文档编号：IMXLXRM
第 LF5.10.72_2.2.0 版，2021 年 12 月 17 日

i.MX Linux 参考手册

目录

图目录

表目录

第 1 章
介绍

1.1 概述

i.MX 系列 Linux 板级支持包（BSP）支持 i.MX 应用处理器上的 Linux 操作系统（OS）。

此软件包的目的是支持 i.MX 系列集成电路（IC）及其关联平台上的 Linux 操作系统。它提供了将标准开源 Linux 内核连接到 i.MX 硬件所需的软件。目标是使 i.MX 客户能够快速构建基于使用 Linux 操作系统的 i.MX 芯片的产品。

BSP 不是平台或产品参考实现。它不包含产品所需的所有产品专用的驱动、与硬件无关的软件栈、图形用户界面（GUI）组件、Java 虚拟机（JVM）和应用。其中一些是基本内核的组成部分，以其原始的开源形式提供。

BSP 不适用于芯片验证。虽然它可以在这方面发挥作用，但 BSP 功能和在 BSP 上运行的测试覆盖率不足以取代传统的芯片验证测试套件。

1.1.1 软件基础

i.MX BSP 基于 Linux 内核官网（www.kernel.org）上的 Linux 内核 5.10.72 版。它通过恩智浦提供的功能得到了增强。

在 Linux 上，要在 Yocto Project 环境中使用菜单配置更改配置，请使用 bitbake，如下所示：

```
bitbake linux-imx -c menuconfig
```

1.1.2 功能

下表描述了 BSP 针对特定平台支持的功能。

表 1. BSP 支持的功能

功能	说明	章节	适用平台
机器特定层（Machine-Specific Layer）			
MSL	机器特定层（MSL）支持中断、定时器、存储器映射、GPIO/IOMUX、SPBA、SDMA。 • 中断 GIC：Linux 内核包含常见的 Arm GIC 中断处理代码。 • 定时器（GPT）：通用定时器（GPT）被设置为按照编程生成中断，以提供 OS 走时标记（tick）。Linux 操作系统通过用于定时延迟、测量、事件、告警、高分辨率定时器特性等的各种功能来促进定时器的使用。Linux OS 定义了 OS 走时（tick）定时器所需的 MSL 定时器 API，并且不会在内核走时（tick）实施之外公开。	机器特定层（MSL）	所有平台

下页继续.....

表 1. BSP 支持的功能（续）

功能	说明	章节	适用平台
	• GPIO/EDIO/IOMUX：MSL 中的 GPIO 和 EDIO 组件在各种驱动以及系统的配置和使用之间提供了一个抽象层，包括 GPIO、IOMUX 和外部电路板 I/O。IO 软件模块是特定于电路板的，并作为一组独立的文件驻留在 MSL 层中。I/O 配置更改集中在 GPIO 模块中进行，因此在各种驱动中都不需要进行更改。 • SPBA：共享外设总线仲裁器（SPBA）在多个主设备之间提供仲裁机制，允许访问共享外设。 MSL 下的 SPBA 定义了 API，以允许不同的主设备获取或释放共享外设的所有权。		
通用驱动			
热驱动	热驱动将以特定频率监测 SoC 的温度，以保护 SoC。它定义了 3 个触发点：critical、hot 和 active。	热驱动	所有平台
OProfile	OProfile 是适用于 Linux 系统的系统范围的分析器，能够以较低的开销分析所有正在运行的代码。	OProfile	所有平台
脉宽调制器	脉宽调制器（PWM）有一个 16 位计数器，经过了优化，可使用存储的样本音频图像生成声音并生成音调。	脉宽调制器（PWM）	所有平台
传感器	传感器包括加速度传感器、环境光和磁力计传感器。	传感器	所有平台
看门狗	看门狗定时器模块可避免意外挂起、无限循环情况或编址错误，能够防止系统故障。	看门狗	所有平台
DMA 引擎			
SDMA API	智能直接存储器存取（SDMA）API 驱动控制 SDMA 硬件，并为其他驱动提供 API，用于在 MCU、DSP 和外设之间传输数据。	智能直接存储器存取（SDMA）API	所有平台
APBH-Bridge-DMA	AHB 到 APBH 和 AHB 到 APBX DMA 都支持可配置的 DMA 描述链。	带 DMA 的 AHB 到 APBH 桥（APBH-Bridge-DMA）	所有平台
电源管理驱动			
低级电源管理	低级电源管理驱动实施硬件特定的操作，以满足电源要求并节省电力。不同平台的驱动实施通常不同。该功能由 DPM 层使用。	低级电源管理（PM）驱动	所有平台

下页继续.....

表 1. BSP 支持的功能（续）

功能	说明	章节	适用平台
动态总线频率	总线频率驱动动态管理各种系统频率，以降低功耗。	动态总线频率驱动	i.MX 6 和 i.MX 7
CPU 频率	CPU 频率调整允许改变 CPU 的时钟速度。	CPU 频率	所有平台
PMIC PF 稳压器	PF 稳压器驱动提供对电源稳压器的低级控制、电压电平的选择以及稳压器的启用/禁用。	PF_Regulator	所有平台
Anatop 稳压器	Anatop 稳压器驱动提供对电源稳压器的低级控制。	Anatop 调节器	i.MX 6 和 i.MX 7
连接驱动			
ENET 1588 堆栈	根据 IEEE 标准 1588 实施精确时间协议（PTP）。	快速以太网控制器（FEC）驱动	所有平台
快速以太网控制器	ENET 驱动执行全套 IEEE 802.3/以太网 CSMA/CD 媒体访问控制和通道接口功能。	快速以太网控制器（FEC）驱动	所有平台
FlexCAN	FlexCAN 驱动提供发送和接收 CAN 消息的接口。	FlexCAN 驱动	i.MX 6Quad、i. MX 6Dual、i.MX 6DualLite、i. MX 6Solo、i. MX 6UltraLite、i. MX 6SoloX
MediaLB	MediaLB 是一种 PCB 或芯片间通信总线，允许应用访问 MOST 网络数据或与其他应用进行通信。	MediaLB	i.MX 6SoloX i. MX 6Quad i.MX 6Dual
PCIe	PCI Express 硬件模块可以配置为根复合体或 PCIe 端点。	PCIe	所有平台
视频			
捕获	摄像头和捕获接口的摄像头概述。	捕获概述	所有平台
显示	显示概述	显示概述	所有平台
VPU	视频处理单元（VPU）是一种多制式视频解码器和编码器，可以执行各种视频格式的解码和编码。	视频处理单元（VPU）驱动	i.MX 6QuadPlus/ Quad/Dual/ Solo 和 i. MX 8

下页继续.....

表 1. BSP 支持的功能（续）

功能	说明	章节	适用平台
JPEGENC/ JPEGDEC	JPEG-E-X 和 JPEG-D-X 核心是独立的高性能 8 位和 12 位 JPEG 编码器和解码器，分别用于静态图像和视频压缩/解压缩应用。	JPEG 编码器和解码器	i.MX 8QuadXPlus、8QuadMax
音频驱动			
ALSA Sound	Advanced Linux Sound Architecture（ALSA）是一种声音驱动，为兼容 ALSA 和 OSS 的应用提供执行音频播放和录制功能的方式。	ALSA Sound 驱动	所有平台
ASRC	异步采样率转换器（ASRC）驱动提供访问异步采样率转换器模块的接口。	异步采样率转换器（ASRC）	所有平台
S/PDIF	S/PDIF 驱动是在 Linux ALSA 子系统下设计的。它实施了一个用于发送的播放设备和一个用于接收的捕获设备。	索尼/飞利浦数字接口（S/PDIF）驱动	所有平台
存储 MTD 驱动			
SPI NOR MTD	SPI NOR MTD 驱动使用 SPI 接口支持 Atmel data Flash。	SPI NOR 闪存技术设备（MTD）驱动	所有平台
NAND MTD	NAND MTD 驱动与支持 UBIFS、CRAMFS 和 JFFS2UBI 以及 UBIFSCRAMFS 和 JFFS2 文件系统的集成 NAND 控制器对接。	NAND GPMI Flash 驱动	i.MX 6Quad、i. MX 6Dual、i.MX 6DualLite、i. MX 6Solo、i. MX 6UltraLite、i. MX 7Dual
SATA	SATA AHCI 驱动基于 Linux 内核块设备基础设施的 LIBATA 层。	SATA 驱动	i.MX 6QuadPlus、i.MX 6Quad、i. MX 6Dual、i.MX 8QuadMax、i.MX 8QuadXPlus
总线驱动			
I2C	低功耗 I2C 总线驱动与 I2C 总线对接，通过 I2C 总线传输数据。	IC 间（I2C）驱动	所有平台

下页继续.....

表 1. BSP 支持的功能（续）

功能	说明	章节	适用平台
eCSPI	低级增强型可配置串行外设接口（ECSPI）驱动将自定义的内核空间 API 连接到两个 ECSPI 模块。	增强型可配置串行外设接口（ECSPI）驱动	所有平台
MMC/SD/SDIO - uSDHC	MMC/SD/SDIO 主驱动实施了与 eSDHC 对接的标准 Linux 驱动接口。	MMC/SD/SDIO 主驱动	所有平台
连接驱动			
UART	通用异步收发器（UART）驱动将串行驱动 API 连接到所有 UART 端口。	通用异步收发器（UART）驱动	所有平台
USB	USB 驱动与 ARC USB-OTG 控制器对接。	CHIPIDEA USB	所有平台

1.2 目标读者

本文档面向将 i.MX Linux® 操作系统板级支持包（BSP）移植到客户特定产品的个人。

读者应该对 Linux 内核内部组件、驱动型号和 i.MX 处理器有一定的了解。

1.2.1 规范

本文档使用以下符号规范：

- Courier 字体等宽用于表示命令、命令参数、代码示例以及文件和目录名称。
- *斜体*表示可替换的命令或函数参数。
- **粗体**表示函数名称。

1.2.2 定义和缩略语

下表定义了本文档使用的缩略语。

表 2. 定义和缩略语

术语	定义
ADC	异步显示控制器
地址转换	从虚拟域到物理域的地址转换
API	应用编程接口
Arm®	高级 RISC 机器处理器架构
AUDMUX	数字音频多路复用器：为主串行接口和外设串行接口之间的语音、音频和同步数据路由提供可编程互连
BCD	二进码十进数
总线	多个设备之间通过数据线的路径
总线负载	总线繁忙时间的百分比

下页继续.....

表 2. 定义和缩略语（续）

术语	定义
CODEC	编码器/解码器或压缩/解压算法，用于对各类数据进行编码和解码（或压缩和解压缩）
CPU	中央处理单元：用于描述处理核心的通用术语
CRC	循环冗余校验：数据通信的误码保护方法
CSI	摄像头传感器接口
DCNANO	显示控制器 Nano：一种高性能图形核心，可用于从帧缓冲区读取渲染的图像
DFS	动态频率缩放
DMA	直接存储器存取：可以发起存储器到存储器数据传输的独立块
DPM	动态电源管理
DCSS	显示控制器子系统
DP	显示端口：与 HDMI 类似的 IP
DPU	显示处理器单元
DSI	显示串行接口
DRM	显示渲染管理器或数字版权管理器
DRAM	动态随机存取存储器
DVFS	动态电压频率缩放
EMI	外部存储器接口：控制系统中所有主设备的所有 IC 外部存储器访问（读/写/擦除/编址）
Endian	指存储器中数据的字节排序。Little endian 表示数据的最低有效字节存储在比最高有效字节低的地址中。在 big-endian 中，字节的顺序是相反的。
EPDC	电泳显示控制器
EPIT	增强型周期中断定时器：一个 32 位的设置和忘记定时器，能够以固定的间隔提供精确的中断，而处理器干预最少
FCS	帧检查器序列
FIFO	先进先出
FIPS	联邦信息处理标准：美国国家标准与技术研究院（NIST）发布的美国政府技术标准。当联邦政府对安全性和互操作性等方面有强制要求却没有可接受的行业标准时，NIST 就会制定 FIPS。
FIPS-140	加密模块的安全要求：联邦信息处理标准 140-2 (FIPS 140-2)描述了美国联邦政府要求 IT 产品应满足的内容敏感但不保密（SBU）使用要求。
闪存	一种类似于 EEPROM 的非易失性存储设备，擦除只能在块或整个芯片中完成。
闪存路径	ROM 引导程序中指向可执行闪存应用的路径

下页继续.....

表 2. 定义和缩略语（续）

术语	定义
刷新（Flush）	实现缓存一致性的过程。指从缓存中删除数据线。该过程包括清洁线路、使其 VBR 失效以及重置标签有效指示符。刷新（Flush）由软件命令触发。
GPIO	通用输入/输出
GPU	图形处理单元
哈希	生成哈希值以访问安全数据。哈希值（简称哈希）也称为消息摘要，是从一串文本生成的数字。哈希值比文本本身要小得多，并且是根据公式生成的，使其他一些文本不可能产生相同的哈希值。
HDMI	高清多媒体接口
I/O	输入/输出
ICE	在线仿真
IP	知识产权
IPU	图像处理单元：支持视频和图形处理功能，并为视频/静态图像传感器和显示器提供接口
IrDA	红外数据组织：一个非营利组织，其目标是制定全球通用的红外无线通信规范。
ISR	中断服务例程
JTAG	JTAG（IEEE® 标准 1149.1）是一种标准，指定如何控制和监测印刷电路板上的兼容设备的引脚
Kill	中止存储器存取
KPP	键盘端口：16 位外设，用作键盘矩阵接口或通用输入/输出（I/O）
LDB	LVDS 显示桥
线	指缓存中与标签相关联的信息单元
LRU	最近最少使用：缓存中的行替换策略
LVDS	低压差分信号
MIPI	移动行业流程接口
MMU	存储器管理单元：负责存储器保护和地址转换的组件
MPEG	运动图像专家组：制定数字视频压缩和音频标准的 ISO 委员会。它也是用于压缩运动图片和视频的算法的名称。
MPEG 标准	运动图片和视频的几种压缩标准： • MPEG-1 针对 CD-ROM 进行了优化，是 MP3 的基础 • MPEG-2 用于数字电视机顶盒和 DVD 等应用中的广播视频 • MPEG-3 并入了 MPEG-2 • MPEG-4 是万维网上的低带宽视频电话和多媒体标准

下页继续.....

表 2. 定义和缩略语（续）

术语	定义
MQSPI	多队列串行外设接口：用于执行配置无线电子系统和选定外设所需的串行编程操作
MSHC	记忆棒主控制器
NAND 闪存	闪存 ROM 技术：NAND 闪存架构是用于存储卡（如 Compact Flash 卡）的两种闪存技术之一（另一种为 NOR）。NAND 最适合需要大容量数据存储的闪存设备。NAND 闪存设备提供高达 512 MB 的存储空间，并提供比 NOR 架构更快的擦除、写入和读取功能
NOR 闪存	参见“NAND 闪存”
PCMCIA	PC 机内存卡国际联合会：一个多公司组织，为小型、信用卡大小的设备（称为 PC 卡）制定了一个标准。有 3 种类型的 PCMCIA 卡，它们有相同的矩形尺寸（85.6 x 54 毫米），但宽度不同
物理地址	物理访问系统存储器的地址
PLL	锁相环：控制振荡器的电子电路，使其在输入或参考信号的频率上保持恒定的相位角（锁定）。
PxP	Pixel Pipeline
RAM	随机存取存储器
RAM 路径	ROM 引导程序中用于下载和执行 RAM 应用的路径
RGB	RGB 颜色模型基于加法模型，其中红光、绿光和蓝光相组合以创建其他颜色。缩写 RGB 来自加色光模型中的 3 种原色
RGBA	RGBA 颜色空间代表红绿蓝阿尔法。Alpha 通道是透明度通道，并且是该颜色空间独有的。RGBA 与 RGB 一样，是一种加色空间，因此放置的颜色越多，图片就越亮。PNG 是最知名的使用 RGBA 颜色空间的图像格式
RNGA	随机数生成器加速器：安全硬件模块，作为安全模块的一部分生成 32 位伪随机数。
ROM	只读存储器
ROM 引导	包含主引导低和异常向量的内部引导代码
RPMSG	远程处理器消息
RTIC	实时完整性检查器：安全硬件模块
SC	系统控制器
SCC	安全控制器：安全硬件模块
SCFW	系统控制器固件
SDMA	智能直接存储器访问
SDRAM	同步动态随机存取存储器
SoC	片上系统
SPBA	共享外设总线仲裁器：一个三对一 IP 总线仲裁器，具有资源锁定机制

下页继续.....

表 2. 定义和缩略语（续）

术语	定义
SPI	串行外设接口：全双工同步串行接口，用于使用四线连接低/中带宽外部设备。SPI 设备通过两条数据线和两条控制线利用主/从关系进行通信：另请参见 *SS*、*SCLK*、*MISO* 和 *MOSI*
SRAM	静态随机存取存储器
SSI	同步串行接口：串行数据传输的标准接口
TBD	待定
UART	通用异步接收器/发射器：与外部设备进行异步串行通信
UID	唯一 ID：处理器和 CSF 中的一个字段，用于标识一台设备或一组设备
USB	通用串行总线：一种支持高速数据传输的外部总线标准。USB 1.1 规范支持 12 Mbps 的数据传输速率，USB 2.0 的最大传输速率为 480 Mbps。一个 USB 端口可用于连接 127 个外围设备，例如鼠标、调制解调器和键盘。USB 还支持即插即用安装和热插拔
USBOTG	USB OTG：USB 2.0 规范的扩展，用于将外围设备相互连接。USB OTG 设备也称为两用外设，根据电缆与设备的连接方式，它们本身可以充当受限主设备或外设，还可以连接到主 PC
VADC	视频模数转换器
VPU	视频处理单元
码字	一组包含 32 位的位

1.3 参考资料

i.MX 有多个软件支持的系列。下面列出了这些系列和每个系列的 SoC。《i.MX Linux 版本说明》介绍了当前版本支持的 SoC。一些以前发布的 SoC 可以在当前版本中构建，但如果它们处于以前的验证级别，则不进行验证。

- i.MX 6 系列：6QuadPlus、6Quad、6DualLite、6SoloX、6SLL、6UltraLite、6ULL、6ULZ
- i.MX 7 系列：7Dual、7ULP
- i.MX 8 系列：8QuadMax、8ULP
- i.MX 8M 系列：8M Plus、8M Quad、8M Mini、8M Nano
- i.MX 8X 系列：8QuadXPlus、8DXL、8DualX

此版本包含以下参考资料和附加信息。

- 《*i.MX Linux®版本说明*》(IMXLXRN)——提供版本信息。
- 《*i.MX Linux®用户指南*》(IMXLUG)——提供有关安装 U-Boot 和 Linux OS 以及如何使用 i.MX 特有功能的信息。
- 《*i.MX Yocto Project 用户指南*》(IMXLXYOCTOUG)——介绍了如何使用 Yocto Project 设置主机、安装工具链和构建源代码以创建镜像的恩智浦开发系统的板级支持包。
- 《*i.MX 机器学习用户指南*》(IMXMLUG)——提供机器学习信息。
- 《*i.MX Linux 参考手册*》(IMXLXRM)——提供有关 i.MX 的 Linux 驱动的信息。
- 《*i.MX 图形用户指南*》(IMXGRAPHICUG)——介绍图形功能。
- 《*i.MX 移植指南*》(IMXXBSPPG)——提供如何将 BSP 移植到新电路板的说明。

- 《*i.MX VPU 应用编程接口 Linux® 参考手册*》(IMXVPUAPI)——提供有关 i.MX 6 VPU 上的 VPU API 的参考信息。
- 《*Harpoon 用户指南*》(IMXHPUG)——介绍 i.MX 8M 系列的 Harpoon 版本。
- 《*面向 i.MX 8QuadMax 的 i.MX 数字驾驶舱硬件分区支持工具*》(IMXDCHPE)——为 i.MX 8QuadMax 提供 i.MX 数字驾驶舱硬件解决方案。
- 《*i.MX DSP 用户指南*》(IMXDSPUG)——提供有关 i.MX 8 的 DSP 的信息。
- 《*i.MX 8M Plus 摄像头和显示器指南*》(IMX8MPCDUG)——提供有关 i.MX 8M Plus 的 ISP 独立传感器接口 API 的信息。

快速入门指南包含电路板及其设置的基本信息。这些指南在恩智浦网站上提供。

- 《SABRE 平台快速入门指南》(IMX6QSDPQSG)
- 《SABRE 电路板快速入门指南》(IMX6QSDBQSG)
- 《i.MX 6UltraLite EVK 快速入门指南》(IMX6ULTRALITEQSG)
- 《i.MX 6ULL EVK 快速入门指南》(IMX6ULLQSG)
- 《SABRE 汽车娱乐中控系统快速入门指南》(IMX6SABREINFOQSG)
- 《i.MX 7Dual SABRE-SD 快速入门指南》(SABRESDBIMX7DUALQSG)
- 《i.MX 8M Quad 评估套件快速入门指南》(IMX8MQUADEVKQSG)
- 《i.MX 8M Mini 评估套件快速入门指南》(8MMINIEVKQSG)
- 《i.MX 8M Nano 评估套件快速入门指南》(8MNANOEVKQSG)
- 《i.MX 8QuadXPlus 多传感器支持工具包快速入门指南》(IMX8QUADXPLUSQSG)
- 《i.MX 8QuadMax 多传感器支持工具包快速入门指南》(IMX8QUADMAXQSG)
- 《i.MX 8M Plus 评估套件快速入门指南》(IMX8MPLUSQSG)

文档可在恩智浦网站（nxp.com）上在线获取。

- 如需了解 i.MX 6 的信息，请访问 nxp.com/iMX6series
- 如需了解 i.MX SABRE 的信息，请访问 nxp.com/imxSABRE
- 如需了解 i.MX 6UltraLite 的信息，请访问 nxp.com/iMX6UL
- 如需了解 i.MX 6ULL 的信息，请访问 nxp.com/iMX6ULL
- 如需了解 i.MX 7Dual 的信息，请访问 nxp.com/iMX7D
- 如需了解 i.MX 7ULP 的信息，请访问 nxp.com/imx7ulp
- 如需了解 i.MX 8 的 信息，请访问 nxp.com/imx8
- 如需了解 i.MX 6ULZ 的信息，请访问 nxp.com/imx6ulz

第 2 章
系统

2.1 机器特定层（MSL）

2.1.1 介绍

机器特定层（MSL）为 Linux 内核提供以下与机器相关的组件。

- 中断，包括 GPIO 和 EDIO（仅在某些平台上）
- 定时器
- 存储器映射
- 通用输入/输出（GPIO），包括某些平台上的 IOMUX
- 时钟
- 共享外设总线仲裁器（SPBA）
- 智能直接存储器访问（SDMA）

