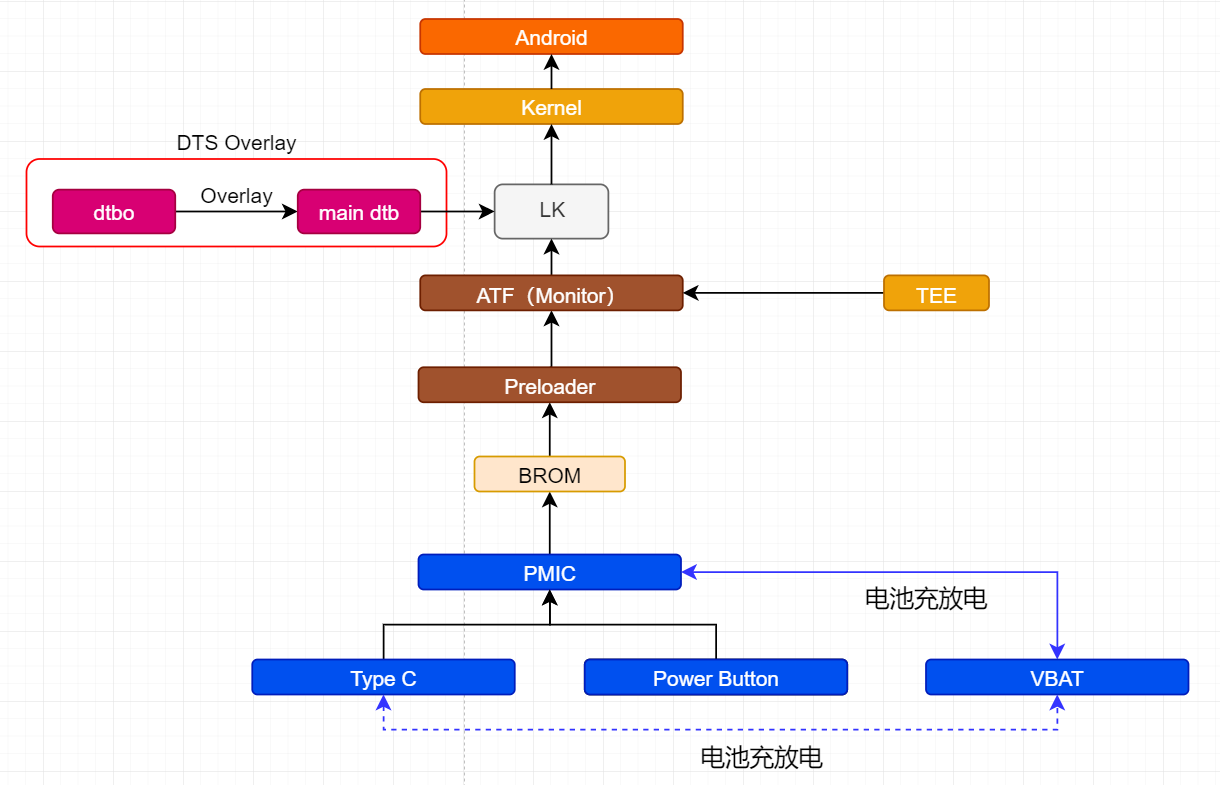


图 5‑1 mtk启动流程

* 通过dts overlay技术，将所有的dtbo合并到main dtb中



图 5‑2 dtbo选取方案

**基于MTK6762平台Android适配多机型方案**