2.1.2 中断（操作）

本节介绍设备中断的硬件和软件操作。

2.1.2.1 中断硬件操作

中断控制器控制所有内部和外部中断源，并确定其优先级。默认情况下，所有中断的优先级都相同。

通过配置中断控制器的寄存器，可以启用或禁用每个中断源。

GIC 中有 3 种中断：PPI、SGI 和 SPI。

- PPI 是每个 CPU 的专用外设中断。它只能由每个 CPU 处理。
- SGI 是软件产生的中断。它可以由软件操作触发，也只能由每个 CPU 处理。
- SPI 是共享外设中断，通常是来自 SoC 平台的外部中断源。它可以由所有 CPU 处理。

2.1.2.2 中断软件操作

对于采用 i.MX 6 和 i.MX 7 SOC 的 GIC-400 的 Arm 架构处理器，普通中断和快速中断是两种不同的异常类型。对于 i.MX 6 和 i.MX 7 平台，可以将异常向量地址配置为从低地址（0x0）或高地址（0xFFFF0000）开始。在 Arm 架构上运行的 Linux 操作系统选择高向量地址模型。

对于使用 i.MX 8 SOC 的 GIC-500 的 Arm 架构处理器，异常向量地址定义为 VBAR_Eln +偏移量。偏移量取决于采取中断异常的异常级别。Documentation/arm/Interrupts 文件描述了 Arm 中断架构。

该软件提供了一个处理器特定的中断结构，在 irqchip 结构中定义了回调函数，并导出一个初始化函数，该函数在系统启动期间调用。

表 3. 中断文件

文件	说明
drivers/irqchip/irq-gic.c	带有 GIC-400 的 i.MX 6/7 SoC
drivers/irqchip/irq-gic-v3.c	带有 GIC-500 的 i.MX 8 SoC
drivers/irqchip/irq-imx-irqsteer.c	带有 CONFIG_IMX_IRQSTEER 配置的中断函数
drivers/irqchip/irq-imx-intmux.c	带有 CONFIG_IMX_INTMUX 配置的中断函数
irq-imx-gpcv2.c	带有 CONFIG_IMX_GPCV2 配置的中断函数

2.1.2.3 中断功能

中断实施支持以下功能：

- 中断控制器中断禁用和启用
- 标准 Arm 中断源代码中定义的 Linux 中断架构所需的功能

2.1.2.4 中断源代码结构

中断模块位于 drivers/irqchip。

下表列出了中断的源文件。

表 4. 中断文件

文件	说明
drivers/irqchip/irq-imx-irqsteer.o.c	带有 CONFIG_IMX_IRQSTEER 配置的中断函数。
drivers/irqchip/irq-imx-gpcv2.c	带有 CONFIG_IMX_GPCV2 配置的中断函数。
drivers/irqchip/irq-imx-intmux.c	带有 CONFIG_IMX_INTMUX 配置的中断函数。

2.1.2.5 中断编程接口

设备特定的中断实施导出单个函数。该函数初始化中断控制器硬件，并注册各函数，以便从每个中断源启用和禁用中断。这是通过 struct irqdesc 类型的全局结构 irq_desc 完成的。完成初始化后，驱动可以通过 request_irq()函数使用中断来注册设备专用的中断处理程序。

除了中断控制器支持的本机中断线外，中断的数量也进行了扩展，以支持 GPIO 中断和（在某些平台上）EDIO 中断。这允许驱动使用运行 Linux 操作系统的 Arm 芯片支持的标准中断接口，例如 request_irq()和 free_irq()函数。

2.1.3 定时器

Linux 内核依靠底层硬件为系统定时器（生成周期性中断）和动态定时器（调度事件）提供支持。

系统定时器发生中断后，会执行以下操作：

- 更新系统正常运行时间
- 更新时间
- 如果当前进程已用完其时间切片，则重新安排新进程

- 运行任何已到期的动态定时器
- 更新资源使用和处理器时间统计信息

下表描述了使用的不同定时器。

表 5. 定时器

定时器	说明
通用定时器（GPT）	GPT 配置为以特定间隔（每 10 毫秒）生成周期性中断。 i.MX 6 使用它进入 WFI 模式。由 i.MX 6 和 i.MX 7 使用。
增强型周期中断定时器（EPIT）	在 i.MX 6 和 i.MX 7 上可用。
Arm Arch 定时器	在 i.MX 8 上使用，而不使用 GPT
系统计数器定时器	在 i.MX 8M 和 i.MX 8X 上使用，而不使用 GPT

2.1.3.1 定时器软件操作

定时器软件实施提供一个初始化函数，该函数使用适当的时钟源、中断模式和中断间隔对 GPT 进行初始化。

然后定时器注册其中断服务例程并开始计时。为了实现上一节 “定时器” 所述的目的，需要中断服务例程来为操作系统服务。另一个函数提供最后一次定时器中断所经过的时间。

2.1.3.2 定时器特性

定时器实施支持以下特性：

- Linux OS 提供系统定时器和动态定时器所需的函数。
- 对于 i.MX6 和 i.MX 7，每 10 毫秒生成一次中断，对于 i.MX 8，每 4 毫秒生成一次中断。这基于 CONFIG_HZ_XXX。

2.1.3.3 定时器源代码结构

表 6. 定时器文件

文件	说明
arch/arm/mach-imx/epit.c	增强型周期中断定时器
driver/clocksource/timer-imx-sysctr.c	系统控制器定时器
driver/clocksource/timer-imx-tpm.c	TPM 定时器
drivers/clocksource/timer-imx-gpt.c	通用定时器
drivers/clocksource/arch-arm-timer.c	Arm arch 定时器

2.1.3.4 定时器编程接口

定时器模块利用 4 个硬件定时器来实现时钟源和时钟事件对象。

这通过 struct clocksource 类型的 clocksource_mxc 结构和 struct clockevent_device 类型的 clockevent_mxc 结构来完成。这两种结构都提供了读取当前定时器值和安排下一个定时器事件所需的例程。该模块实施了一个定时器中断例程，该例程为 Linux 操作系统提供定时器事件，用于本章开头提到的目的。

2.1.4 存储器映射

由于 Linux 内核在启用了存储器管理单元（MMU）的虚拟地址空间中运行，设备驱动需要一个预定义的虚拟到物理存储器映射表才能访问设备寄存器。

2.1.4.1 存储器映射硬件操作

MMU 作为 Arm 核心的一部分，提供页表中定义的虚拟到物理地址映射。如需了解更多信息，请参见 Arm 有限公司的《*Arm 技术参考手册*》（TRM）。

2.1.4.2 存储器映射特性

存储器映射实施对存储器映射模块进行编址，为所有 I/O 模块创建物理到虚拟存储器映射。

2.1.5 IOMUX

高度集成处理器的引脚数量有限，可用于多种用途。IOMUX 模块控制一个引脚的使用，以便为不同的目的配置同一个引脚，并由不同的模块使用同一个引脚。这是一种常用方法，既可以满足不同的客户需求又减少了引脚数。没有 IOMUX 硬件模块的平台可以通过 GPIO 模块进行引脚复用。

IOMUX 模块提供多路复用控制，因此每个引脚都可以配置为功能引脚或 GPIO 引脚。功能引脚可细分为主用功能或备用功能。引脚操作由特定的硬件模块控制。GPIO 引脚由用户通过软件控制，并通过 GPIO 模块进行进一步配置。例如，TXD1 引脚可能具有以下功能：

- TXD1——内部 UART1 发送数据。这是该引脚的主要功能。
- UART2 DTR——备用模式 3
- LCDC_CLS——备用模式 4
- GPIO4[22]——备用模式 5
- SLCDC_DATA[8]——备用模式 6

如果在系统集成层面选择硬件模式，则该引脚仅用于该目的，不能通过软件更改。否则，需要对 IOMUX 模块进行配置，以满足系统（电路板）设计规定的特定用途。如果该引脚连接到外部 UART 收发器，并因此用作 UART 数据发送信号，则应将其配置为主用功能。如果该引脚连接到外部以太网控制器用于中断 Arm 核心，则应将其配置为启用了中断的 GPIO 输入引脚。同样，请注意，软件无法控制引脚应具有哪些功能。软件仅根据系统设计配置引脚的使用。

2.1.5.1 IOMUX 硬件操作

以下讨论仅适用于具有 IOMUX 硬件模块的处理器。本节简要介绍 IOMUX 控制器寄存器。如需了解详细信息，请参见《IC 参考手册》的“引脚复用”章节。

- SW_MUX_CTL：选择引脚的主用或备用功能。还可以在适用时启用环回模式。
- SW_SELECT_INPUT：控制引脚输入路径。仅当多个引脚驱动同一内部端口时才需要此寄存器。
- SW_PAD_CTL：控制引脚压摆率、驱动强度、上拉/下拉电阻等。

2.1.5.2 IOMUX 软件操作

IOMUX 软件实施提供了一个 API 用于设置引脚功能和引脚特性。

2.1.5.3 IOMUX 特性

IOMUX 实施对 IOMUX 模块进行编址，以配置硬件支持的引脚。

2.1.5.4 IOMUX 源代码结构

下表列出了 IOMUX 模块的源文件。这些文件位于 drivers/princtrl/freescale 文件夹中。

表 7. IOMUX 文件

文件	说明
drivers/pinctrl/freescale/pinctrl-imx.c	i.MX pinctrl 核心驱动
drivers/pinctrl/freescale/pinctrl-imx6q.c	i.MX 6Quad/DualLite pinctrl 驱动
drivers/pinctrl/freescale/pinctrl-imx6sx.c	i.MX 6SoloX pinctrl 驱动
drivers/pinctrl/freescale/pinctrl-imx6sll.c	i.MX 6SLL pinctrl 驱动
drivers/pinctrl/freescale/pinctrl-imx6ul.c	i.MX 6UltraLite 和 6ULL pinctrl 驱动
drivers/pinctrl/freescale/pinctrl-imx7d.c	i.MX 7Dual pinctrl 驱动
drivers/pinctrl/freescale/pinctrl-imx7ulp.c	i.MX 7ULP pinctrl 驱动
drivers/pinctrl/freescale/pinctrl-imx8qm.c	i.MX 8QuadMax pinctrl 驱动
drivers/pinctrl/freescale/pinctrl-imx8qxp.c	i.MX 8QuadXPlus pinctrl 驱动
drivers/pinctrl/freescale/pinctrl-imx8mq.c	i.MX 8M Quad pinctrl 驱动
drivers/pinctrl/freescale/pinctrl-imx8mm.c	i.MX 8M Mini pinctrl 驱动
drivers/pinctrl/freescale/pinctrl-imx8mn.c	i.MX 8M Nano pinctrl 驱动
drivers/pinctrl/freescale/pinctrl-imx8ulp.c	i.MX 8ULP pinctrl 驱动

2.1.5.5 IOMUX 编程接口

请参见 pinctrl 绑定文档 Documentation/devicetree/bindings/pinctrl/fsl。

- imx-pinctrl.txt
- imx6q-pinctrl.txt
- imx6dl-pinctrl.txt
- imx6sll-pinctrl.txt
- imx6sx-pinctrl.txt
- imx6ul-pinctrl.txt
- imx7d-pinctrl.txt
- imx7ulp-pinctrl.txt
- imx8qm-pinctrl.txt
- imx8qxp-pinctrl.txt
- imx8mq-pinctrl.txt
- imx8mm-pinctrl.txt
- imx8mn-pinctrl.txt

2.1.5.6 通过 GPIO 模块进行 IOMUX 控制

对于多用引脚，GPIO 控制器提供复用控制，以便每个引脚都可以配置为功能引脚或 GPIO 引脚。功能引脚的操作可细分为主用功能或备用功能，由特定的硬件模块控制。如果配置为 GPIO 引脚，则该引脚由用户通过软件控制，并通过 GPIO 模块进行进一步配置。此外，GPIO 引脚还有一些特殊配置（例如基于输出的 A_IN、B_IN、C_IN 或 DATA 寄存器，以及基于输入的 A_OUT 或 B_OUT）。

以下内容适用于通过 GPIO 模块控制引脚复用的平台。

如果在系统集成层面选择硬件模式，则该引脚仅用于该用途，无法通过软件进行更改。否则，需要正确配置 GPIO 模块，以满足系统（电路板）设计规定的特定用途。如果该引脚连接到外部 UART 收发器，则应将其配置为主用功能，或者如果该引脚连接到外部以太网控制器用于中断该核心，则应将其配置为启用了中断的 GPIO 输入引脚。软件无法控制引脚应具有哪些功能。软件仅根据系统设计为该用途配置引脚。

2.1.5.6.1 GPIO 硬件操作

GPIO 控制器模块分为 MUX 控制和 PULLUP 控制子模块。以下各节简要介绍了硬件操作。如需了解详细信息，请参见相关芯片文档。

2.1.5.6.1.1 复用控制

GPIO In Use 寄存器控制 GPIO 模块中的多路复用器。

这些寄存器中的设置选择引脚是用于外设功能还是用于其 GPIO 功能。每个 GPIO 端口专用一个 32 位通用寄存器。这些寄存器可用于 GPIO IOMUX 模块的软件控制。

2.1.5.6.1.2 PULLUP 控制

GPIO 模块的每个 GPIO 端口都有一个 PULLUP control 寄存器（PUEN），用于控制该端口的每个引脚。

2.1.5.6.2 GPIO 软件操作（通用）

GPIO 软件实施提供了一个 API，用于设置引脚功能和引脚特性。

2.1.5.6.3 GPIO 实现

GPIO 实施对 GPIO 模块进行编址，以配置硬件支持的引脚。

2.1.6 通用输入/输出（GPIO）

GPIO 模块提供可配置为输入或输出的通用引脚。当配置为输出时，引脚状态（高或低）可以通过写入内部寄存器来控制。当配置为输入时，可以从内部寄存器读取引脚输入状态。

2.1.6.1 GPIO 软件操作

通用输入/输出（GPIO）模块提供了一个用于配置 i.MX 处理器外部引脚的 API，以及控制 GPIO 中断的中心。

应调用 GPIO 实用程序函数来配置引脚，而不是直接访问 GPIO 寄存器。GPIO 中断实施包含中断服务例程（ISR）注册/注销和发生中断后的 ISR 调度等功能。所有驱动专用的 GPIO 设置功能都应该在 MSL 层的设备初始化期间进行，以提高可移植性和可维护性。该 GPIO 中断在系统启动期间自动初始化。

如果 IOMUX 将某个引脚配置为 GPIO，则还应设置该引脚的状态，因为它不是由专用硬件模块进行初始化的。可能还需要使用引脚控制功能设置引脚上拉、下拉、压摆率等。

2.1.6.1.1 API for GPIO

API for GPIO 列出了 GPIO 实现支持的功能。

GPIO 实现支持以下功能：

- 用于将中断服务例程注册到 GPIO 中断的 API。这是因为 NR_IRQS 定义的中断数量被扩展，以容纳所有能够产生中断的 GPIO 引脚。
- 请求和释放 IOMUX 引脚的函数。如果一个引脚用作 GPIO，则提供另一组请求/释放函数调用。用户在修改引脚状态之前应检查请求调用的返回值，以查看该引脚是否已被保留。不需要引脚时应进行释放函数调用。如需了解详细信息，请参见 API 文档。
- IOMUX 和 GPIO 函数调用的对齐参数传递。在此实施中，iomux_pins 的同一枚举用于 IOMUX 和 GPIO 调用，用户不必确定引脚位于 GPIO 模块中的哪个位位置。
- 由于注册中断不需要特殊的 GPIO 函数调用，只需要对公共驱动（例如以太网和 UART 驱动）进行较少的更改。

2.1.6.2 GPIO 特性

此 GPIO 实现支持以下功能：

- 实现访问 GPIO 硬件模块的功能
- 提供一种方式用于控制 GPIO 信号方向和 GPIO 中断

2.1.6.3 GPIO 模块源代码结构

所有 GPIO 模块源代码都在 GPIO 框架中，位于本章开头所示目录中的以下文件中：

表 8. GPIO 文件

文件	说明
drivers/gpio/gpio-mxc.c	功能实现

2.1.6.4 GPIO 编程接口 2

如需了解更多信息，请参见编程接口的 Linux 源代码目录下的 Documentation/gpio/gpio.txt。

2.1.7 时钟

Linux 时钟框架依靠底层硬件为时钟树管理提供支持。

下表介绍了使用的不同时钟硬件。

文件	说明
时钟控制器模块（CCM）	i.MX 6Quad/DualLite/SoloX/UltraLite/ULL/SLL、i.MX 7Dual、i. MX 8M Quad、i.MX 8M Mini 和 i.MX 8M Nano
外设时钟控制（PCC）和系统时钟发生器（SCG）	i.MX 7ULP
分布式从系统控制器（DSC）	i.MX 8QuadMax/8QuadXPlus

2.1.7.1 时钟软件操作

时钟软件实现提供初始化函数，根据硬件时钟类型和设置对时钟树进行初始化，然后提供时钟操作回调以操作硬件时钟模块。

2.1.7.2 时钟特性

时钟实现根据不同的时钟类型支持以下功能：

- 准备时钟/取消准备时钟。
- 启用/禁用时钟。
- 获取/设置时钟速率。
- 获取/设置父时钟。

2.1.7.3 源代码结构

源代码结构如下所示。

文件	说明
drivers/clk/imx/clk-imx6q.c	i.MX 6Quad/6DualLite 时钟驱动
drivers/clk/imx/clk-imx6sx.c	i.MX 6SoloX 时钟驱动
drivers/clk/imx/clk-imx6ul.c	i.MX 6UltraLite 和 6ULL 时钟驱动
drivers/clk/imx/clk-imx6sll.c	i.MX 6SLL 时钟驱动
drivers/clk/imx/clk-imx7d.c	i.MX 7Dual 时钟驱动
drivers/clk/imx/clk-imx7ulp.c	i.MX 7ULP 时钟驱动
drivers/clk/imx/clk-imx8qm.c	i.MX 8QuadMax 时钟驱动
drivers/clk/imx/clk-imx8qxp.c	i.MX 8QuadXPlus 时钟驱动
drivers/clk/imx/clk-imx8mq.c	i.MX 8M Quad 时钟驱动
drivers/clk/imx/clk-imx8mm.c	i.MX 8M Mini 时钟驱动
drivers/clk/imx/clk-imx8mn.c	i.MX 8M Nano 时钟驱动
drivers/clk/imx/clk-imx8ulp.c	i.MX 8ULP 时钟驱动

2.1.7.4

不同的时钟类型提供不同的时钟操作回调。设备驱动向时钟框架调用标准时钟 API，最终调用到平台时钟驱动中，执行相应时钟节点的操作回调。

2.2 系统控制器

2.2.1 介绍

i.MX 8 和 i.MX 8X 系列提供系统控制器，对硬件的许多底层功能进行抽象，并在执行 SC 固件（SCFW）的 Cortex-M 处理器上运行。本节介绍了 SCFW 的功能和向给其他软件组件公开的 API。

系统控制器的功能如下所示：

- 系统初始化和启动——SCU 只读存储器（ROM）完成从第一个容器加载代码/数据镜像后，SC 固件立即在 SCU 上运行。它负责系统许多方面的初始化。其中包括额外的电源和时钟配置以及资源隔离硬件配置。默认情况下，SC 固件将主用启动核心配置为拥有大部分资源，并启动该启动核心。额外的配置可以通过启动代码来完成。

- 系统控制器通信——系统中的其他软件组件通过公开的 API 库与 SC 进行通信。实施该库是为了通过底层处理器间通信（IPC）机制进行远程过程调用（RPC）。基于硬件的邮箱系统为 IPC 提供便利。为 i.MX8 提供的软件组件（Linux、QNX、FreeRTOS、MCUXpresso SDK）已经包含客户端 API 的端口。其他第三方需要先将该 API 移植到他们的环境中，然后才能使用该 API。移植工具包版本包括现有软件的客户端 API 的存档。这些可用作移植客户端 API 的参考。只需要实施 IPC 层，它将利用消息传递单元（MU）与 SCFW 进行通信。
- 电源管理——电源管理的所有方面，包括电源控制、偏差控制、时钟控制、复位控制和唤醒事件监测，都归为 SC 电源管理服务。
  - 电源控制——SC 固件负责对电源控制和外部电源管理设备的集中管理。它管理电源域的电源状态和电压以及偏置控制。由于电源状态转换，它还根据需要重置外设。通过传达各个资源的电源状态需求，通过 API 实施上述操作。
  - 时钟控制——SC 固件负责时钟控制的集中管理。其中包括振荡器和锁相环（PLL）等时钟源以及时钟分频器、多路复用器和门控。通过传达各个资源的时钟需求，通过 API 实施上述操作。
  - 复位控制——SC 固件负责复位控制。其中包括启动/重新启动分区、获取复位原因以及启动/停止 CPU。

  在使用 SoC 中的任何硬件之前，软件必须首先为资源上电，并启用它需要的任何时钟，否则访问将产生总线错误。
- 资源管理——SC 固件负责管理系统资源的所有权和访问权限。SC 固件支持的资源管理服务的功能如下所示：
  - 管理系统资源，例如 SoC 外设、存储器区域和引脚
  - 允许将资源划分为与不同执行环境相关联的不同所有权组，包括在不同核心、信任区（TrustZone）和管理程序（hypervisor）上运行的多个操作系统
  - 将所有权与来自资源分区内的消息传递单元的请求相关联
  - 允许将存储器划分为不同的存储器区域，然后像其他资源一样进行管理
  - 允许所有者配置对资源的访问权限
  - 配置硬件组件，以提供硬件强制隔离
  - 配置硬件组件，直接控制总线结构上驱动的安全/非安全属性
  - 向其他系统控制器功能提供所有权和访问权限信息（例如，向引脚复用功能提供引脚所有权信息）
  - 有两种资源保护方式。首先，当进行影响特定资源的 API 调用时，SCFW 本身会检查资源访问权限。根据 API 调用，可能要求调用方是所有者、所有者的父级或所有者的祖先。其次，任何可用于实施访问控制的硬件都是基于 RM 状态配置的。其中包括 XRDC2、XRDC 或 RDC 等 IP 的配置，以及 CAAM 等 IP 的管理页面。
- 引脚配置——引脚配置由 SC 固件管理。SC 固件支持的引脚配置功能包括：
  - 配置多路复用器、输入/输出连接和低功耗隔离模式。
  - 配置技术特定的引脚设置，例如驱动强度、上拉/下拉等。
  - 为具有双电压能力的引脚组配置补偿。

- 定时器——许多面向定时器的服务被归入了 SC 定时器服务。其中包括看门狗、RTC 和系统计数器。
  - 看门狗——SC 固件为所有执行环境提供“虚拟”看门狗。功能包括更新看门狗超时、启动/停止看门狗、刷新看门狗、返回看门狗状态，例如可设置的最大看门狗超时、看门狗超时间隔和剩余看门狗超时间隔。
  - 实时时钟——SC 固件负责提供对 RTC 的访问。功能包括设置时间、获取时间和设置告警。
  - 系统计数器——SC 固件负责提供对 SYSCTR 的访问。功能包括设置绝对告警或相对周期性告警。读取是通过每个 CPU 可用的本地硬件接口直接完成的。
- 中断——系统控制器需要一种方法来告知用户异步通知事件。这是通过中断服务完成的。该服务提供 API 来启用/禁用用户中断，并读取挂起中断的状态。读取状态会自动清除任何挂起状态。
- 杂项——在以前的 i.MX 6 和 7 芯片上，使用 IOMUX GPR 寄存器控制杂项功能，信号连接到可配置硬件。此功能正被 DSC GPR 信号所取代。SC 固件负责对 GPR 信号进行编址，以配置这些子系统功能。SC 固件还负责监测各种温度、电压和时钟传感器。
  - 控制——SC 固件提供对杂项控制的访问。功能包括设置（写入）杂项控制的软件请求和获取（读取）杂项控制的软件请求。
  - 安全性——SC 固件提供对多种安全功能的访问，包括镜像加载和身份验证。
  - DMA——SC 固件提供对 DMA 通道分组和优先级功能的访问。
  - 温度——SC 固件提供对温度传感器的访问。

通过这种抽象，其他核心无法直接访问 SCFW 使用的 SoC 参考手册中介绍的某些硬件。其中包括以下内容：

- SCU 子系统中的所有资源（SCU M4、SCU LPUART、SCU LPI2C 等）。
- 通过 MSI 链接从 SCU 子系统访问的所有资源（包括引脚、DSC、XRDC2、eCSR）
- OCRAM 控制器、CAAM MP、eDMA MP 和 LPCG
- DB STC 和 LPCG、IMG GPR
- GIC/IRQSTR LPCG、IRQSTR.SCU 和 IRQSTR.CTI
- SCFW 端口为电路板保留的任何其他资源

系统控制固件（简称 SCFW）的每个发布版本与 i.MX 参考板相关联，并提供了一个移植工具包，该工具包提供了可为新电路板定制的源代码子集。此移植工具包在恩智浦网站（nxp.com）上提供，并包含移植指南。

2.3 引导镜像

2.3.1 介绍

对于 i.MX 6 和 i.MX 7，引导镜像仅使用 U-boot 引导加载程序。对于 i.MX 8 系列中的 SoC，引导镜像更复杂，包括 U-boot 以及成功启动所需的各种固件。本章介绍了 i.MX 8 系列引导加载程序的附加组件。

对于 i.MX 7ULP，引导分区需要 Arm Cortex M-4 SDK 闪存，因为 Arm Cortex M-4 启动 U-boot 引导加载程序，但其他搭载了 Arm Cortex M-4 内核的 i.MX 6 和 i.MX 7 不需要此闪存便可成功启动。

i.MX 8 引导加载程序是使用 imx-mkimage on code aurora forum/上的 imx-mkimage 工具创建的，所有 i.MX 8 系列芯片都需要 imx-atf on code aurora forum/上提供的 Arm Trusted 固件。

如需了解如何使用 imx-mkimage 工具创建 i.MX 引导分区的详细信息，请参见《*i.MX Linux 用户指南*》。此执行工具需要以下组件。

对于 i.MX 8M Quad、i.MX 8M Mini 和 i.MX 8M Nano，需要以下固件：

- Synopys DDR 固件。
- 签名的 HDMI 固件——与 DCSS 驱动相集成。HDMI 固件仅适用于 i.MX 8M Quad。
- Arm Trusted 固件——bl31-*soc*。

对于 i.MX 8QuadMax，需要以下固件：

- 系统控制器固件（SCFW）
- Arm Trusted 固件 - bl31-*soc*
- B0 的 SECO 固件容器映像（ahab-container.img）

对于 i.MX 8QuadXPlus、i.MX 8DualX 和 i.M 8DualXLite，需要以下固件：

- 系统控制器固件（SCFW）
- Arm Trusted 固件 - bl31-*soc*
- SECO 固件容器映像（ahab-container.img）

所有 i.MX 系列都需要 Arm trusted 固件和 U-boot。此外，支持通过 OP-TEE 引导启动 OP-TEE（所有 i.MX 6、7 和 8M 系列）的 i.MX SoC 需要通过构建 optee_ox 创建 tee.bin。

Xen 等 1 型管理程序是引导加载程序的一部分。然而，2 型管理程序（如 jailhouse 和 kvm）并非如此。

2.4 Anatop 稳压器驱动

2.4.1 介绍

Anatop 稳压器驱动提供电源稳压器的低电平控制和电压电平选择。该设备驱动利用稳压器核心驱动访问 Anatop 硬件控制寄存器，只有 i.MX 6 和 i.MX 7 芯片支持它。

2.4.2 硬件操作

芯片上的电源管理单元用于简化外部电源接口，并允许以适当的供电方式配置芯片。电力系统由输入电源及其特性、集成功率变换和控制元件以及最终负载互连和要求组成。

采用 7 个 LDO 稳压器，大大减少了外部电源的数量。如果忽略备用纽扣电池和 USB 输入，则外部电源的数量将减少到两个。外部电源总数中不包括为所需存储器接口供电所需的外部电源。这将根据所选外部存储器的类型而不同。如果不同的 I/O 电源段的 I/O 电压需要不同于上面提供的电压，则可能还需要其他电源为其供电。

一些内部稳压器可以旁路，让外部 PMIC 可以直接供电，以减少电源数量，例如 VDD_SOC 和 VDD_ARM。

2.4.3 软件操作

Anatop 稳压器客户端驱动通过重新配置 Anatop 硬件控制寄存器来执行操作。通过调用具有所需寄存器设置的稳压器核心 API 实现。

2.4.4 驱动特性

Anatop 稳压器驱动基于稳压器核心驱动。此处列出了为稳压器控制提供的服务列表。

- 打开/关闭所有稳压器。
- 设置所有稳压器的值。
- 获取所有稳压器的当前值。

2.4.5 驱动接口详情

可通过稳压器核心驱动的 API 访问 Anatop 稳压器。Anatop 稳压器驱动提供以下稳压器控件：

- 7 个 LDO 稳压器
- 通过设置适当的 Anatop 硬件寄存器值来处理所有稳压器功能。通过调用稳压器核心 API 访问 Anatop 硬件寄存器来完成。

2.4.6 稳压器 API

稳压器电源架构旨在为 Linux 内核中的稳压器和稳流器提供通用接口。它旨在为客户端或消费者驱动提供电压和电流控制，并通过 sysfs 接口向用户空间应用提供状态信息。其目的是允许系统动态控制稳压器输出，以节省电力并延长电池寿命。这适用于稳压器（电压输出可控）和电流吸收器（电流输出可控）。

如需了解更多详细信息，请访问 opensource.wolfsonmicro.com/node/15

在此框架下，大多数电源操作可以通过以下统一的 API 调用完成：

- `regulator_get` 用于查找和获取稳压器的参考：
  - ```
    struct regulator *regulator_get(struct device *dev, const char *id);
    ```
- `regulator_put` 用于释放稳压器源：
  - ```
    void regulator_put(struct regulator *regulator, struct device *dev);
    ```
- `regulator_enable` 用于启用稳压器输出：
  - ```
    int regulator_enable(struct regulator *regulator);
    ```
- `regulator_disable` 用于禁用稳压器输出：
  - ```
    int regulator_disable(struct regulator *regulator);
    ```
- `regulator_is_enabled` 是指启用的稳压器输出：
  - ```
    int regulator_is_enabled(struct regulator *regulator);
    ```
- `regulator_set_voltage` 用于设置稳压器输出电压：
  - ```
    int regulator_set_voltage(struct regulator *regulator, int uV);
    ```
- `regulator_get_voltage` 用于获取稳压器输出电压：
  - ```
    int regulator_get_voltage(struct regulator *regulator);
    ```
    ```

如需了解 Linux 内核中稳压器核心源代码的更多 API 和详细信息，请参见 drivers/regulator/core.c：

2.4.7 源代码结构

Anatop 稳压器驱动位于 drivers/regulator 目录中：

表 9. Anatop 电源管理驱动文件

文件	说明
drivers/regulator/core.c	稳压器接口
drivers/regulator/anatop-regulator.c	Anatop 稳压器客户端驱动

Anatop 稳压器在 arch/arm/boot/dts 中的每个 SoC 专用 dts 文件中注册。

2.4.8 菜单配置选项

在菜单配置中启用以下模块：

- Device Drivers > Voltage and Current regulator support > Anatop Regulator Support.
- System Type > Freescale i.MX on-chip ANATOP LDO regulators.

2.5 电源管理

2.5.1 低电平电源管理

2.5.1.1 介绍

本节介绍了控制低功耗模式的低电平电源管理（PM）驱动。

下面描述了每个受支持的 i.MX 系列的电源管理方式之间的差异。

表 10. 电源管理模式

i.MX 系列	支持的低功耗模式
i.MX 6	RUN、WAIT、STOP 和 DORMANT
i.MX 7	RUN、WAIT、STOP、DORMANT 和 LPSR
i.MX 8M	RUN、IDLE、SUSPEND 和 SNVS
i.MX 8 和 i.MX 8X	无 - 由系统控制器处理

下表列出了不同低功耗模式的详细时钟信息。

表 11. 低功耗模式

模式	核心	模块	PLL	CKIH/FPM	CKIL
RUN	活跃	活跃、空闲或禁用	On	On	On
WAIT	禁用	活跃、空闲或禁用	On	On	On
STOP	禁用	禁用	Off	On	On
LPSR	关闭	禁用	Off	Off	On
DORMANT	关闭	禁用	Off	Off	On
SNVS	关闭	禁用	Off	Off	On

如需了解低功耗模式的详细信息，请参见与 SoC 相关的《应用处理器参考手册》。

2.5.1.2 软件操作

i.MX 6 和 i.MX 7 电源管理驱动将低功耗模式映射到内核电源管理状态，如下所示：

- 待机映射到 STOP 模式，可节省许多电力，因为系统中的所有模块都进入低功耗状态，但 Arm®核心除外，它仍处于通电状态，且存储器处于自刷新模式，以保留其内容。
- Mem（挂起到 RAM）映射到 DORMANT 模式，可节省许多电力，因为系统中的所有模块都进入低功耗状态，但存储器除外，它被置于自刷新模式以保留其内容。如果在 DTB ocrams 节点中定义了“fsl,enable-lpsr”，则 mem 映射到 LPSR 模式而不是 DORMANT，且系统中的所有块都进入断电状态，LPSR、SNVS 和 DRAM 电源域除外。
- 系统空闲映射到 WAIT 模式。
- 如果 Arm Cortex-M4 处理器与 Arm Cortex-A 处理器在内核进入待机/mem 模式之前处于活跃状态，并且 Arm Cortex-M4 处理器未处于低功耗空闲模式，则 Arm Cortex-A 处理器将触发 SoC 进入 WAIT 模式而不是 STOP 模式，以确保 Arm Cortex-M4 处理器可以继续运行。

i.MX 6 和 i.MX 7 电源管理驱动执行以下步骤，进入和退出低功耗模式：

1. 允许 Cortex-A 平台发出深度睡眠模式请求。
2. 如果处于 STOP 或 DORMANT 模式：
   - 对 i.MX 6 CCM_CLPCR 或 i.MX 7 GPC_LPCR_A7_BSC 和 GPC_SLPCR 寄存器进行编址，以设置低功耗控制寄存器。
   - 如果处于 DORMANT 模式，则在 pdn_req 生效时请求关闭 CPU 电源。
   - 当 pdn_req 生效时，请求关闭嵌入式存储器外设电源。
   - 对 GPC 屏蔽寄存器进行编址，以解除屏蔽唤醒中断。
3. 调用 cpu_do_idle 执行等待模式的 WFI 挂起指令。
4. 在 IRAM 中执行 imx6_suspend 或 imx7_suspend。
5. 在 DORMANT 模式下，保存 Arm 上下文，并将 DDR 引脚的驱动强度改为“低”，以最大限度地减少 DDR 引脚中的漏电。
   执行停止模式的 WFI 挂起指令。
6. 生成唤醒中断并退出低功耗模式。在 DORMANT 模式下，恢复 Arm 核心和 DDR 驱动强度。

在 DORMANT 模式下，i.MX 6 和 i.MX 7 可以将 PMIC_STBY_REQ 引脚钳位到 PMIC 并请求改变电压。U-Boot 或机器特定层（MSL）通常根据 i.MX 6 和 i.MX 7 数据说明在 STOP 模式下设置备用电压。

在 i.MX 8M 系列上，电源管理驱动使用以下模式。

- RUN 模式：在此模式下，Quad-A53 CPU 核心处于活跃状态并正在运行。可以关闭某些部分以节省电力。
- IDLE 模式：该模式定义为当没有线程在运行且所有高速器件都处于非活跃状态时，CPU 可自动进入的模式。CPU 可以进入电源门控状态，但保留 L2 数据，DRAM 和总线时钟减少，大多数内部逻辑都被时钟门控但仍保持通电。
- SUSPEND 模式：此模式定义为最省电的模式，所有时钟都关闭，所有不必要的电源都关闭。Cortex-A53 CPU 平台是完全电源门控的。所有可以断电的内部数字逻辑和模拟电路都将关闭。
- SNVS 模式：此模式也称为 RTC 模式。在这种模式下，只有 SNVS 域的电源保持开启，使 RTC 和 SNVS 逻辑处于活跃状态。

在 i.MX 8 和 i.MX 8X 上：

- 低功耗模式管理不受专用硬件模块控制。
- 所有低功耗模式均使用软件在系统控制器固件（SCFW）中实现。
- 当系统挂起时，SCFW 会关闭集群/CPU。

2.5.1.3 源代码结构

下表列出了电源管理驱动源文件。

表 12. 电源管理驱动文件

文件	说明
• arch/arm/mach-imx/pm-imx6.c • arch/arm/mach-imx/suspend-imx6.S • arch/arm/mach-imx/cpuidle-imx6q.c	支持 i.MX 6 QuadPlus/Quad/Dual/Solo 电源管理操作
• arch/arm/mach-imx/pm-imx6.c • arch/arm/mach-imx/suspend-imx6.S • arch/arm/mach-imx/cpuidle-imx6sll.c • arch/arm/mach-imx/imx6sll_low_power_idle.S	支持 i.MX 6 SLL 电源管理操作
• arch/arm/mach-imx/pm-imx6.c • arch/arm/mach-imx/suspend-imx6.S • arch/arm/mach-imx/cpuidle-imx6ul.c • arch/arm/mach-imx/imx6ul_low_power_idle.S	支持 i.MX 6 UltraLite 电源管理操作
• arch/arm/mach-imx/pm-imx6.c • arch/arm/mach-imx/suspend-imx6.S • arch/arm/mach-imx/cpuidle-imx6ul.c • arch/arm/mach-imx/imx6ull_low_power_idle.S	支持 i.MX 6 ULL 电源管理操作
• arch/arm/mach-imx/pm-imx6.c • arch/arm/mach-imx/suspend-imx6.S • arch/arm/mach-imx/cpuidle-imx6sx.c • arch/arm/mach-imx/imx6sx_low_power_idle.S	支持 i.MX 6 SoloX 电源管理操作
• arch/arm/mach-imx/pm-imx7.c • arch/arm/mach-imx/suspend-imx7.S • arch/arm/mach-imx/cpuidle-imx7d.c • arch/arm/mach-imx/imx7d_low_power_idle.S	支持 i.MX 7Dual 电源管理操作
• arch/arm/mach-imx/pm-imx7ulp.c • arch/arm/mach-imx/suspend-imx7ulp.S • arch/arm/mach-imx/cpuidle-imx7.c	支持 i.MX 7ULP 电源管理操作
• drivers/soc/pm-domain-imx8.h	支持 i.MX 8、8X 和 8M 电源域

下页继续.....

表 12. 电源管理驱动文件（续）

文件	说明
Arm Trusted 固件位于 imx-atf on code aurora forum/	支持 i.MX 8、8X、8M 和 8ULP 使用 Arm Trusted 固件进行电源管理操作

2.5.1.4 菜单配置选项

在菜单配置中启用 CONFIG_PM: CONFIG_PM 构建电源管理支持。默认情况下，该选项在 menuconfig 中为 Y，该选项位于：Power management options > Power Management support。

在菜单配置中启用 CONFIG_SUSPEND。CONFIG_SUSPEND 构建了 SUSPEND 支持。在 menuconfig 中，此选项位于以下位置：Power management options > Suspend to RAM and standby

2.5.1.5 编程接口

在源代码结构表中列出的每个 SoC 的 cpu_idle 中查找并搜索 lpm。这些是低功耗模式的 API。实现将系统置于 WAIT 和 STOP 模式所需的所有步骤。

2.5.2 PMIC PF 稳压器

2.5.2.1 介绍

PF100/200/300 是 PMIC 芯片。

PF200/PF3000 基于 PF100，几乎没有变更，它们共享相同的 PF100 驱动。PF100 稳压器驱动提供电源调节器的低电平控制、电压电平选择以及稳压器的启用/禁用。该设备驱动利用 PF100 稳压器驱动访问 PF100 硬件控制寄存器。PF100 稳压器驱动基于稳压器核心驱动，连接到内核 I2C 总线。

PF8100/8200 PMIC 专为 i.MX 8 和 i.MX 8X 系列设计，由于是系统级设备，由系统控制器固件（SCFW）控制。SCFW 为 Linux touch 创建了一些特定的电源，例如“SC_R_BOARD_R0”。

2.5.2.2 硬件操作

PMIC PF 稳压器为应用处理器和外围设备提供参考电压和电源电压。

包括 4 个降压（step down）转换器（最多 6 个独立输出）和一个升压（step up）转换器。降压转换器为处理器核心和其他低压电路（如存储器）提供电源。提供动态电压缩放，允许对处理器核心和/或其他电路的受控电源轨进行调整。

线性稳压器直接由电池或开关电源供电，包括用于 I/O 和外设、音频、摄像头、BT、WLAN 等的电源。命名规范表示典型或可能的用例应用，但在指定能力的指导方针范围内，开关电源和稳压器可用于满足其他系统电源需求。

PF100 的唯一上电事件是 PWRON 置高，PF100 的唯一断电事件是 PWRON 置低。i.MX 6 的 PMIC_ON_REQ 引脚由 i.MX 6 的 SNVS 块控制，将与 PF100 的 PWRON 引脚连接，以控制 PF100 的开/关，从而使系统能够断电。

2.5.2.3 软件操作

PMIC PF 稳压器客户端驱动通过重新配置 PMIC 硬件控制寄存器来执行操作。

某些 PMIC 电源管理操作取决于系统设计和配置。例如，如果系统由 PMIC 以外的电源供电，那么关闭或调整 PMIC 稳压器将不起作用。相反，如果系统由 PMIC 供电，那么使用电源管理驱动和稳压器客户端驱动的任何更改都会影响整个系统的运行或稳定性。

2.5.2.4 驱动特性

PMIC PF 稳压器驱动基于稳压器核心驱动。它为 PMIC 组件的稳压器控制提供以下服务：

- 打开/关闭所有稳压器。
- 设置所有稳压器的值。
- 获取所有稳压器的当前值。

2.5.2.5 稳压器 API

稳压器电源架构旨在为 Linux 内核中的稳压器和稳流器提供通用接口。

它旨在为客户端或消费者驱动提供电压和电流控制，并通过 sysfs 接口向用户空间应用提供状态信息。其目的是允许系统动态控制稳压器输出，以节省电力并延长电池寿命。这适用于稳压器（电压输出可控）和电流吸收器（电流输出可控）。

如需了解更多详细信息，请参见 opensource.wolfsonmicro.com/node/15

在此框架下，大多数电源操作可以通过以下统一的 API 调用完成：

- `regulator_get` 是一个统一 API 调用，用于查找和获取稳压器的参考：

```
struct regulator *regulator_get(struct device *dev, const char *id);
```

- `regulator_put` 是一个统一 API 调用，用于释放稳压器源：

```
void regulator_put(struct regulator *regulator, struct device *dev);
```

- `regulator_enable` 是启用稳压器输出的统一 API 调用：

```
int regulator_enable(struct regulator *regulator);
```

- `regulator_disable` 是用于禁用稳压器输出的统一 API 调用：

```
int regulator_disable(struct regulator *regulator);
```

- `regulator_is_enabled` 是指启用的稳压器输出：

```
int regulator_is_enabled(struct regulator *regulator);
```

- `regulator_set_voltage` 是用于设置稳压器输出电压的统一 API 调用：

```
int regulator_set_voltage(struct regulator *regulator, int uV);
```

- `regulator_get_voltage` 是用于获取稳压器输出电压的统一 API 调用：

```
int regulator_get_voltage(struct regulator *regulator);
```

可在 Linux 内核中的稳压器核心源代码中找到更多 API 和详细信息，位于：

```
drivers/regulator/core.c
```

2.5.2.6 驱动架构

下图显示了 PMIC PF 稳压器驱动的基本架构。

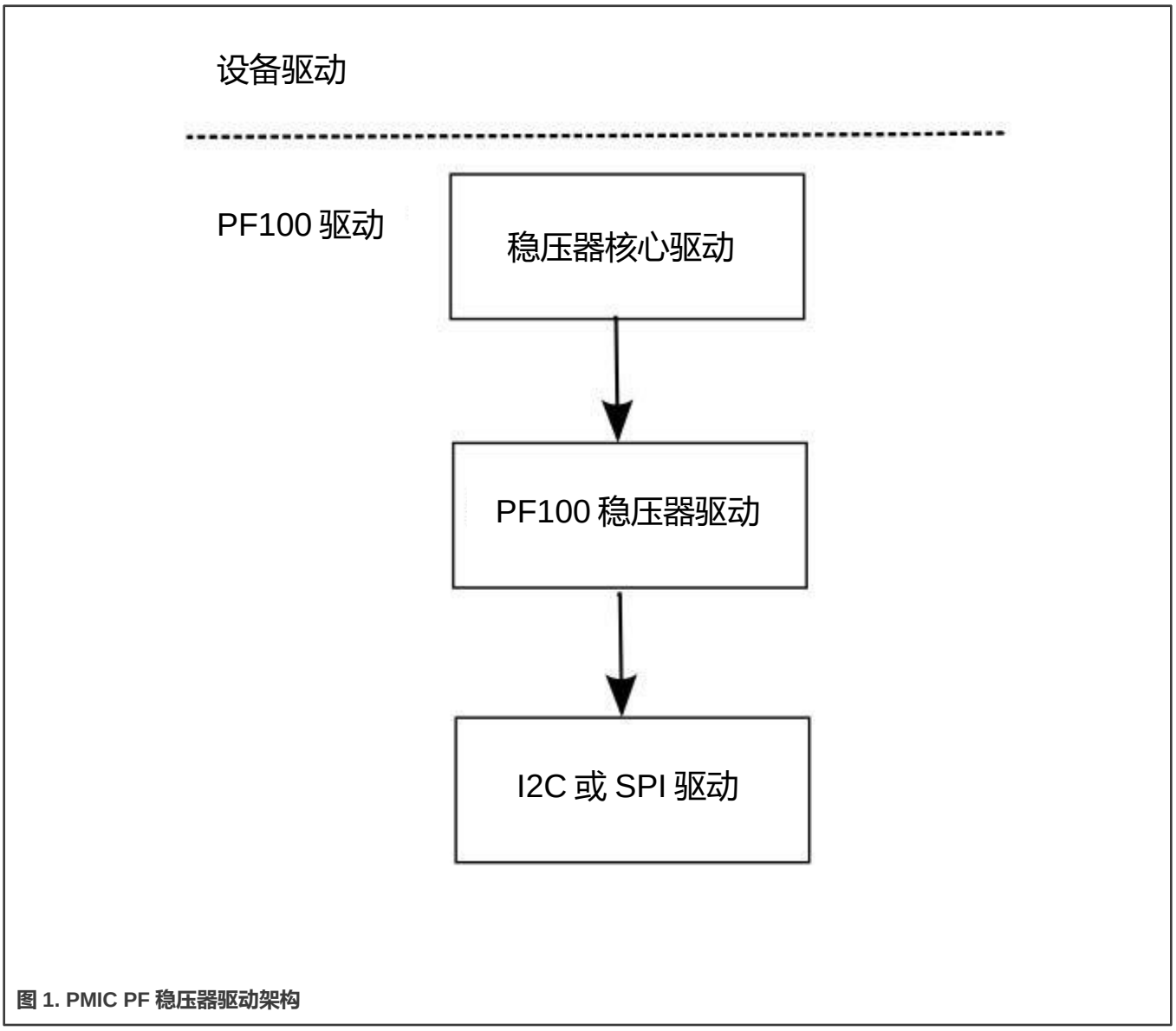


图 1. PMIC PF 稳压器驱动架构

2.5.2.7 驱动接口详情

可通过稳压器核心驱动的 API 访问 PFUZE100 稳压器。

PFUZE100 稳压器驱动提供以下稳压器控制：

- 普通模式下的 4 个降压开关稳压器（最多 6 个独立电源轨）：SW1AB、SW1C、SW2、SW3A、SW3B 和 SW4。
- 可以使用特定寄存器（PFUZE100_SWxSTANDBY）预先将降压开关编程为待机状态。
- 6 个线性稳压器：VGEN1、VGEN2、VGEN3、VGEN4、VGEN5 和 VGEN6。
- 1 个 LDO/开关电源，用于在 i.MX 处理器上支持 VSNVS。
- 1 个 DDR 存储器参考电压的低电流、高精度、电压参考。
- 1 个支持 USB OTG 的升压稳压器。

- PFUZE100 的大多数电源轨已根据硬件设计进行了正确编程。因此，没有使用 PFUZE100 稳压器的内核。PFUZE100 稳压器驱动已经实现了这些稳压器，如果默认的 PFUZE100 值不能满足客户的硬件设计，客户可以自由使用。

2.5.2.8 源代码结构

PFUZE 稳压器驱动位于 drivers/regulator 目录中：

表 13. PFUZE 驱动文件

文件	说明
drivers/regulator/pfuze100-regulator.c	PFUZE100 稳压器客户端驱动的实施。
drivers/regulator/pf1550.c	PFUZE1550 稳压器客户端驱动的实施。
drivers/regulator/pf1550-regulator-rpmsg.c	PFUZE150 稳压器 RPMSG 代码的实施。

没有与 PMIC 相关的电路板文件。一些 PFUZE 驱动代码已移至 U-Boot，例如待机电压设置。部分代码通过 DTS 文件实现。在 Uboot 源中搜索 PFUZE100 并在设备树 dtsi 文件中搜索 pfuze，对于 i.MX 6 和 i.MX7 在 arch/arm/boot/dts 中，对于 i.MX 8M 在 arch/arm64/boot/dts 中。

2.5.2.9 菜单配置选项

在菜单配置中启用以下模块：
Device Drivers > Voltage and Current regulator support > Freescale PFUZE100/200/3000 regulator driver.

2.5.3 CPU 频率调节（CPUFREQ）

2.5.3.1 介绍

CPU 频率调节设备驱动允许动态更改 CPU 的时钟速度。在 CPU 频率改变后，所需电源的电压将改为设备树脚本（DTS）中定义的电压值。这种方法可以降低功耗（从而节省电池电量），因为随着时钟速度的降低，CPU 使用的电量会减少。

2.5.3.2 软件操作

CPUFREQ 设备驱动旨在动态更改 CPU 频率和电压。

如果 DTS 中未定义频率，CPUFREQ 驱动会将 CPU 频率更改为阵列中最接近的更高频率。使用时钟框架 API 控制频率，使用稳压器 API 设置电压。阵列中的 CPU 频率基于引导 CPU 频率。使用交互式 CPU 频率调节器，该调节器无法手动更改。要手动更改 CPU 频率，可使用用户空间 CPU 频率调节器。默认情况下，使用保守式 CPU 频率调节器。

如需了解驱动中实施的功能的更多信息，请参见 API 文档。

要查看 CPU 频率可以更改为哪些值（以 KHz 为单位）（第一列中的值是频率值），请使用以下命令：

```
cat /sys/devices/system/cpu/cpu0/cpufreq/stats/time_in_state
```

要将 CPU 频率更改为使用上述命令给出的值（例如，改为 792 MHz），请使用以下命令：

```
echo 792000 > /sys/devices/system/cpu/cpu0/cpufreq/scaling_setspeed
```

频率 792000 以 KHz 为单位，即 792 MHz。

可以使用以下命令检查最大频率：

```
cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_max_freq
```

使用以下命令查看当前 CPU 频率（以 KHz 为单位）：

```
cat /sys/devices/system/cpu/cpu0/cpufreq/cpuinfo_cur_freq
```

使用以下命令查看可用的调节器：

```
cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_available_governors
```

使用以下命令更改为交互式 CPU 频率调节器：

```
echo interactive > /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
```

2.5.3.3 源代码结构

下表显示了以下目录中提供的源文件和头文件。

表 14. CPUFREQ 驱动文件

文件	说明
drivers/cpufreq/imx6q-cpufreq.c	i.MX 6 CPUFREQ 函数
drivers/cpufreq/imx-cpufreq-dt.c	i.MX 7 和 8 CPUFREQ 函数

如需了解 CPU 频率工作点设置，请参见 SoC 对应的 dtsi 文件，对于 i.MX 6 和 i.MX7 为 arch/arm/boot/dts，对于 i.MX 8、i.MX 8X 和 i.MX 8M 为 arch/arm64/boot/dts。

2.5.3.4 菜单配置选项

为该模块提供了以下 Linux 内核配置：

- CONFIG_CPU_FREQ；在 menuconfig 中，此选项位于：
  - CPU Power Management > CPU Frequency scaling
- 可以选择以下选项：
  - CPU Frequency scaling
  - CPU frequency translation statistics
  - Default CPU frequency governor (conservative)(interactive)
  - Performance governor
  - Powersave governor
  - Userspace governor for userspace frequency scaling
  - Interactive CPU frequency policy governor
  - Conservative CPU frequency governor
  - Schedutil CPU frequency governor
  - CPU frequency driver for i.MX CPUs

2.5.4 动态总线频率

2.5.4.1 介绍

为了降低功耗，总线频率驱动动态管理 i.MX 6、i.MX 7 和 i.MX 8M 系列的各种系统频率。

频率变化对更高层是透明的，不需要驱动或中间件的干预。根据外设的活动和 CPU 负载情况，总线频率驱动在 24 MHz 和其最大频率之间改变 DDR 频率。同样，AHB 频率在 24 MHz 和其最大频率之间变化。

2.5.4.2 操作

总线频率驱动是 Linux BSP 中电源管理模块的一部分。该驱动的主要目的是根据外设活动和 CPU 负载情况调节系统时钟（如 AHB、DDR、AXI 等）的各种工作频率。

2.5.4.3 软件操作

总线频率取决于设备驱动对其操作的请求和释放。驱动将调用总线频率 API 来请求或释放他们想要的总线设定点。总线频率将根据当前请求的外设将系统频率设置为最高频率设定点。

要启用总线频率驱动，请使用以下命令：

```
echo 1 > /sys/bus/platform/drivers/imx_busfreq/soc\:busfreq/enable
```

要禁用总线频率驱动，请使用以下命令：

```
echo 0 > /sys/bus/platform/drivers/imx_busfreq/soc\:busfreq/enable
```

如果 Arm Cortex-M4 处理器与 Arm Cortex-A 处理器一起处于活动状态，Arm Cortex-M4 处理器也会请求或释放其操作的总线频率高设定点。这意味着 Arm Cortex-A 处理器将 Arm Cortex-M4 处理器视为其高速设备之一。

设定点模式执行以下操作：

- 当大多数需要更高频率以获得良好性能的外设处于活跃状态时，使用高频设定点模式。例如，视频播放和图形处理。
- 音频播放设定点模式用于音频播放模式。
- 当系统空闲等待用户输入（显示器关闭）时，使用低频设定点模式。对于 i.MX 8M，当没有外设请求高模式或音频模式时，使用此模式。

下表说明了每个系列的软件设定点。

表 15. 总线频率设定点

SoC	设定点
i.MX 6	• 高频设定点：AHB 为 132 MHz，AXI 为 264 MHz。 • 音频播放设定点：在 i.MX 6 上，AHB 为 25 MHz，AXI 为 50 MHz，对于 DDR，DDR3 为 50 MHz，LPDDR2 为 100 MHz。 • 低频设定点：AHB 为 24 MHz，AXI 为 24 MHz，DDR 为 24 MHz。
i.MX 7Dual	• 高频设定点：AHB 为 135 MHz，AXI 为 332 MHz，DDR 为最大频率。

下页继续……

表 15. 总线频率设置点（续）

SoC	设定点
	• 音频播放设定点：AHB 为 24 MHz，AXI 为 24 MHz，DDR 为 100 MHz。 • 低频设定点：AHB 为 24 MHz，AXI 为 24 MHz，DDR 为 24 MHz。
i.MX 8M	• 高总线频率模式：DDRC 核心时钟设置为 800 MHz。DDRC APB 时钟设置为 200 MHz。NOC 时钟设置为 800 MHz。 主 AXI 时钟设置为 333 MHz，AHB 时钟设置为 133 MHz。 • 音频总线频率模式：DDRC 核心时钟设置为 25 MHz，DDRC APB 时钟设置为 20 MHz，NOC 时钟设置为 100 MHz。主 AXI 主时钟设置为 25 MHz，AHB 时钟设置为 20 MHz。为了省电，DDR PLL 被断电。 • 低总线频率模式：DDRC 核心时钟设置为 25 MHz。DDRC APB 时钟设置为 20 MHz。NOC 时钟设置为 100 MHz。主 AXI 时钟设置为 25 MHz。AHB 时钟设置为 20 MHz。为了省电，DDR PLL 被断电。

2.5.4.4 源代码结构

下表列出了源文件和头文件。

表 16. BusFrequency 驱动文件

文件	说明
arch/arm/mach-imx/busfreq-imx.c	i.MX 6 和 i.MX 7 总线频率函数
include/linux/busfreq-imx.h	i.MX 总线频率 API 定义
arch/arm/mach-imx/busfreq_ddr3.c	i.MX 6 和 i.MX 7 DDR3 总线频率函数
arch/arm/mach-imx/busfreq_lpddr2.c	i.MX 6 和 i.MX 7 LPDDR2 总线频率函数
arch/arm/mach-imx/lpddr2_freq_imx6.S	i.MX 6 LPDDR2 总线频率函数
arch/arm/mach-imx/lpddr2_freq_imx6q.S	i.MX 6 QuadPlus/Quad/Dual/Solo LPDDR2 总线频率函数
arch/arm/mach-imx/lpddr2_freq_imx6sll.S	i.MX 6 SLL LPDDR2 总线频率函数
arch/arm/mach-imx/lpddr2_freq_imx6sx.S	i.MX 6 SoloX LPDDR2 总线频率函数
arch/arm/mach-imx/lpddr3_freq_imx.S	i.MX 6 和 i.MX 7 LPDDR3 总线频率函数
arch/arm/mach-imx/ddr3_freq_imx6.S	i.MX 6 总线频率函数
arch/arm/mach-imx/ddr3_freq_imx6sx.S	i.MX 6 SoloX 总线频率函数
arch/arm/mach-imx/ddr3_freq_imx7d.S	i.MX 7 Dual DDR3 总线频率函数
drivers/doc/imx/busfreq-imx8mq.c	i.MX 8M 总线频率函数

总线频率模式在 SoC dtsi 文件中定义，对于 i.MX 6 和 i.MX 7 为 arch/arm/boot/dts，对于 i.MX 8M 为 arch/arm64/boot/dts。

2.5.4.5 菜单配置选项

此驱动没有菜单配置选项。对于支持总线频率驱动的 SoC，默认情况下包括并启用总线频率驱动。

2.5.5 电池充电

2.5.5.1 介绍

i.MX 6 SABRE SD 电路板的 max8903 充电器支持电池充电。

2.5.5.2 软件操作

无

2.5.5.3 源代码结构

电池充电源文件位于 drivers/power/supply/sabresd_battery.c 中。

2.5.5.4 菜单配置选项

在菜单配置中启用以下模块：
Device Drivers > Power supply class support > Sabresd Board Battery DC-DC Charger for USB and Adapter Power.

2.6 OProfile

2.6.1 介绍

OProfile 是一个系统级分析器，能够以较低的开销分析所有正在运行的代码。

OProfile 由一个内核驱动、一个用于收集样本数据的守护进程（daemon），以及几个用于将数据转换为信息的后分析工具组成。

2.6.1.1 概述

OProfile 利用 CPU 的硬件性能计数器对各种相关统计数据进行分析，这些统计数据也可用于基本的时间分析。

所有代码都经过了分析：硬件和软件中断处理程序、内核模块、内核、共享库和应用。

2.6.1.2 特性

OProfile 具有以下特性。

- 无特别要求——不需要特殊的重新编译或包装库。除非用户想要生成带注释的源代码，否则甚至不需要调试符号（gcc 的 -g 选项）。不需要内核补丁；只需插入模块便可。
- 系统级分析——对系统上运行的所有代码都进行了分析，从而能够分析系统性能。
- 性能计数器支持——支持收集特定代码部分的各种低级数据，并进行关联。
- 调用图支持——OProfile 可以提供 gprof 样式的调用图分析数据。
- 低开销——OProfile 的典型开销为 1-8%，具体取决于采样频率和工作负载。
- 后剖析分析——剖析数据可以在函数级或指令级产生。可以创建带有配置文件信息注释的源代码树。可以生成在整个系统中使用 CPU 时间最多的应用和函数列表。
- 系统支持——可与任何支持 i.MX 的内核配合使用。

2.6.1.3 硬件操作

OProfile 是一个统计连续分析器。

通过定期采样每个 CPU 上的当前寄存器（在中断处理程序中，存储中断时保存的 PC 值），并将运行时 PC 值转换为对编程人员有意义的值，生成配置文件。

OProfile 通过获取采样的 PC 值流，以及中断时正在运行的任务的详细信息，并将这些值转换为相对于特定二进制文件的文件偏移量来执行该操作。因此，每个 PC 值都被转换为二进制镜像偏移的元组（组或集合）。用户空间工具可使用这些数据重建代码的来源，包括特定的汇编指令、符号和源代码行（如存在，通过二进制调试信息）。

定期采样 PC 值可以近似于实际执行的内容和频率，而且通常情况下，这种统计近似值足以反映现实情况。在一般操作中，每个采样中断之间的时间通过固定数量的时钟周期进行调节。这意味着结果能够反映 CPU 在哪些方面花费的时间最多。这是非常有用的性能分析信息源。

Arm CPU 提供能够在硬件层面测量这些事件的硬件性能计数器。通常，这些计数器在发生每个事件时递增一次，并在事件达到预定数量时生成中断。OProfile 可使用这些中断来生成样本，分析结果为哪个代码导致了给定事件的多少个实例的统计近似值。

2.6.1.4 架构专用组件

OProfile 支持特定架构上可用的硬件性能计数器。用于管理这些计数器的设置和管理的详细信息的代码可能位于相关 arch/arm/oprofile 目录的内核源代码树中。架构特定实施通过在初始化时填充 oprofile_operations 结构来运行。这提供了一组操作，如 setup()、start()、stop()等，用于管理性能计数器寄存器中硬件特定的详细信息。

架构代码可用的另一个重要工具是 oprofile_add_sample()。可将中断时获取的特定样本馈送到通用 OProfile 驱动代码。

2.6.1.5 oprofilefs 伪文件系统

Oprofile 实施了一个名为 oprofilefs 的伪文件系统，它在/dev/Oprofile 从用户空间装入。其中包括用于报告和接收用户空间配置的小文件，以及 OProfile 用户空间从中接收样本的实际字符设备。在 setup()时，架构特定的代码可能会添加更多与性能计数器详情有关的配置文件。文件系统还包含一个 stats 目录，其中包含许多用于各种 OProfile 事件的有用计数器。

2.6.1.6 通用内核驱动

通用内核驱动位于 drivers/oprofile 中，是决定 oprofile 在内核中如何运行的核心。通用内核驱动从架构特定的代码中获取样本（通过 oprofile_add_sample()），并缓冲该数据（在转换后的配置中），直到通过/dev/oprofile/buffer 字符设备将数据释放到用户空间守护进程。

2.6.1.7 OProfile Daemon

OProfile 用户空间守护进程获取内核提供的原始数据并将其写入磁盘。它从内核获取单个数据流，并根据多个示例文件（可在/var/lib/oprofile/samples/current/中找到）记录样本数据。为了使用单独的功能，已更改了这些示例文件的名称和路径，以反映示例的来源。可包括线程 ID、二进制文件路径、使用的事件类型等等。

在完成从中断到磁盘文件的最后一步之后，数据现在是永久性的（也就是说，系统运行的变更不会使存储的数据失效）。这样后分析工具可随时使用该数据（假设原始二进制文件仍然可用且未更改）。

2.6.1.8 后分析工具

收集的数据必须以有用的形式呈现给用户。这是后分析工具的工作。通常，它们会整理可用样本文件的子集，加载并处理每个与相关二进制文件有关的文件，并生成用户可读信息。

2.6.1.9 中断要求

与 OProfile 驱动有关的中断数量很多。没有延迟要求。中断的生成速率取决于事件。

2.6.2 软件操作

对于 Cortex-A7 i.MX，Oprofile 需要添加 Cortex-A7 事件监视器

2.6.2.1 源代码结构

Oprofile 平台特定的源文件位于 arch/arm/oprofile 中。

表 17. OProfile 源文件

文件	说明
common.c	包含所有平台所需实施的源文件

Oprofile 的通用内核驱动位于 drivers/oprofile 下

2.6.2.2 菜单配置选项

为本模块提供了以下 Linux 内核配置。

在菜单配置中启用以下模块：

- CONFIG_OPROFILE-configuration option for the oprofile driver. In the menuconfig this option is available under
- General Setup > Profiling support (EXPERIMENTAL) > OProfile system profiling (EXPERIMENTAL)

2.6.2.3 编程接口

该驱动实施了配置和控制 PMU 和 L2 缓存 EVTMON 计数器所需的所有方法。

如需了解更多信息，请参见从 build:make htmldocs 生成的 Linux 文档。

2.6.2.4 示例软件配置

以下步骤和示例展示了如何配置 OProfile：

1. 使用命令 bitbake linux-imx -c menuconfig。在屏幕上，首先，转到包（Package）列表并选择 Oprofile。
2. 然后，返回第一个窗口并选择 Configure Kernel（配置内核），按照 Menu Configuration Options（菜单配置选项）中的说明，在内核空间中启用 Oprofile。
3. 保存配置并开始构建。
4. 将 Oprofile 二进制文件复制到目标 rootfs。将 vmlinux 复制到/boot 目录并运行 Oprofile

```
root@ubuntu:/boot# opcontrol --separate=kernel --vmlinux=/boot/vmlinux
root@ubuntu:/boot# opcontrol --reset
Signalling daemon... done
root@ubuntu:/boot# opcontrol --setup --event=CPU_CYCLES:100000
root@ubuntu:/boot# opcontrol --start
Profiler running.
```

```
root@ubuntu:/boot# opcontrol --dump
root@ubuntu:/boot# opreport
Overflow stats not available
CPU: ARM V7 PMNC, speed 0 MHz (estimated)
Counted CPU_CYCLES events (Number of CPU cycles) with a unit mask of 0x00 (No un
it mask) count 100000
CPU_CYCLES:100000|
samples|      %|
------------------
      4 22.2222 grep
      CPU_CYCLES:100000|
        samples|      %|
      ------------------
              4 100.000 libc-2.9.so
      2 11.1111 cat
      CPU_CYCLES:100000|
        samples|      %|
      ------------------
              1 50.0000 ld-2.9.so
              1 50.0000 libc-2.9.so
...
root@ubuntu:/boot# opcontrol --stop
Stopping profiling.
```

2.7 脉宽调制器（PWM）

2.7.1 介绍

脉宽调制器（PWM）有一个 16 位计数器，经过了优化，可根据存储的样本音频图像生成声音并生成音调。PWM 还提供背光控制。

PWM 具有 16 位分辨率，使用 4x16 数据 FIFO 产生声音。该软件模块由 Linux 驱动组成，允许特权用户通过 PWM 输出（PWMO）信号的适当占空比来控制背光。

2.7.2 硬件操作

下图展示了 PWM 框图。

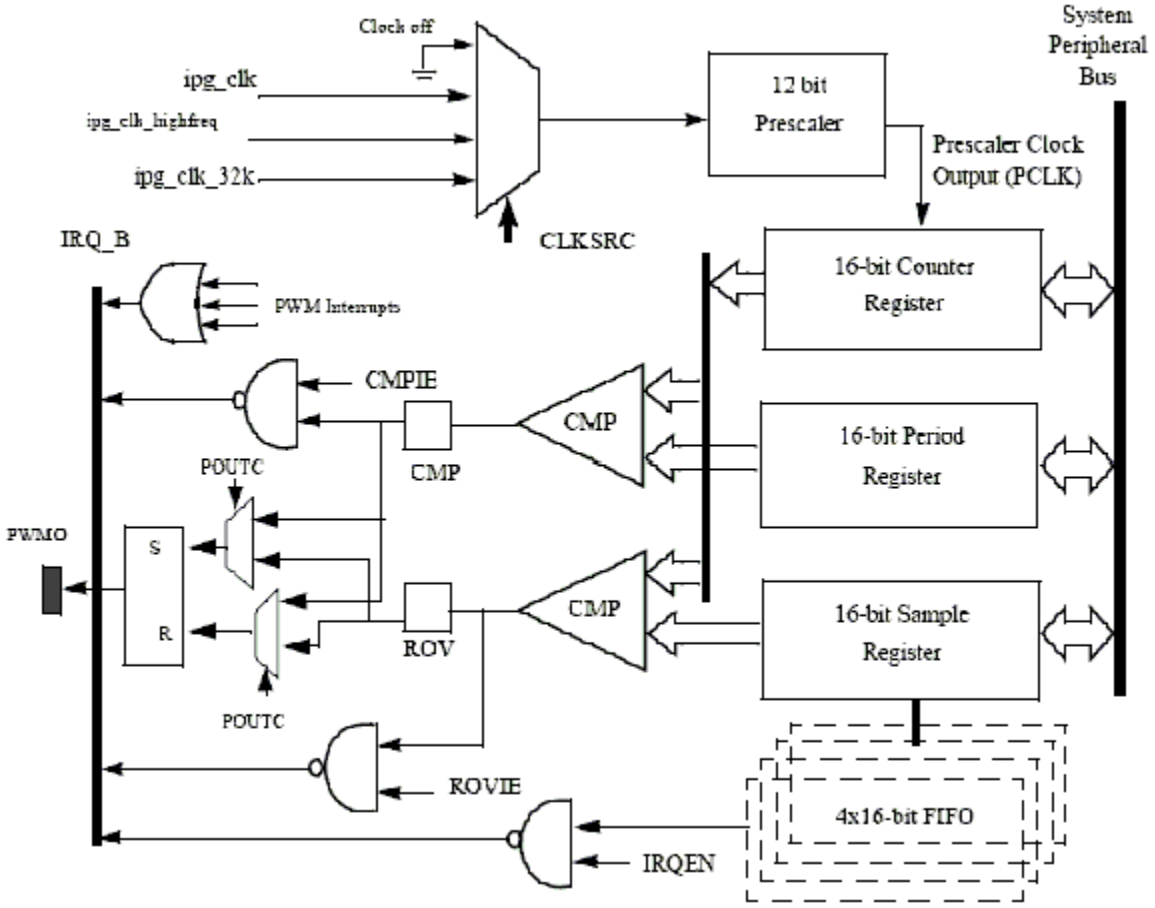


图 2. PWM 框图

PWM 遵循 IP 总线协议与处理器核心连接。除了来自时钟控制模块（CCM）的时钟和复位输入以及发送给处理器中断处理程序的中断信号外，它不与器件内的任何其他模块对接。PWM 包含单个外部输出信号 PMWO。PWM 包含以下内部信号：

- 3 个时钟输入
- 4 个中断线
- 一根硬件复位线
- 4 个低功耗和调试模式信号
- 4 个扫描信号
- 标准 IP 从总线信号

2.7.3 时钟

可以从以下选项中选择为预分频器馈送的时钟：

- CCM 提供的高频时钟。PWM 可在该时钟上以低功耗模式运行。
- 低参考时钟——CCM 提供的 32 KHz 低参考时钟。PWM 可在该时钟上以低功耗模式运行。
- 全局功能时钟——用于普通操作。在低功耗模式下，可关闭该时钟。

时钟输入源由 PWM 控制寄存器的 CLKSRC 字段决定。只有在禁用 PWM 时才能更改 CLKSRC 值。

2.7.4 软件操作

PWM 设备驱动通过改变发送到电源的一系列脉冲的宽度来减少发送到负载的电量。PWM 的一个常见且有效的用途是通过可变占空比控制 QVGA 面板的背光。

下表总结了源代码中的接口函数。

表 18. PWM 驱动综述

函数	说明
struct pwm_device *pwm_get(struct device *dev, const char *con_id)	查找并请求 PWM 设备
void pwm_put(struct pwm_device *pwm)	释放 PWM 设备
int pwm_config(struct pwm_device *pwm, int duty_ns, int period_ns)	更改 PWM 设备配置
int pwm_enable(struct pwm_device *pwm)	启动 PWM 输出切换
int pwm_disable(struct pwm_device *pwm)	停止 PWM 输出切换

函数 pwm_config()包含 PWM 模块的大多数配置任务，包括 PWM 输出信号的时钟源选项、周期和占空比。建议选择 PWM 模块的外设时钟，而不是本地功能时钟，因为本地功能时钟可能会发生变更。

2.7.5 驱动特性

PWM 驱动支持以下软件和硬件：

- 占空比调制
- 不同的输出间隔
- 两种电源管理模式——全开和全关

2.7.6 源代码结构

表 19. PWM 驱动文件

文件	说明
drivers/pwm/pwm.h	函数声明
drivers/pwm/pwm-imx.c	i.MX 脉宽调制函数

2.7.7 菜单配置选项

在菜单配置中启用以下模块：

- Device Drivers > Pulse-Width Modulation (PWM) Support > i.MX PWM support
- 选择以下选项，启用背光驱动：
  Device Drivers > Graphics support > Backlight & LCD device support > Generic PWM based Backlight Driver

2.8 远程处理器消息传递

2.8.1 介绍

通过采用 Arm Cortex®-A 系列处理器和 ArmCortex-M 系列处理器设计的最新多核架构，工业应用可提高能效，减少碳足迹。这样可以在不降低性能的情况下降低功耗。

传统上，同构 SoC 将运行控制所有存储器的单个操作系统（OS）。操作系统或管理程序将处理可用核心的任务管理，以最大限度地提高系统利用率。这种系统称为对称多处理（SMP）系统。

在异构多核芯片中，不同的处理核心运行不同的指令集和不同的操作系统。每个处理核心根据需要处理特定的任务。这种系统称为非对称多处理（AMP）系统。关于 SMP 和 AMP 系统之间的区别，同构多核 SoC 可以是 AMP 系统，但异质多核 SoC 不能作为 SMP 系统。

多核架构给系统设计带来了新的挑战，因为必须重写软件才能在可用核心之间分配任务。此外，需要合理分配所有外设资源，以避免资源争用现象，并实现核心之间数据空间的高效共享。多核 SoC 还需要一种机制，使其能够在不同处理核心上运行的任务之间进行可靠的通信和同步。

RPMsg 是一种基于 virtio 的消息总线，它允许内核驱动与系统上可用的远程处理器进行通信。反过来，如需要，驱动可以公开适当的用户空间接口。每个 RPMsg 设备都是与远程处理器进行通信的通道（因此 RPMsg 设备被称为通道）。通道采用文本名称标识，具有本地（“源”）RPMsg 地址和远程（“目的地”）RPMsg 地址。如需了解更多信息，请参见 www.kernel.org/doc/Documentation/rpmsg.txt。

如下图所示，消息通过双向无连接通信通道在端点之间传递。

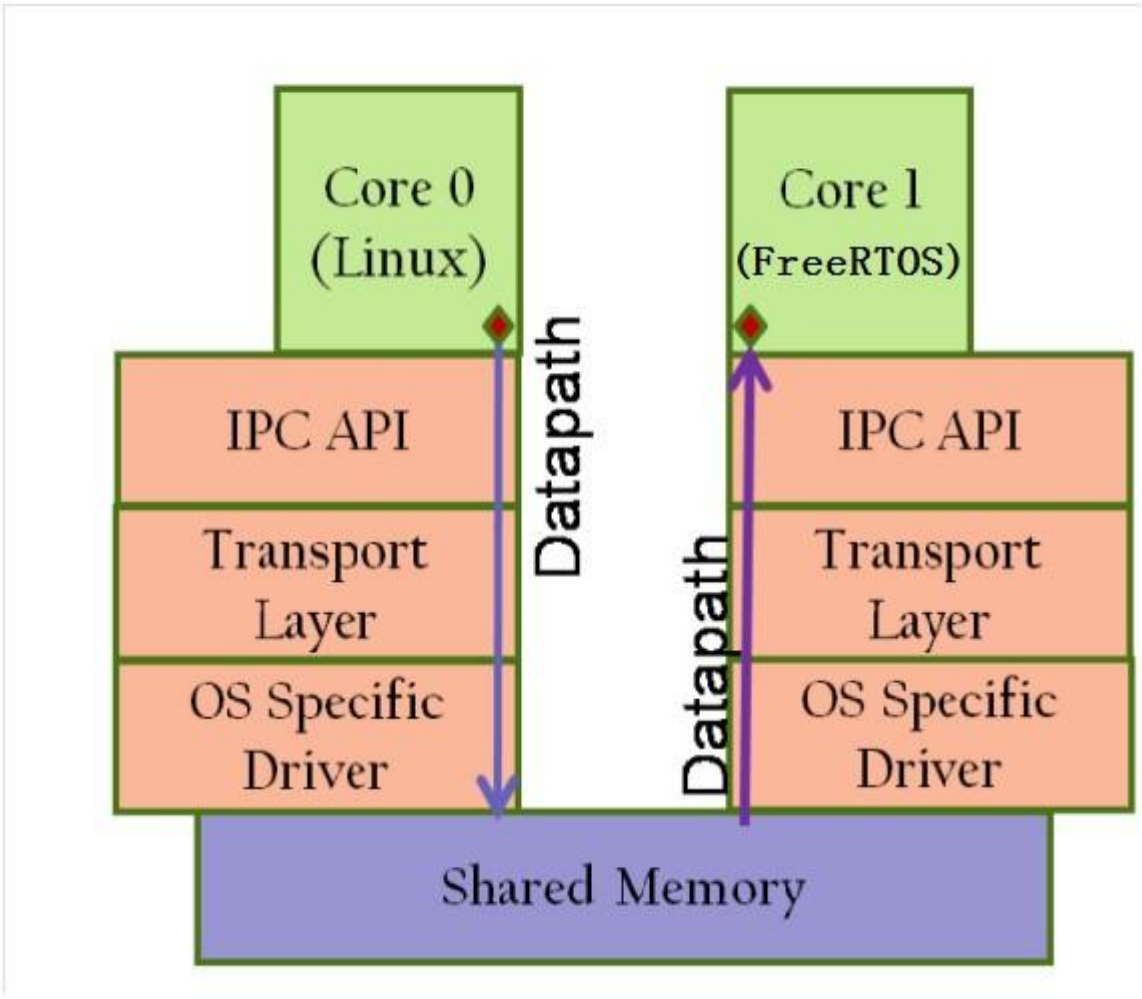


图 3. 新的多核、多操作系统架构

2.8.2 特性

- 专为低延迟和低开销操作而设计，并符合 Linux RPMsg 框架。
- 针对 CPU 和存储资源受限的嵌入式环境进行了优化。
- 使用共享存储实现，无需数据转换或消息头。
- 使用客户端-服务器方法进行应用通信。
- RPMsg 通道的动态分配。

2.8.3 源代码

RPMSG 驱动软件位于 drivers/rpmsg 中

表 20. RPMSG 源代码

文件	说明
drivers/rpmsg/virtio_rpmsg_bus.c	通用代码

下页继续……

表 20. RPMSG 源代码（续）

文件	说明
drivers/rpmsg/imx_rpmsg.c	i.MX 平台相关代码
drivers/rpmsg/imx_rpmsg_pingpong.c	i.MX RPMsg ping-pong 测试
drivers/rpmsg/imx_rpmsg_tty.c	i.MX RPMsg TTY 驱动

2.8.4 菜单配置选项

在菜单配置中启用以下模块：

- Device Drivers > IMX RPMSG pingpong driver——仅限可加载模块
- Device Drivers > IMX RPMSG tty driver——仅限可加载模块

2.8.5 运行 i.MX RPMsg 测试程序

要运行 i.MX RPMsg 测试程序，请执行以下操作：

1. 确保使用正确的 Cortex-M4 处理器 RTOS 和 Linux 镜像。

   例如在 i.MX 7Dual 平台上：

   - rpmsg_pingpong_sdk_7dsdb.bin -> ping-pong test used on the i.MX 7Dual SDB board
   - rpmsg_str_echo_sdk_7dsdb.bin -> tty string echo test used on the i.MX 7Dual SDB board
   - rpmsg_pingpong_sdk_7dval.bin -> ping-pong test used on the i.MX 7Dual 12x12 LPDDR3 Arm2 board
   - rpmsg_str_echo_sdk_7dval.bin -> tty string echo test used on the i.MX 7Dual 12x12 LPDDR3 Arm2 board

2. 加载 Cortex-M4 处理器 RTOS 镜像，并在 U-Boot 中启动。

   通过 TFTP 服务器或 U-Boot 中的可启动 SD 卡加载 Cortex-M4 处理器 RTOS 镜像。

   - 通过 TFTP 服务器加载 Cortex-M4 处理器 RTOS 镜像。例如，
     a. 启动到 U-Boot 并停止。
     b. 对 TFTP 使用以下命令，以响应 Cortex-M4 处理器 RTOS 镜像并启动它。

     ```
     dhcp 0x7e0000 10.192.242.53:rpmsg_pingpong_sdk_7dval.bin;  bootaux 0x7e0000
     ```

   - 通过 SD 卡加载 Cortex-M4 处理器 RTOS 镜像。例如，
     a. 采用 MFGtools 创建了一个可启动 SD 卡。然后，将 Cortex-M4 处理器 RTOS 文件复制到 VFAT 文件系统格式化的第一个分区。
     b. 更改 U-Boot 的默认 Cortex-M4 处理器 RTOS 名称。

     ```
     setenv m4image '<The name of the M4/RTOS image>';save
     ```

     c. 设置 Cortex-M4 处理器使用的启动参数。

     ```
     setenv run_m4_tcm 'if run loadm4image; then cp.b ${loadaddr} 0x7e0000 0x8000; bootaux
     0x7e0000; fi'; save
     ```

     d. 添加 run run_m4_tcm，修改原来的 bootcmd。

     ```
     setenv bootcmd "run run_m4_tcm; <original contents of the bootcmd>"; save
     ```

注意

当 Cortex-M4 处理器 RTOS 镜像运行时，i.MX 6SoloX 强制要求“uart_from_osc”。因此，U-Boot 的 mmcargs 应该在 i.MX 6SoloX 上进行修改。

```
setenv mmcargs 'setenv bootargs console=${console},${baudrate} root=$
{mmcroot}, uart_from_osc';save
```

3. 运行 RPMsg 测试程序。
   a. 确保 `imx_rpmsg_pingpong.ko` 和 `imx_rpmsg_tty.ko` 都已构建完成。
   b. 使用 `insmod imx_rpmsg_pingpong.ko` 或 `insmod imx_rpmsg_tty.ko` 运行测试程序。

注意

请勿同时运行不同的测试程序。

   c. 运行以下命令，并确保在启动 RPMsg TTY 测试时，RPMsg TTY 接收程序在后端运行。

```
/unit_tests/mxc_mcc_tty_test.out /dev/ttyRPMSG30 115200 R 100 1000 &
```

Linux 操作系统侧的日志：

```
insmod imx_rpmsg_tty.ko
imx_rpmsg_tty rpmsg0: new channel: 0x400 -> 0x1!
Install rpmsg tty driver!
echo deadbeaf > /dev/ttyRPMSG30
imx_rpmsg_tty rpmsg0: msg(<- src 0x1) deadbeaf len 8
```

2.9 散热

2.9.1 介绍

热驱动是监测和保护 SoC 的必要驱动。热驱动通过内部热传感器以特定频率监测 SoC 的温度。

它定义了两个跳变点：临界点（critical）和被动点（passive）。散热装置将根据 SoC 达到的不同跳变点采取措施保护 SoC：

- 当达到临界点时，散热装置将关闭系统。
- 当达到被动点时，散热装置会降低 CPU 频率，并通知 GPU/VPU 以较低频率运行。
- 当温度降至被动点以下 10°C 时，散热装置将释放所有散热动作。

热驱动由两个部分组成：

- 热区定义跳变点并监测 SoC 的温度。
- 散热装置根据不同的跳变点采取行动。

临界点和被动点阈值在以下文件中进行配置。

- i.MX 6 和 i.MX 7 SoC 在 drivers/thermal/imx_thermal.c 中进行配置
- i.MX 8M SoC 在其 dtsi 文件中对此进行配置，并在 defconfig 中指定 CONFIG_IMX8M_THERMAL。
- i.MX 8 和 i.MX 8X SoC 在其 dtsi 文件中对此进行配置，并在 defconfig 中指定 CONFIG_IMX_SC_THERMAL。

2.9.2 软件操作

热驱动注册一个热区和一个散热装置。thermal_zone_device_ops 结构描述了散热框架所需的必要接口。该框架将调用相关的热区接口来监测 SoC 温度并进行散热保护。

可通过以下接口访问热驱动：

- /sys/bus/platform/drivers/imx_thermal 用于 i.MX 6 和 i.MX 7。
- /sys/class/thermal/thermal_zoneX 适用于 i.MX 8 和 i.MX 8X。
- /sys/bus/platform/drivers/qoriq_thermal 用于 i.MX 8M Quad。
- /sys/class/thermal/thermal_zone0/temp 用于 i.MX 8M Mini。

2.9.3 源代码结构

下表显示了 drivers/thermal 中提供的驱动源文件：

表 21. 热驱动文件

文件	说明
imx_thermal.c, device_cooling.c	i.MX 6 或 i.MX 7 的热区驱动源文件。
qoriq_thermal.c, device_cooling.c	i.MX 8M 的热区驱动源文件。
imx_sc_thermal.c, device_cooling.c	i.MX 8 和 i.MX 8X 的热区驱动源文件。

2.9.4 菜单配置选项

在菜单配置中启用以下模块：

- 对于 i.MX6 和 i.MX7：Device Drivers > Generic Thermal sysfs driver > Temperature sensor driver for i.MX SoCs。
- 对于 i.MX 8QuadMax 和 i.MX 8QuadXPlus：Device Drivers > Generic Thermal sysfs driver > thermal sensor driver for NXP i.MX8 SoCs

2.10 传感器

2.10.1 介绍

传感器包括一组用于加速度传感器、压力、陀螺仪、环境光和磁力计的驱动。

传感器在每个电路板的设备树中进行配置。

i.MX 对以下 SoC 支持加速度传感器：

- i.MX 6SABRE-SD、6SABRE-AI 和 6SoloX 使用 MMA8451 传感器
- i.MX 6UltraLite 和 6ULL EVK 使用 FXLS8571Q 传感器。
- i.MX 7Dual SABRE-SD、 i.MX 8QuadMax 和 i.MX 8QuadXPlus 使用 FX0S8700 传感器。

i.MX 对以下 SoC 支持压力传感器 MPL3115：

- i.MX 7 Dual SABRE-SD
- i.MX 8 QuadMax
- i.MX 8 QuadXPlus

i.MX 对以下 SoC 支持陀螺仪传感器 FXAS2100：

- i.MX 7 Dual SABRE-SD

i.MX 对以下 SoC 支持环境光传感器 ISL29023：

- i.MX 6 SABRE-SD 和 6 SABRE-AI
- i.MX 6 SoloX
- i.MX 8 QuadMax
- i.MX 8 QuadXPlus。

i.MX 对以下 SoC 支持磁力计传感器 MAG3110：

- i.MX 6 SABRE-SD
- i.MX 6 UL EVK
- i.MX 6 ULL EVK
- i.MX 6SoloX

2.10.2 传感器驱动软件操作

2.10.3 源代码结构

下表展示了目录中提供的驱动源文件：

表 22. 传感器驱动文件

文件	说明
drivers/hwmon/mxc_mma8451.c	加速计传感器
drivers/misc/fxos8700.c	加速计和磁力计传感器
drivers/misc/fxls8471.c	加速计和磁力计光传感器
drivers/input/misc/isl29023.c	环境光传感器
drivers/input/misc/fxas2100x.c	磁力计光传感器
drivers/hwmon/mag3110.c	磁力计传感器

2.10.4 菜单配置选项

在菜单配置中启用以下模块：

- Device Drivers > Misc devices > Intersil ISL29020 ambient light sensor
- Device Drivers > Misc devices > Freescale FXOS8700 M+G combo sensor
- Device Drivers > Misc devices > Freescale FXAS2100X gyroscope sensor
- Device Drivers > Hardware Monitoring support > Freescale MAG3110 e-compass sensor
- Device Drivers > Hardware Monitoring support > MMA8451 device driver

2.11 看门狗（WDOG）

2.11.1 介绍

看门狗定时器模块可从意外挂起或无限循环情况或编程错误中逃逸，以防止系统故障。

某些平台可能有两个看门狗模块，其中一个具有中断功能。i.MX 6 和 7Dual 与 i.MX 8M 共享同一个看门狗驱动。i.MX 7ULP 有一个单独的看门狗驱动。i.MX 8 和 i.MX 8X 通过系统控制器固件共享一个虚拟看门狗驱动接口。

2.11.2 硬件操作

看门狗定时器激活后，必须定期通过软件对其进行维护。

如未及时进行维护，看门狗将超时。超时后，看门狗会根据软件配置发出 wdog_b 信号或 wdog_rst_b 系统重置信号。看门狗模块激活后无法停用。

2.11.3 软件操作

Linux 操作系统有一个标准的看门狗接口，支持特定平台的看门狗驱动。

看门狗可以在 STOP/DOZE 和 WAIT 模式下单独暂停/恢复。由于 WDOG 寄存器的某些位在启动后只能一次性编程，请确保正确写入这些寄存器。

2.11.4 通用看门狗

通用看门狗驱动在 drivers/watchdog/imx2_wdt.c 文件中实施。

它为各种 IOCTL 和来自用户级程序的读/写调用提供函数，以控制看门狗。

2.11.5 驱动特性

此看门狗实施包括以下功能：

- 如果看门狗已启用，但未在预定义的超时值（在一个看门狗源文件中以毫秒为单位定义）内为其提供维护，则产生重置信号
- 如果看门狗在预定义的超时值内得到维护，则不会产生重置信号
- 提供标准看门狗子系统所需的 IOCTL/读/写

2.11.6 源代码结构

下表列出了 drivers/watchdog 中 watchdog WDOG 驱动的源文件。

表 23. 看门狗驱动文件

文件	说明
driveers/watchdog/imx2_wdt.c	i.MX 6、i.MX 7Dual 和 i.MX 8M 看门狗函数实施。对于 i.MX 6 和 i.MX 7，看门狗系统复位函数位于 arch/arm/mach-imx/system.c 下。
drivers/watchdog/imx7ulp_wdt.c	i.MX 7ULP 看门狗函数实施
drivers/watchdog/imx8_wdt.c	在 i.MX 8 和 i.MX 8X 上，系统控制器固件（SCFW）中使用的软件看门狗和内核通过虚拟看门狗驱动 imx8_wdt.c 调用这些接口。这不适用于 i.MX 8M。

2.11.7 菜单配置选项

在菜单配置中启用以下模块：

Device Drivers > Watchdog Timer Support > IMX2+ Watchdog

Device Drivers > Watchdog Timer Support > IMX7ULP Watchdog

Device Drivers > Watchdog Timer Support > IMX8 Watchdog

2.11.8 编程接口

看门狗驱动支持以下 IOCTL：

- WDIOC_GETSUPPORT
- WDIOC_GETSTATUS
- WDIOC_GETBOOTSTATUS
- WDIOC_KEEPALIVE
- WDIOC_SETTIMEOUT
- WDIOC_GETTIMEOUT

如需了解有关这些 IOCTL 的详细说明，请参见 `Documentation/watchdog`。

第 3 章
存储

3.1 带 DMA 的 AHB 到 APBH 桥接器（APBH-Bridge-DMA）

3.1.1 概述

AHB 到 APBH 桥接器为处理器提供了一条在 AHB 的 HCLK 上运行的价廉的外设连接总线。APBH 中的 H 表示 APBH 与 HCLK 同步。

AHB 到 APBH 桥接器包括 AHB-to-APB PIO 桥接器，用于将存储器映射到 APB 设备的 I/O，以及用于该总线上设备的中央 DMA 设备和用于 Arm 核心的矢量中断控制器。包括矢量中断控制器在内的每个 APB 外设都在本文档的其他章节中进行了说明。

这些设备没有单独的 DMA 总线。通过内部仲裁逻辑调解 DMA 使用 APBH 总线和 APBH 使用 AHB 到 APB 桥接功能之间的争用。对于这两个单元之间的争用，优先使用 DMA，AHB 从机将通过其 HREADY 输出报告“未就绪”，直到桥接传输完成。仲裁器跟踪重复的锁定并反转优先级，保证 Arm 平台每四次在 APB 上传输一次。

3.1.1.1 硬件操作

SDMA 控制器负责在 MCU 存储器空间和外设之间传输数据，包含以下功能。

- 支持多达 32 个时分复用 DMA 通道的多通道 DMA
- 由 16 位指令集 micro-RISC 引擎驱动
- 每个通道执行一个特定的脚本
- 非常快速的上下文切换，基于两级优先级的抢先式多任务处理
- 4 KB ROM，包含启动脚本（即启动代码）和其他可由 RAM 中的脚本参考的常用实用程序
- 8 KB RAM 区域分为处理器上下文区域和代码空间区域，用于存储从系统存储器下载的通道脚本。

3.1.1.2 软件操作

DMA 支持 16 个 DMA 服务通道，如下表所示。共享的 DMA 资源允许每个独立通道遵循简单的链式命令列表。命令链是使用通用结构构建的。

表 24. APBH DMA 通道分配

APBH DMA 通道编号	用途
0	GPMI0
1	GPMI1
2	GPMI2
3	GPMI3
4	GPMI4
5	GPMI5
6	GPMI6

下页继续……

表 24. APBH DMA 通道分配（续）

APBH DMA 通道编号	用途
7	GPMI7
8	空
9	空
10	空
11	空
12	空
13	空
14	空
15	空

3.1.1.3 源代码结构

下表列出了 drivers/dma/中提供的源文件

表 25. APBH DMA 源文件

文件	说明
mxs-dma.c	APBH DMA 实施驱动

3.1.1.4 菜单配置选项

在菜单配置中启用以下模块：

- Device Drivers > DMA Engine support > MXS DMA support.

3.1.1.5 编程接口

该模块实现标准的 DMA API。如需了解 GPMI NAND 驱动等驱动中实施的函数的更多信息，请参见 Linux 文档包中的 API 文档。

3.2 EIM NOR

3.2.1 介绍

外部接口模块（EIM）NOR 驱动支持并行 NOR 闪存。

3.2.2 硬件操作

默认情况下，i.MX 6Quad/6Dual SABRE-AI 电路板中有一个并行 NOR。并行 NOR 的引脚比 SPI NOR 的引脚多。某些电路板上配备了 M29W256GL7AN6E。如需了解并行 NOR 的详细信息，请参见数据手册。

3.2.3 软件操作

与 SPI NOR 类似，并行 NOR 使用 MTD 子系统。因为并行 NOR 非常小，只能使用 jffs2 而不能使用 UBIFS。

3.2.4 源代码

表 26. WEIM-NOR 驱动文件

文件	说明
drivers/bus/imx-weim.c	仅对并行 NOR WEIM-NOR 源更改时序

3.2.5 启用 EIM NOR

如需了解如何启用 EIM NOR，请参见 DTS 文件：imx6q-sabreauto-gpmi-weim.dts 或 imx6dl-sabreauto-gpmi-weim.dts。

3.3 MMC/SD/SDIO 主机

3.3.1 介绍

多媒体卡（MMC）/安全数字（SD）/安全数字输入输出（SDIO）主驱动实施了与超 MMC/SD 主控制器（uSDHC）对接的标准 Linux 驱动接口。

主驱动是 Linux 内核 MMC 框架的一部分。

MMC 驱动具有以下功能：

- SD3.0 和 SDIO 2.0 卡的 1 位或 4 位操作（目前我们支持 SDIO v2.0（AR6003 已验证））。
- 支持插卡和拔卡检测。
- 支持标准的 MMC 命令。
- PIO 和 DMA 数据传输。
- 支持电源管理。
- 支持 MMC 卡的 1/4 8 位操作。
- 对于 i.MX 6，USDHC 支持 eMMC4.4 SDR 和 DDR 模式。
- 对于 i.MX 7Dual，USDHC 支持 eMMC5.0，包括 HS400 和 HS200。
- 支持 SD3.0 SDR50 和 SDR104 模式。

3.3.2 硬件操作

MMC 通信基于先进的 11 引脚串行总线，在低电压范围内工作。uSDHC 模块支持 MMC 以及 SD 存储器和 I/O 功能。uSDHC 通过向卡发送命令并执行与卡之间的数据访问来控制 MMC、SD 存储器和 I/O 卡。SD 存储卡系统定义了两种可选通信协议：SD 和 SPI。uSDHC 只支持 SD 总线协议。

uSDHC 命令传输类型和 uSDHC 命令参数寄存器允许向卡下发命令。uSDHC 命令、系统控制和协议控制寄存器允许用户指定数据和响应的格式，并控制读取等待周期。

uSDHC 中有 4 个 32 位寄存器用于存储来自卡的响应。uSDHC 读取这 4 个寄存器以直接获得命令响应。uSDHC 使用完全可配置的 128x32 位 FIFO 进行读写。缓冲区用作在主系统和卡之间传输数据的临时存储器，反之亦然。uSDHC 数据缓冲区访问寄存器位在读或写传输时保存 32 位数据。

接收数据的步骤如下所示：

1. 当缓冲区中接收到的字数超过 RD_WML 寄存器中设置的数量时，uSDHC 控制器会生成 DMA 请求
2. 接收到该请求后，DMA 引擎通过读取数据缓冲区访问寄存器开始将数据从 uSDHC FIFO 传输到系统存储器。

发送数据的步骤如下所示：

1. 每当缓冲区空间量超过 WR_WML 寄存器中设置的值时，uSDHC 控制器都会生成一个 DMA 请求。
2. 接收到该请求后，DMA 引擎向数据缓冲区访问寄存器写入多个预定义字节，将数据从系统存储器移动到 uSDHC FIFO。

只读 uSDHC 当前状态和中断状态寄存器提供 uSDHC 操作状态、应用 FIFO 状态、错误情况和中断状态。

当发生某些事件时，该模块能够产生中断并设置相应的状态寄存器位。uSDHC 中断状态使能和信号使能寄存器允许用户控制是否会发生这些中断。

3.3.3 软件操作

Linux 操作系统包含一个实施 MMC 总线协议的 MMC 总线驱动。MMC 块驱动处理文件系统读/写调用，并使用低级 MMC 主控制器接口驱动向 uSDHC 发送命令。

MMC 驱动负责实施 `init`、`exit`、`request` 和 `set_ios` 的标准入口点。该驱动实施以下函数：

对于 uSDHC：

- `init` 函数 `esdhc_pltfm_init()` 初始化平台硬件并将平台相关标志或值设置为 sdhci_host 结构。
- `exit` 函数 `esdhc_pltfm_exit()` 取消平台硬件初始化，并释放分配的存储器。
- `esdhc_pltfm_get_max_clock()` 函数获取平台支持的最大 SD 总线时钟频率。
- `esdhc_pltfm_get_min_clock()` 函数获取平台支持的最小 SD 总线时钟频率。
- `esdhc_pltfm_get_ro()` 获取卡只读状态。
- `esdhc_execute_tuning()` 处理调整的准备工作。它只用于 SD3.0 UHS-I 模式。
- `esdhc_set_clock()` 处理时钟更改请求。

下图展示了 MMC 相关驱动是如何分层的。

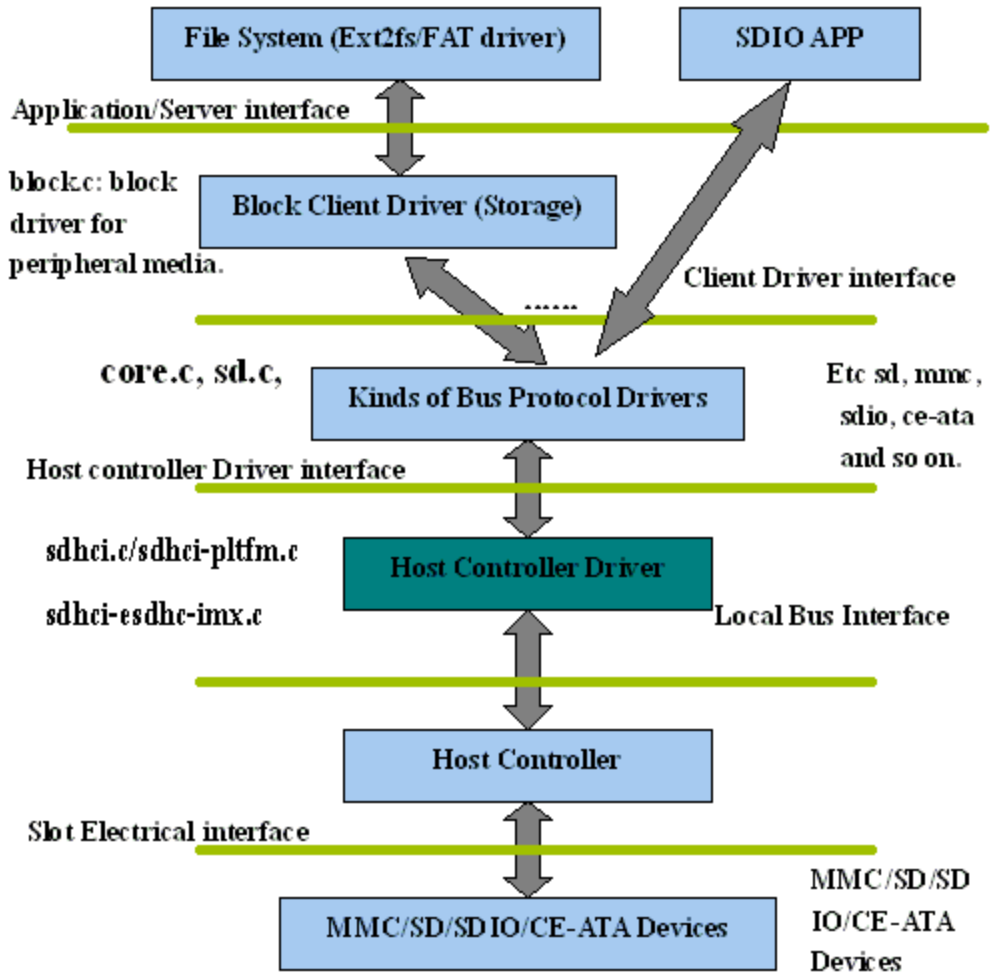


图 4. MMC 驱动分层

3.3.4 驱动特性

MMC 驱动支持以下功能：

- 支持多个 uSDHC 模块。
- 提供与 Linux MMC 核心驱动对接的所有入口点。
- MMC 和 SD 卡。
- SDIO 卡。
- SD3.0 卡。
- 识别数据传输错误，例如命令超时和 CRC 错误。
- 电源管理。
- 支持构建为可加载模块或内置模块

3.3.5 源代码结构

下表展示了 drivers/mmc/host 中提供的 uSDHC 源文件。

表 27. uSDHC 驱动文件 MMC/SD 驱动文件

文件	说明
drivers/mmc/host/sdhci.c	sdhci 标准堆栈代码
driers/mmc/host/sdhci-pltfm.c	sdhci 平台层
drivers/mmc/host/sdhci-esdhc-imx.c	uSDHC 驱动
drivers/mmc/host/sdhci-esdhc.h	uSDHC 驱动头文件

3.3.6 菜单配置选项

为该模块提供了以下 Linux 内核配置选项。

- CONFIG_MMC 为 MMC 总线协议提供支持。在 menuconfig 中，此选项可在以下位置找到：
  - Device Drivers > MMC/SD/SDIO Card support
  - 默认情况下，此选项为 Y。
- CONFIG_MMC_BLOCK 为可用于安装文件系统的 MMC 块设备驱动提供支持。在 menuconfig 中，此选项可在以下位置找到：
  - Device Drivers > MMC/SD Card Support > MMC block device driver
  - 默认情况下，此选项为 Y。
- CONFIG_MMC_SDHCI_ESDHC_IMX 用于 i.MX USDHC 端口。在 menuconfig 中，此选项位于：
  - Device Drivers > MMC/SD Card Support > Secure Digital Host Controller Interface support > SDHCI support on the platform-specific bus > SDHCI platform support for the eSDHC i.MX controller

  要将 SDHCI 驱动编译为可加载模块，应选择如下所示的几个选项：
  - CONFIG_MMC_SDHCI=m，可以在“Device Drivers > MMC/SD Card Support > Secure Digital Host Controller Interface support”下找到
  - CONFIG_MMC_SDHCI_PLTFM=m，可在“Device Drivers > MMC/SD Card Support > Secure Digital Host Controller Interface support > SDHCI support on the platform-specific bus”中找到。
  - CONFIG_MMC_SDHCI_ESDHC_IMX=y，可在“Device Drivers > MMC/SD Card Support > Secure Digital Host Controller Interface support > SDHCI support on the platform-specific bus > SDHCI platform support for the Freescale eSDHC i.MX controller”下找到

  要将 SDHCI 驱动编译为内置模块，应选择如下所示的几个选项：
  - CONFIG_MMC_SDHCI=y，可在“Device Drivers > MMC/SD Card Support > Secure Digital Host Controller Interface support”下找到
  - CONFIG_MMC_SDHCI_PLTFM=y，可在“Device Drivers > MMC/SD Card Support > Secure Digital Host Controller Interface support > SDHCI support on the platform-specific bus”下找到。
  - CONFIG_MMC_SDHCI_ESDHC_IMX=y，可在“Device Drivers > MMC/SD Card Support > Secure Digital Host Controller Interface support > SDHCI support on the platform-specific bus > SDHCI platform support for the Freescale eSDHC i.MX controller”下找到
- CONFIG_MMC_UNSAFE_RESUME 用于将 MMC/SD/SDIO 卡用于 rootfs 的嵌入式系统。在 menuconfig 中，此选项位于：

3.3.7 设备树绑定

必选属性：

- compatible：应为“fsl,<chip>-esdhc”

- reg：应包含 eSDHC 寄存器位置
- interrupts：应包含 eSDHC 中断

可选属性：

- non-removable：表示该卡已永久连接到主机
- fsl,cd-internal：表示使用控制器内部卡检测
- fsl,wp-internal：表示使用控制器内部写保护
- cd-gpios：指定用于卡检测的 GPIO
- wp-gpios：指定用于写保护的 GPIO
- fsl,delay-line：指定 eMMC DDR 模式的延迟线值

```
Example:usdhc@02194000 { /* uSDHC2 */
compatible = "fsl,imx6q-usdhc";
reg = <0x02194000 0x4000>;
interrupts = <0 23 0x04>;
clocks = <&clks 164>, <&clks 164>, <&clks 164>; clock-
names = "ipg", "ahb", "per";
pinctrl-names = "default";
pinctrl-0 = <&pinctrl_usdhc2_1>;
cd-gpios = <&gpio2 2 0>;
wp-gpios = <&gpio2 3 0>;
bus-width = <8>;
no-1-8-v;
keep-power-in-suspend;
enable-sdio-wakeup;
status = "okay";
};
```

参考文献：

- Documentation/devicetree/bindings/mmc/fsl-imx-esdhc.txt
- arch/arm/boot/dts/imx6*.dtsi

3.3.8 编程接口

该驱动实施 MMC 总线协议与 i.MX uSDH 模块对接所需的函数。请参见从 build: make htmldocs 生成的 Linux 文档。

3.3.9 可加载模块操作

SDHCI 驱动可以构建为可加载模块或内置模块。

1. 如何将 SDHCI 驱动构建为可加载模块。
   - CONFIG_MMC_SDHCI=m，可在“Device Drivers > MMC/SD Card Support > Secure Digital Host Controller Interface support > SDHCI support on the platform-specific bus”下找到
   - CONFIG_MMC_SDHCI_PLTFM=m，可在“Device Drivers > MMC/SD Card Support > Secure Digital Host Controller Interface support > SDHCI support on the platform-specific bus”下找到。
   - CONFIG_MMC_SDHCI_ESDHC_IMX=y，可在“Device Drivers > MMC/SD Card Support > Secure Digital Host Controller Interface support > SDHCI support on the platform-specific bus > SDHCI platform support for the i.MX eSDHC i.MX controller”下找到
2. 如何加载和卸载 SDHCI 模块。

   由于依赖关系，请按照如下所示的顺序加载或卸载该模块。

运行以下命令加载模块：

- 假设当前目录下有 sdhci.ko 和 sdhci-platform.ko 文件，通过 insmod 命令加载模块。

```
$> insmod sdhci.ko
$> insmod sdhci-platform.ko
```

- 通过 modprobe 命令加载模块，确保对应的内核模块 lib 目录中有 sdhci.ko 和 sdhci-platform.ko 文件。

```
$> modprobe sdhci.ko
$> modprobe sdhci-platform.ko
```

运行以下命令卸载模块：

- 通过 insmod 命令卸载模块。

```
$> rmsmod sdhci-platform
$> rmsmod sdhci
```

- 通过 modprobe 命令卸载模块。

```
$> modprobe -r sdhci-platform
$> modprobe -r sdhci
```

3.4 NAND GPMI 闪存

3.4.1 介绍

NAND 闪存技术设备（MTD）驱动用于 i.MX 6 系列和 i.MX 7Dual 上的通用媒体接口（GPMI）控制器。

NAND MTD 驱动只需实施硬件特定层即可运行。

闪存读取/写入/擦除等其余功能由 Linux MTD 子系统为 NAND 设备提供的通用层自动处理。

NAND MTD 驱动与支持文件系统的集成 NAND 控制器对接，例如 UBIFS、CRAMFS 和 JFFS2UBI 以及 UBIFSCRAMFS 和 JFFS2。该驱动实施支持外部 NAND 闪存芯片上的最低级别操作，例如块读取、块写入和块擦除，因为 NAND 闪存技术仅支持块访问。由于 NAND 闪存中的块不能保证是好的，NAND MTD 驱动还能够检测坏块，并将该信息提供给上层以进行坏块管理。

3.4.2 硬件操作

NAND 闪存是一种用于嵌入式系统的非易失性存储设备。

驱动不支持存储器的随机存取，如 RAM 或 NOR 闪存。必须通过 GPMI 才能执行读取或写入 NAND 闪存操作。NAND 闪存是一种适用于大容量存储应用的顺序存取设备。存储在 NAND 闪存上的代码不能在闪存里执行。必须将代码加载到 RAM 存储器中并在该存储器执行。i.MX 6 包含一个硬件纠错块。

3.4.3 软件操作

Linux 中的 MTD 涵盖所有存储器设备，如 RAM、ROM 和各种 NOR/NAND 闪存。

MTD 子系统提供对所有此类设备的统一访问。在 MTD 设备之上，可以使用闪存文件系统（JFFS2）或 UBI 层进行 MTD 块设备仿真。反过来，UBI 层可以在卷的上方有 UBIF，也可以在其上方有一个具有常规文件系统（FAT，Ext2/3）的闪存转换层（FTL）。硬件特定的驱动与 i.MX 6 上的 GPMI 模块对接。它实施了读、写和擦除等最低级别的操作。如启用，它还提供有关 NAND 设备上分区的信息。该信息必须由平台代码提供。

如果可能，可在 NAND 驱动上纠正读/写错误。硬件纠错由 BCH 块执行，并由 NAND 驱动代码驱动。

如需了解 NAND 驱动接口的详细信息，请访问 www.linux-mtd.infradead.org。

3.4.4 基本操作：读/写

NAND 驱动导出以下回调：

```
gpmi_ecc_read_page (with ECC)
gpmi_ecc_write_page (with ECC)
gpmi_read_byte (without ECC)
gpmi_read_buf (without ECC)
gpmi_write_buf (without ECC)
gpmi_ecc_read_oob (with ECC)
gpmi_ecc_write_oob (with ECC)
```

从 Kernel 4.1 开始，GPMI 驱动提供了原始读/写模式，可导出以下回调：

- gpmi_ecc_read_page_raw (without ECC)
- gpmi_ecc_write_page_raw (without ECC)
- gpmi_ecc_read_oob_raw (without ECC)
- gpmi_ecc_write_oob_raw (without ECC)

这些函数读取请求的数据量，无论是否进行纠错。在读取的情况下，调用 gpmi_read_page()函数，创建 DMA 链，提交执行，等待完成。写入操作情况更复杂：通过调用 gpmi_send_page()对要写入的数据进行映射和刷新。

3.4.5 向后兼容性

用户应知道 kernel 4.1 中的几个主要 GPMI NAND 驱动变更，这些变更可能会导致内核升级不兼容。

- 通过 debugfs 将必要的信息导出到用户空间应用(kobs-ng)
- 新的 BCH 布局算法
- 新的原始读/写模式

在 Kernel 4.1 中，NAND GPMI 驱动通过 debugfs 将必要的信息导出到上层。最常见的情况是使用 NAND 烧录工具 kobs-ng。如果不启用 debugfs，kobs-ng 可能无法完全使用新功能，或者可能使用不合适的参数。如果自定义内核中未启用 debugfs，用户需要向 kobs-ng 提供正确的 BCH 几何信息和原始访问模式。

之前内核中的 BCH 布局可能无法满足 NAND 芯片的最低 ECC 要求。从 Kernel 4.1 开始，默认情况下，BCH 布局算法使用 NAND 所需的 ECC 强度和步长，如果可以访问，可以从 ONFI 参数中获取。此更改可能与之前内核中的 BCH 布局设置不兼容。为了向后兼容，内核和 U-boot 提供了使用遗留 BCH 布局的开关。

- 对于 Kernel，在设备树文件中添加“fsl,legacy-bch-geometry”。
- 对于 U-Boot，在电路板配置文件中添加“CONFIG_NAND_MXS_BCH_LEGACY_GEO”。

BCH 遗留布局设置必须在 Kernel 和 U-boot 中同时打开/关闭，以保持统一。

Kobs-ng 检查 debugfs 中的内核版本或原始模式标志，以确定是否使用新的原始模式访问 NAND 芯片。新的 Kobs-ng 完全向后兼容以前的内核，而旧版本的 Kobs-ng 无法在 Kernel 4.1 上工作。

3.4.6 纠错

在向闪存读取或写入数据时，某些位可能会翻转。这是正常的现象，NAND 驱动利用各种纠错方案来纠正这一问题。可通过软件或硬件纠错来解决。GPMI 驱动在硬件加速器 BCH 的帮助下仅使用硬件纠错方案。

对于 BCH，2K 页面的页面布局为（2K+64），4K 页面的页面布局为（4K+218），8K 页面的页面布局为（8K+448）。

3.4.7 启动控制块管理

在启动过程中，NAND 驱动扫描第一个块，看是否存在 NAND 控制块（NCB）。通过 magic 签名检测是否存在该模块。找到签名后，使用汉明码检查启动块候选项是否存在错误。如发现错误，则尽可能对其进行修复。如果找到 NCB，将对其进行解析以检索 NAND 芯片的时序。

所有启动控制块都是在使用用户空间应用 kobs-ng 格式化介质时创建的。

3.4.8 坏块处理

默认情况下，当驱动启动时，会构建坏块表格。可以动态地确定一个块是否坏了，但为了提高性能，在启动时完成。检查块的每一页的空闲区域开头的模式，确定擦除块是好是坏。然而，如果芯片使用硬件纠错，则坏标记会落入 ECC 字节区域。因此，如果使用硬件纠错，则应删除坏块标记。

3.4.9 源代码结构

NAND 驱动位于 drivers/mtd/nand/gpmi-nand。

下表列出了 NAND 驱动的源文件。

表 28. NAND 驱动文件

文件	说明
• drivers/mtd/nand/gpmi-nand/bch-regs.h • drivers/mtd/nand/gpmi-nand/gpmi-nand.h • drivers/mtd/nand/gpmi-nand/gpmi-regs.h	函数声明
• drivers/mtd/nand/gpmi-nand/gpmi-lib.c • drivers/mtd/nand/gpmi/nand/gpmi-nand.c	GPMI NAND 功能

3.4.10 菜单配置选项

要启用 NAND 驱动，必须设置以下选项：

- Device Drivers > Memory Technology Device (MTD) support > GPMI NAND Flash Controller driver

此外，必须启用以下 MTD 选项：

- CONFIG_MTD_NAND = [y | m]
- CONFIG_MTD = y
- CONFIG_MTD_BLOCK = y

另外，必须启用这些 UBI 选项：

- CONFIG_MTD_UBI=y
- CONFIG_UBIFS_FS=y

3.5 四线串行外设接口（QuadSPI）

3.5.1 介绍

4 线串行外设接口（QuadSPI）块充当一个或两个外部串行闪存设备的接口，每个设备最多有四条双向数据线。

它支持以下功能：

- 灵活的序列引擎，支持各种闪存供应商设备。
- 单、双、4 和 8 线制操作模式。
- DDR/DTR 模式，其中数据在串行闪存时钟的每个边沿生成。
- 支持闪存数据选通信号，用于 DDR 和 SDR 模式下的数据采样。
- DMA 支持通过 AMBA AHB 总线（64 位宽度接口）或 IP 寄存器空间（32 位访问）读取 RX 缓冲区数据。

3.5.2 硬件操作

一些电路板上配备了 4 线 SPI NOR-N25Q256A，而其他一些电路板则配备了 S25FL128S。检查电路板上的 4 线 SPI NOR 类型，然后正确配置。

N25Q256A 是一款高性能多输入/输出串行闪存设备。创新的高性能双线和 4 线输入/输出指令使读取和编址操作的传输带宽增加了两倍或 4 倍。存储器由 512 个（64 KB）主扇区组成，一次可擦除 64 KB 的扇区。该设备具有 3 字节或 4 字节地址模式，可访问超过 128 MB 的存储器。启用 4 字节地址模式时，必须采用 4 字节地址模式命令（ENTER 4-BYTE ADDRESS MODE 命令和 EXIT 4-BYTE ADDRESS MODE 命令）输入和退出所有需要地址的命令。4 字节地址模式也可以通过非易失性配置寄存器启用。存储器可以使用 3 种不同的协议运行：扩展 SPI（标准 SPI 协议升级为双线和 4 线操作）、双线 I/O SPI 和 4 线 I/O SPI。每个协议都包含在 DTR 模式下执行读取操作的唯一命令。可在较低时钟频率下运行时实现较高的数据吞吐量。

S25FL128S 器件是闪存非易失性存储器产品。它通过串行外设接口（SPI）连接到主系统。支持传统的 SPI 单位串行输入和输出（单 I/O 或 SIO），以及可选的两位（双 I/O 或 DIO）和 4 位（4 线 I/O 或 QIO）串行命令。它还增加了对 SIO、DIO 和 QIO 的双数据速率（DDR）读取命令的支持，在时钟的两个边沿传输地址和读取数据。

3.5.3 软件操作

在基于 Flash 的嵌入式 Linux 系统中，许多 Linux 技术协同工作，实施一个文件系统。下图展示了一些标准组件之间的关系。

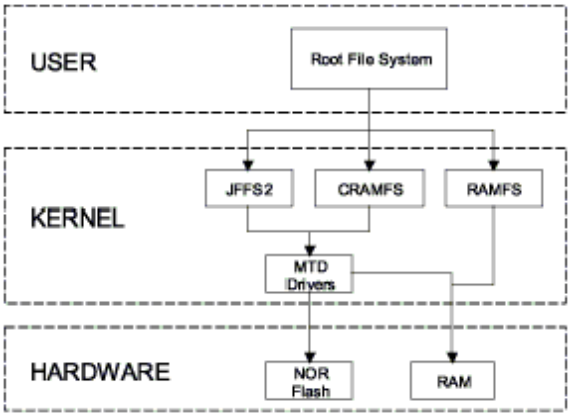


图 5. 基于 Flash 的文件系统的组件

Linux 操作系统的 MTD 子系统是对接存储器设备（如闪存和 RAM）的通用接口，提供对物理存储器设备的简单读、写和擦除操作。名为 mtdblock 的设备可以由 JFFS、JFFS2 和 CRAMFS 文件系统挂载。4 线 SPI NOR MTD 驱动基于内核中的 MTD Data Flash 驱动，增加了 SPI 访问。在初始化阶段，4 线 SPI NOR MTD 驱动通过读取 JEDEC ID 来检测 Data Flash。然后驱动添加 MTD 设备。SPI NOR MTD 驱动还提供用于读取、写入和擦除 NOR 闪存的接口。

3.5.4 驱动特性

此 4 线 NOR 驱动实施支持以下功能：

- 为上层 MTD 驱动提供必要的信息。

3.5.5 源代码结构

表 29. 4 线 SPI 驱动文件

文件	说明
drivers/mtd/spi-nor/spi-nor.c	SPI-NOR 框架
drivers/spi/spi-fsl-qspi.c	4 线 SPI 驱动

3.5.6 菜单配置选项

要启用 4 线 SPI 驱动，必须设置以下选项：

- Device Drivers >Memory Technology Device (MTD) support > Freescale Quad SPI controller

对于配置，将这些设置为启用 4 线 SPI。

- CONFIG_MTD_SPI_NOR：SPI NOR 框架，可供 SPI 设备驱动和 SPI-NOR 设备驱动使用。
- CONFIG_SPI_FSL_QUADSPI：在主模式下启用对 4 线 SPI 控制器的支持。

3.6 SATA

3.6.1 介绍

SATA AHCI 驱动基于 Linux 内核的块设备基础设施的 LIBATA 层。如需了解 SATA 的详细硬件操作，请参见名为 SATA_Data_Book.pdf 的 Synopsys DesignWare Cores SATA AHCI 文档。

3.6.2 电路板配置选项

关闭电源后，安装 SATA 线缆和硬盘驱动。

3.6.3 软件操作

如需了解 libata API 的详细信息，请参见《libATA 开发指南》。

SATA AHCI 驱动基于 Linux 内核块设备基础设施的 LIBATA 层。i.MX 集成式 AHCI Linux 驱动结合了标准的 AHCI 驱动，处理集成式 i.MX SATA AHCI 控制器的详情，而 LIBATA 层理解并执行 SATA 协议。SATA 设备（如硬盘）通过/dev/sda*接口向给用户空间中的应用公开。文件系统在块设备上构建。AHCI 指定了集成的 DMA 引擎，在 DMA 传输模式下协助 SATA 控制器硬件。

3.6.4 源代码结构

i.MX AHCI SATA 驱动的源代码位于 drivers/ata 中。标准 AHCI 和 AHCI 平台驱动用于执行实际的 SATA 操作。

表 30. SATA 驱动文件

文件	说明
drivers/ata/ahci_imx.c	i.MX AHCI SATA 驱动
drivers/ata/ahci.c	标准 AHCI 驱动
drivers/ata/ahci-platform.c	标准 AHCI 平台驱动

3.6.5 菜单配置选项

为 SATA 驱动提供了以下 Linux 内核配置：

```
Symbol: AHCI_IMX [=y]
Type  : tristate
Prompt: Freescale i.MX AHCI SATA support
  Location:
    -> Device Drivers
      -> Serial ATA and Parallel ATA drivers (ATA [=y])
        -> Platform AHCI SATA support (SATA_AHCI_PLATFORM [=y])
```

在 busybox 中，启用“Linux System Utilities”下的“fdisk”。

3.6.6 编程接口

SATA 驱动的应用接口是/dev/sda*上的标准 POSIX 设备接口（例如：open、close、read、write 和 ioctl）。

3.6.7 使用示例

注意

初始化几种 SATA 磁盘时，可能会出现已知的错误消息，例如：
ata1.00: serial number mismatch '090311PB0300QKG3TB1A' != ''
ata1.00: revalidation failed (errno=-19)
应该将其忽略。

1. 构建内核和 SATA AHCI 驱动并部署后，启动目标，并以 root 身份登录。
2. 确保 AHCI 和 AHCI 平台驱动内置在内核中或加载到内核中。

应出现类似于以下内容的消息：

```
ahci: SSS flag set, parallel bus scan disabled
ahci ahci: AHCI 0001.0300 32 slots 1 ports 3 Gbps 0x1 impl platform mode
ahci ahci: flags: ncq sntf stag pm led clo only pmp pio slum part ccc apst
scsi0 : ahci_platform
ata1: SATA max UDMA/133 mmio [mem 0x02200000-0x02203fff] port 0x100 irq 71
ata1: SATA link up 3.0 Gbps (SStatus 123 SControl 300)
ata1.00: ATA-8: SAMSUNG HM100UI, 2AM10001, max UDMA/133
ata1.00: 1953525168 sectors, multi 0: LBA48 NCQ (depth 31/32)
ata1.00: configured for UDMA/133
scsi 0:0:0:0: Direct-Access     ATA      SAMSUNG HM100UI  2AM1 PQ: 0 ANSI: 5
sd 0:0:0:0: [sda] 1953525168 512-byte logical blocks: (1.00 TB/931 GiB)
sd 0:0:0:0: [sda] 4096-byte physical blocks
sd 0:0:0:0: [sda] Write Protect is off
sd 0:0:0:0: [sda] Write cache: enabled, read cache: enabled, doesn't support DPO or FUA
sda: sda1
sd 0:0:0:0: [sda] Attached SCSI disk
```

可使用标准 Linux 实用程序在驱动器（例如：fdisk 和 mke2fs）上进行分区并创建文件系统，供应用安装和使用。

驱动器及其分区的设备节点在/dev/sda*下显示。例如，要检查驱动器的基本内核设置，请执行 hdparm /dev/sda。

3.6.8 使用示例

创建分区

下面的命令可用于确定硬盘的容量。如果硬盘已进行了预格式化，此命令会显示硬盘大小、分区和文件系统类型：

```
$fdisk -l /dev/sda
```

如果硬盘未进行格式化，请使用下面的命令在硬盘上创建分区：

```
$fdisk /dev/sda
```

分区后，创建的文件类似于/dev/sda[1-4]。

块读/写测试：dd 命令用于读/写块。请注意，此命令会损坏硬盘上的分区和文件系统。

要清除卡的前 5 KB，请执行以下操作：

```
$dd if=/dev/zero of=/dev/sda1 bs=1024 count=5
```

响应应如下所示：

5+0 records in
5+0 records out

要将文件内容写入卡中，请输入以下文本，将要写入的文件名替换为 file_name，请执行以下操作：

```
$dd if=file_name of=/dev/sda1
```

要从卡中读取 1KB 的数据，请输入以下文本，将要写入的文件名替换为 output_file，请执行以下操作：

```
$dd if=/dev/sda1 of=output_file bs=1024 count=1
```

文件系统测试

使用 mkfs.vfat 或 mkfs.ext2 格式化硬盘分区，具体取决于文件系统：

```
$mkfs.ext2 /dev/sda1
$mkfs.vfat /dev/sda1
```

装载文件系统，如下所示：

```
$mkdir /mnt/sda1
$mount -t ext2 /dev/sda1 /mnt/sda1
```

装载文件/目录后，可在/mnt/sda1 中进行操作。

按如下方式卸载文件系统：

```
$umount /mnt/sda1
```

3.7 SDMA API

3.7.1 概述

智能直接存储器访问（SDMA）API 驱动控制 SDMA 硬件。它为其他驱动提供了一个 API，用于在 MCU 存储器空间和外设之间传输数据。它支持以下功能：

- 将通道脚本从 MCU 存储器空间加载到 SDMA 内部 RAM
- 加载脚本的上下文参数
- 加载脚本的缓冲区描述符参数
- 控制脚本的执行
- 脚本执行结束时的回调机制

3.7.2 硬件操作

SDMA 控制器负责在 MCU 存储器空间和外设之间传输数据，包括以下功能：

- 多通道 DMA 支持多达 32 个时分复用 DMA 通道。
- 由 16 位指令集 micro-RISC 引擎提供支持。
- 每个通道执行特定的脚本。
- 非常快速的上下文切换，提供基于两级优先级的抢占式多任务处理。
- 4 KB ROM，包含启动脚本（即启动代码）和其他可由 RAM 中的脚本参考的常用实用程序。
- 8 KB RAM 区域分为处理器上下文区域和代码空间区域，用于存储从系统存储器下载的通道脚本。

3.7.3 软件操作

该驱动为其他驱动提供了一个控制 SDMA 通道的 API。SDMA 通道根据外设和传输类型运行专用脚本。SDMA API 驱动负责将脚本加载到 SDMA 存储器中，初始化通道描述符，并控制缓冲区描述符和 SDMA 寄存器。

下表列出了使用 SDMA 的驱动以及每个驱动使用的 SDMA 物理通道的数量。驱动可指定其想要使用的 SDMA 通道号（静态通道分配），也可以让 SDMA 驱动提供空闲的 SDMA 通道供驱动使用（即动态通道分配）。对于动态通道分配，SDMA 通道列表从通道 32 扫描到通道 1。当找到空闲通道时，该通道被分配给请求的 DMA 传输。

表 31. SDMA 通道使用情况

驱动名称	SDMA 通道数	使用的 SDMA 通道
SDMA CMD	1	静态通道分配——使用 SDMA 通道 0
SSI	每台设备 2 个	动态通道分配
UART	每台设备 2 个	动态通道分配
SPDIF	每台设备 2 个	动态通道分配
ESAI	每台设备 2 个	动态通道分配
ASRC	每台设备 6 个	动态通道分配
AUD2HTX	每台设备 1 个	动态通道分配
EASRC	每台设备 8 个	动态通道分配
ECSPI	每台设备 2 个	动态通道分配
MICFIL	每台设备 1 个	动态通道分配
XCVR	每台设备 2 个	动态通道分配

注意

此表包含 SDMA 脚本当前支持的功能，但每个平台的具体实施可能不同。SDMA 支持的外设取决于每个平台的 DTS 配置。

3.7.4 源代码结构

dmaengine.h（SDMA API 的头文件）在目录 linux/include/linux 下提供。

下表展示了在目录 drivers/dma 中提供的源文件。

表 32. SDMA API 源文件

文件	说明
drivers/dma/dmaengine.c	SDMA 管理例程
drivers/dma/imx-sdma.c	SDMA 实施驱动
drivers/dma/imx-dma.c	i.MX DMA 驱动

下表展示了 4.1 和 4.9 内核的 firmware/imx/sdma 目录中提供的镜像文件。对于 4.14 内核，sdma 固件随 firmware-imx 包提供，不在内核源代码树中。

表 33. SDMA 脚本文件

文件	说明
sdma-imx6q.bin	用于 i.MX 6 的 SDMA RAM 脚本
sdma-imx7d.bin	用于 i.MX 7 和 i.MX 8M 的 SDMA RAM 脚本

3.7.5 带 SDMA 外壳的特殊外设

3.7.5.1 i.MX 6/7Dual/8M/中的 I2C

在当前版本中，i.MX 6/7Dual/8M 中的 I2C 控制器和 SDMA 脚本都不支持 SDMA。

有两个限制：

- I2C 在帧长度大于 16 字节时使用 DMA 模式，因为 I2C 本身在发送和接收帧时仍需要使用 CPU 处理前几个字节和后几个字节。因此，当发送的数据不长时，使用 DMA 发送数据并不能提高效率。
- SDMA 脚本在 rootfs 阶段加载，因此在内核启动阶段使用 I2C DMA 进行的传输都将失败。

  强烈建议在发送少量数据时不要使用 I2C SDMA 模式。如果有特殊情况需要发送大量 I2C 数据，请联系 NXP Pro-support 获取补丁集。

3.8 SPI NOR 闪存技术设备（MTD）

3.8.1 介绍

SPI NOR 闪存技术设备（MTD）驱动通过 SPI 接口为 Data Flash 提供支持。默认情况下，SPI NOR 闪存 MTD 驱动创建静态 MTD 分区以支持 Data Flash。

3.8.2 硬件操作

某些电路板上配备了 SPI NOR-AT45DB321D，而某些电路板则配备了 M25P32。检查电路板上的 SPI NOR 类型，然后正确配置。

AT45DB321D 是一款 2.7 V 串行接口顺序存取闪存。AT45DB321D 串行接口与 SPI 兼容，频率最高可达 66 MHz。存储器由 512 字节或 528 字节的 8192 页组成。AT45DB321D 还包含两个 512/528 字节的 SRAM 缓冲区，每个缓冲区允许在对主存储器中的页面进行重新编址时接收数据，以及写入连续的数据流。

M25P32 是一款 32 MB (4M x 8) 串行闪存，具有先进的写保护机制，可通过 75 MHz 的高速 SPI 兼容总线访问。存储器由 64 个扇区组成，每个扇区包含 256 页。 每页宽 256 字节。因此，整个存储器可看作由 16384 页或 4、194、304 字节组成。可以使用 Page Program 指令一次对存储器进行 1 到 256 个字节进行编址。可以使用 Bulk Erase 指令擦除整个存储器，也可以使用 Sector Erase 指令一次擦除一个扇区。

与随机访问的传统闪存不同，这两个 SPI NOR 按顺序访问数据。它们通过单个 2.7-3.6 V 电源进行编址和读取操作。它们通过芯片选择引脚启用，并通过 3 线接口访问：串行输入、串行输出和串行时钟。

3.8.3 软件操作

在一个基于 Flash 的嵌入式 Linux 系统中，许多 Linux 技术协同工作来实施文件系统。下图展示了一些标准组件之间的关系。

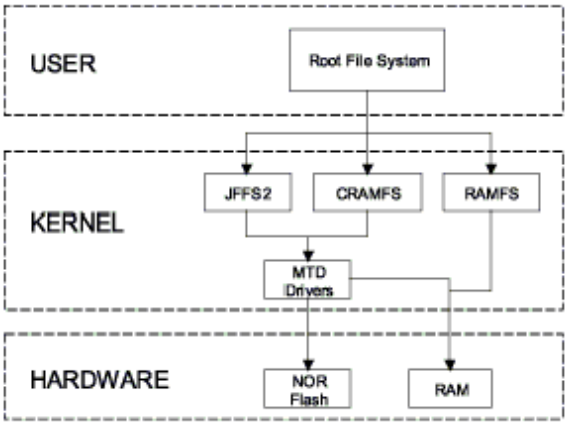


图 6. 基于 Flash 的文件系统的组件

Linux 操作系统的 MTD 子系统是对接存储器设备（如闪存和 RAM）的通用接口，提供对物理存储器设备的简单读、写和擦除操作。名为 mtdblock 的器件可通过 JFFS、JFFS2 和 CRAMFS 文件系统装载。SPI NOR MTD 驱动基于内核中的 MTD Data Flash 驱动，增加了 SPI 访问。在初始化阶段，SPI NOR MTD 驱动通过读取 JEDEC ID 来检测 Data Flash。然后驱动添加 MTD 设备。SPI NOR MTD 驱动还提供了用于读取、写入和擦除 NOR 闪存的接口。

3.8.4 源代码结构

下表展示了驱动文件：

表 34. SPI NOR MTD 驱动文件

文件	说明
drivers/mtd/devices/m25p80.c	源文件
drivers/mtd/spi-nor/spi-nor.c	源文件

3.8.5 菜单配置选项

在菜单配置中启用以下模块：

- CONFIG_MTD_M25P80：此配置允许访问用于编址和数据存储的大多数现代 SPI 闪存芯片。
- Device Drivers > Memory Technology Device (MTD) support >Self-contained MTD device drivers > Support most SPI Flash chips (AT26DF, M25P, W25X, and so on)

第 4 章
连接

4.1 ADC

4.1.1 ADC 介绍

模数转换器的特点如下所示：

- 两个 12 位 ADC
- 线性逐次逼近算法，分辨率为 12 位，精度为 10/11 位
- 1 MS/s 的采样率
- 最多 8 个单端外部模拟输入
- 单次或连续转换（单次转换后自动返回空闲）
- 输出模式：（右对齐无符号格式）
  - 12 位
  - 10 位
  - 8 位
- 可配置的采样时间和转换速度/功率
- 转换完成和硬件平均完成标志和中断
- 输入时钟可从 4 个时钟源中选择
- 用于低噪声操作的异步时钟源，可选择输出时钟
- 带有硬件通道选择的可选异步硬件转换触发器
- 可选电压参考，内部、外部或备用
- 在低功耗模式下运行，以降低噪音
- 硬件平均功能
- 自校准模式

4.1.2 ADC 外部信号

- ADC_VREFH：电压参考高
- ADC_VREHL：参考电压低
- ADC1_IN0：模拟通道 1 输入 0
- ADC1_IN1：模拟通道 1 输入 1
- ADC1_IN2：模拟通道 1 输入 2
- ADC1_IN3：模拟通道 1 输入 3
- ADC2_IN0：模拟通道 2 输入 0
- ADC2_IN1：模拟通道 2 输入 1
- ADC2_IN2：模拟通道 2 输入 2
- ADC2_IN3：模拟通道 2 输入 3

ADC 引脚设置应在 ADCx_PCTL 寄存器中完成。不需要其他额外的 IOMUX 设置。

4.1.3 ADC 驱动概述

ADC 驱动是在 Linux IIO（Industrial I/O）驱动帧下开发的。ADC 驱动仅提供基本功能。支持以下功能：

- 每个 ADC 控制器通道有 4 个外部输入
- 12 位 ADC
- 单次转换
- 硬件平均值
- ADC 的低功耗模式
- 可用采样率组中的采样率变化

4.1.4 源代码结构

表 35. ADC 驱动文件

文件	说明
drivers/iio/adc/vf610_adc.c	i.MX 6UltraLite 和 i.MX 6SoloX ADC 功能。
drivers/iio/adc/imx7d_adc.c	i.MX 7Dual ADC 功能。
drivers/iio/adc/imx8qxp_adc.c	i.MX 8QXP ADC 功能。

4.1.5 菜单配置选项

通过 menuconfig 配置内核选项，以启用以下模块：

Device Drivers > Industrial I/O support > Analog to digital converters > Freescale vf610 ADC driver

Device Drivers > Industrial I/O support > Analog to digital converters > i.MX 7Dual ADC driver

Device Drivers > Industrial I/O support > Analog to digital converters > i.MX 8QXP ADC driver

4.1.6 编程接口

Linux IIO 提供了一些系统接口，从相关输入获取原始 ADC 数据。用户还可以在可用的采样率组中设置采样率。ADC 控制器系统接口位于：

/sys/devices/soc0/soc.1/2200000.aips-bus/2280000.adc/iio:device0:

/sys/devices/soc0/soc.1/2200000.aips-bus/2284000.adc/iio:device1:

下表列出了软件接口。

表 36. 软件接口

软件接口	说明
in_voltage0_raw~ in_voltage3_raw	cat in_voltage0_raw 获取原始 ADC 数据
sampling_frequency_available	cat sampling_frequency_available 获取可用的采样率组
in_voltage_sampling_frequency	cat in_voltage_sampling_frequency 显示当前采样率 echo value > in_voltage_sampling_frequency 设置采样率

4.2 ENET IEEE-1588

4.2.1 介绍

ENET IEEE-1588 驱动执行一组功能，可在网络通信中实现时钟的精确同步。

该驱动需要一个协议栈来完成 IEEE-1588 完整协议。它符合 LinuxPTP 堆栈。

为了实施 IEEE 1588 或类似的时间同步协议，ENET MAC 与时间戳模块相结合，以支持传入和传出帧的精确时间戳。

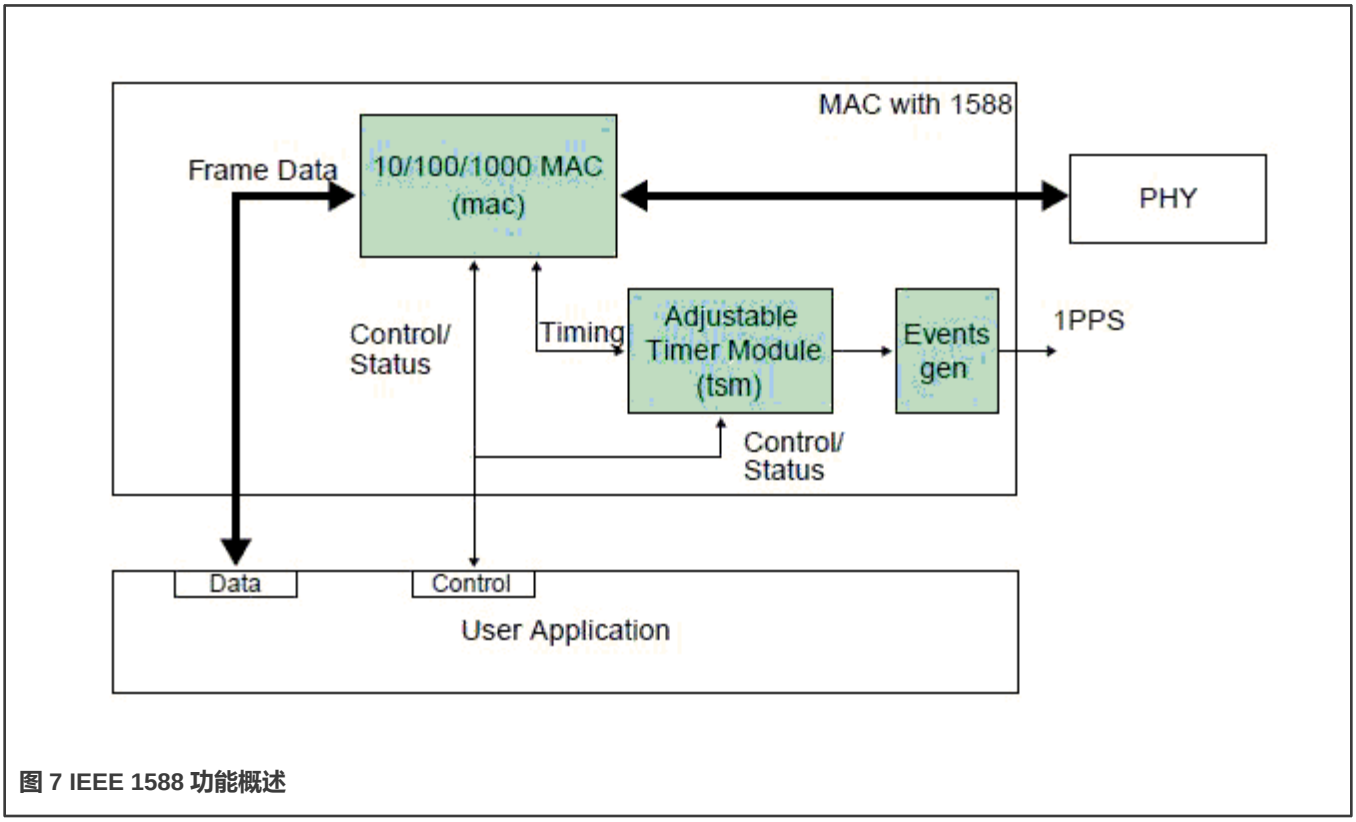


图 7 IEEE 1588 功能概述

4.2.1.1 发送时间戳

在发送时，只有 1588 事件帧需要打时间戳。客户端应用（例如 MAC 驱动）应检测 1588 事件帧，并将信号 ff_tx_ts_frm 与帧一起设置。

对于每一个发送的帧，MAC 返回 tx_ts (31:0)上捕获的时间戳以及帧序列号（tx_ts_id(3:0)）和发送状态。发送状态位 tx_ts_stat (5)表示应用已为该帧产生了 ff_tx_ts_frm 信号。

如果 ff_tx_ts_frm 设置为“1”，MAC 会将帧的时间戳记在寄存器 TS_TIMESTAMP 中。设置中断位 EIR（TS_AVAIL），指示新的时间戳可用。

软件将通过在传输需要时间戳的帧时设置 ff_tx_ts_frm 信号来实施握手过程，然后等待 EIR（TS_AVAIL）中断位，了解时间戳何时可用。之后，它可以从 TS_TIMESTAMP 寄存器中读取时间戳。为所有事件帧都执行上述操作；其他帧不使用 ff_tx_ts_frm 指示符，因此不会干扰时间戳捕获。

4.2.1.2 接收时间戳

当接收到帧时，MAC 在检测到帧 SFD 字段时锁定定时器的值，并在 ff_rx_ts(31:0)上提供捕获的时间戳。对所有接收到的帧执行上述操作。

DMA 控制器必须确保将为帧提供的时间戳传输到接收描述符中的相应字段，以进行软件访问。

4.2.2 软件操作

1588 驱动具有下列功能：

- 模块初始化——使用设备特定的结构对模块进行初始化，并注册字符驱动。
- 中断服务例程——支持事件，例如 TS_AVAIL、TS_TIMER。该驱动与 FEC 驱动共享中断服务例程。

4.2.2.1 源代码结构

下表列出了 drivers/net/ethernet/freescale 目录中的源文件。

表 37. ENET 1588 文件列表

文件	说明
drivers/net/ethernet/frescale/fec.h	定义寄存器的头文件
drivers/net/ethernet/freescale/fec_ptp.c	ENET 1588 定时器

4.2.2.2 菜单配置选项

默认启用 ENET 1588。

4.2.2.3 编程接口

1588 驱动程符合 Linuxptp 协议栈接口。堆栈特定的定义被添加到头文件（fec.h）。

4.2.3 1588 堆栈介绍

此版本支持以下类型的 1588 堆栈：

- Linuxptp 堆栈

  该软件是根据 IEEE 1588 Linux 操作系统标准的精确时间协议（PTP）实施的。它具有双重设计目标，即稳健实施标准，并使用 Linux 操作系统内核提供的最相关、最新的应用编程接口（API）。支持遗留 API 和其他平台不是目标。该软件受作者版权保护，并根据 GNU 通用公共许可获得了许可。

软件开发托管在 Source Forge：sourceforge.net/projects/linuxptp/

4.2.3.1 Linuxptp 堆栈特性

Linuxptp 支持以下功能：

- 普通/边界时钟
- 最优主时钟算法
- 通过 UDP/IPv4、UDP/IPv6 和 IEEE 802.3 进行传输
- 透明时钟（E2E/P2P）
- 仅限从模式
- 作为终端站支持 IEEE 802.1AS-2011

4.2.3.2 使用 Linuxptp

使用以下命令运行 ptp4 1588 堆栈二进制文件。

Linuxptp：

```
Transport on UDP IPV4 with E2E delay mechanism: ptp4l -A -4 -H -m -i eth0
Transport on UDP IPV4 with P2P delay mechanism: ptp4l -P -A -4 -H -m -i eth0
Transport on UDP IPV6 with E2E delay mechanism: ptp4l -A -6 -H -m -i eth0
Transport on UDP IPV6 with P2P delay mechanism: ptp4l -P -A -6 -H -m -i eth0
Transport on IEEE 802.3 with E2E delay mechanism: ptp4l -A -2 -H -m -i eth0
Transport on IEEE 802.3 with P2P delay mechanism: ptp4l -P -A -2 -H -m -i eth0
```

4.3 增强型可配置串行外设接口（ECSPI）

4.3.1 介绍

ECSPI 驱动实施了与 ECSPI 控制器对接的标准 Linux 驱动接口。

它支持以下功能：

- 中断驱动的字节发送/接收
- 多主控制器接口
- 多从器件选择
- 多客户端请求

ECSPI 用于快速数据通信，与传统串行通信相比，软件中断更少。>每个 ECSPI 都配备了数据 FIFO，是一个主/从可配置串行外设接口模块，允许处理器与外部 SPI 主设备或从设备进行对接。

ECSPI 的主要功能包括：

- 主/从可配置
- 4 个片选信号，支持多个外设
- 32 位可编程数据传输
- 64 x 32 位 FIFO，用于发送和接收数据
- 片选（SS）和 SPI 时钟（SCLK）的极性和相位可配置

ECSPI 模块支持以下功能：

- 实施 ECSPI 模块与 Linux 操作系统对接所需的每个功能
- 多个 SPI 主控制器
- 多客户端同步请求

4.3.2 软件操作

以下各节介绍了 ECSPI 软件操作。

4.3.3 Linux 操作系统中的 SPI 子系统

ECSPI 驱动层位于客户端层（SPI-NOR 闪存是客户端示例）和硬件访问层之间。下图展示了 Linux 操作系统中的 SPI 子系统的框图。

SPI 请求进入 I/O 队列。对给定 SPI 设备的请求按 FIFO 顺序执行，并通过完成回调异步完成。这些调用也有一些简单的同步包装器（wrapper），包括用于常见事务类型的调用，例如编写命令然后读取其响应。

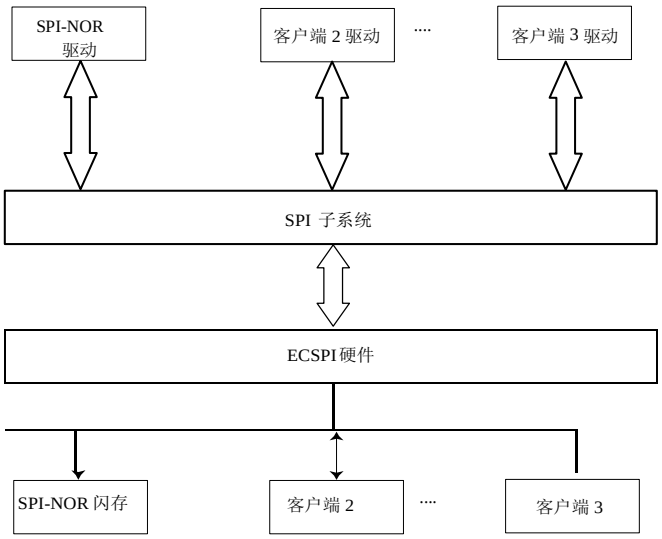


图 8. SPI 子系统

所有 SPI 客户端都必须有与之关联的协议驱动，并且必须共享同一个控制器驱动。只有控制器驱动才能与底层 SPI 硬件模块进行交互。下图展示了不同的 SPI 驱动在 SPI 子系统中是如何分层的。

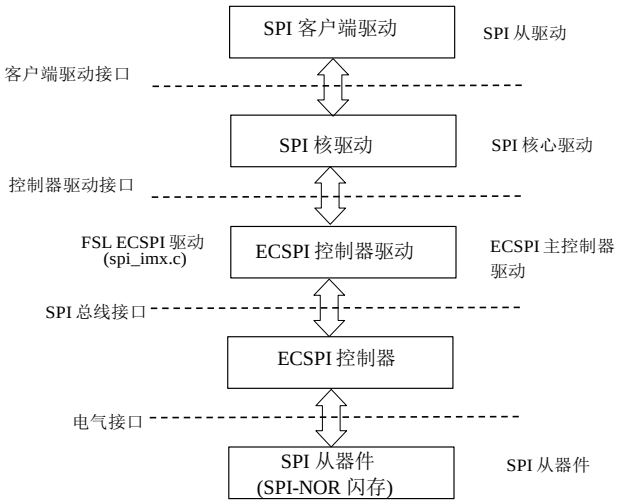


图 9. SPI 子系统中 SPI 驱动的分层

4.3.4 软件限制

ECSPI 驱动限制如下所示：

- 当前未实现 SPI 从逻辑
- 不支持单个客户端连接到多个主器件
- 目前未借助设备节点入口实现用户空间接口，但支持 sysfs 接口

4.3.5 标准操作

ECSPI 驱动负责为 init、exit、chip select 和 transfer 实施标准入口点。该驱动实施以下函数：

- 初始化函数 spi_imx_init()注册 device_driver 结构。
- 探测函数 spi_imx_probe()使用 SPI 核心驱动执行 SPI 设备特定结构的初始化和注册。该驱动探测存储器和 IRQ 资源。配置 IOMUX 以启用 ECSPI I/O 引脚、请求 IRQ 并重置硬件。
- 片选函数 spi_imx_chipselect()为当前 SPI 设备配置硬件 ECSPI。设置此设备的字长、传输模式和数据速率。
- SPI 传输函数 spi_imx_transfer()处理数据传输操作。
- SPI 设置函数 spi_imx_setup()初始化当前 SPI 设备。
- SPI 驱动 ISR spi_imx_isr()在数据传输操作完成并产生中断时被调用。

4.3.6 ECSPI 同步操作

下图展示了 ECSPI 如何提供同步读/写操作。

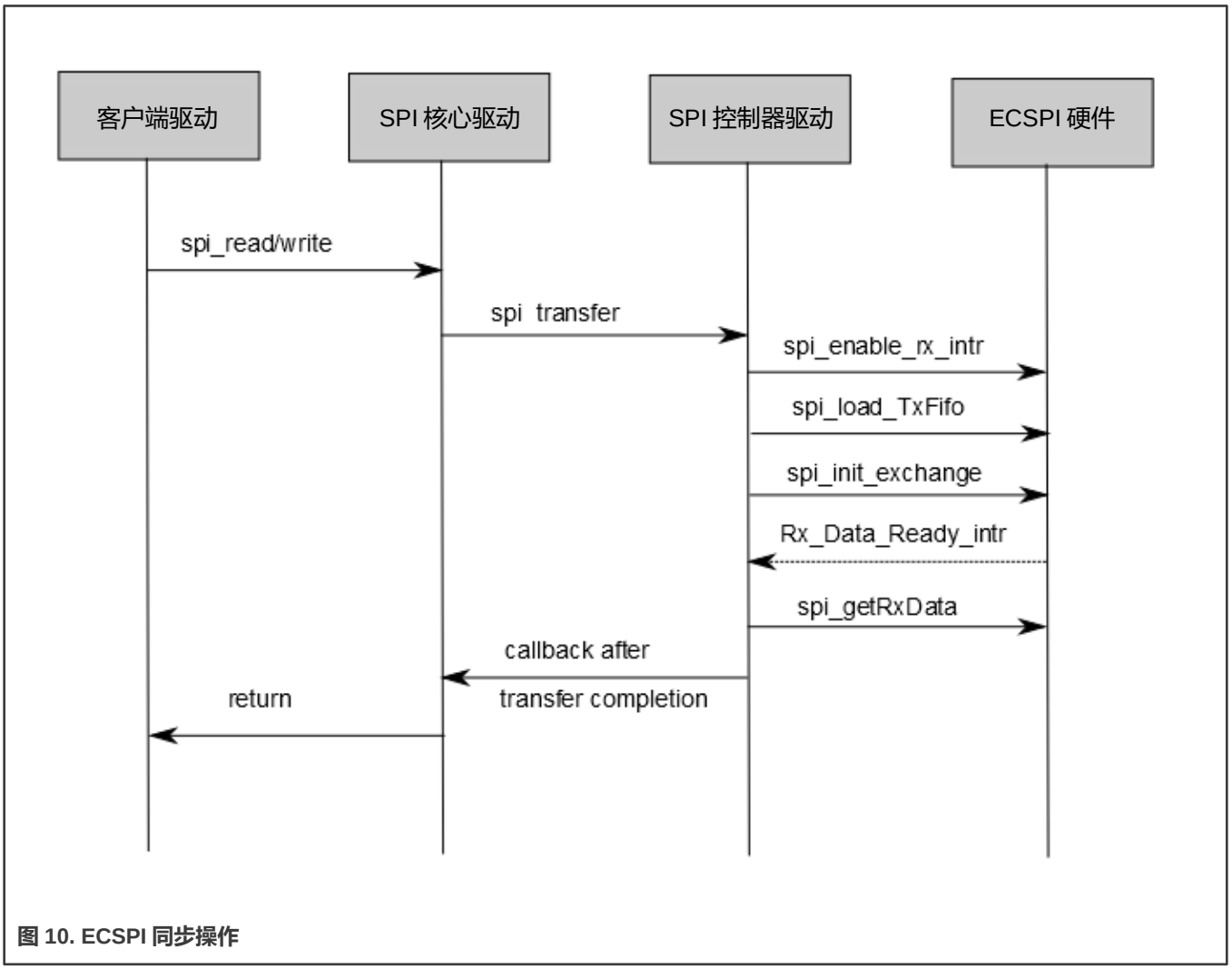


图 10. ECSPI 同步操作

4.3.7 源代码结构

下表列出了 drivers/spi 目录中的源文件：

表 38. ECSPI 驱动文件

文件	说明
driveers/spi/spi-imx.c	SPI 主控制器驱动

4.3.8 菜单配置选项

在菜单配置中启用以下模块：

- CONFIG_SPI 构建 SPI 核心支持。在 menuconfig 中，此选项位于以下位置：
  - Device Drivers > SPI Support。
- CONFIG_BITBANG 是由需要它的驱动自动选择的库代码。SPI_IMX 选择它。在 menuconfig 中，此选项位于以下位置：
  - Device Drivers > SPI Support > Utilities for Bitbanging SPI masters。
- CONFIG_SPI_IMX 为 ECSPI 实施 SPI 主模式。在 menuconfig 中，此选项位于以下位置：
  - Device Drivers > SPI Support > Freescale i.MX SPI controllers。

4.3.9 编程接口

该驱动实施 SPI 核心与 ECSPI 硬件对接所需的所有函数。

如需了解更多信息，请参见从 build: make htmldocs 生成的 Linux 文档。

4.3.10 中断要求

SPI 接口产生中断。

下表列出了 ECSPI 中断要求。

表 39. ECSPI 中断要求

参数	方程式	典型值	最坏情况
波特率/传输长度	（波特率/（传输长度））*（1/Rxtl）	31250	1500000

典型值基于 1 Mbps 的波特率，接收器触发电平（Rxtl）为 1，传输长度为 32 位。最坏情况基于 12 Mbps 波特率（SPI 接口支持的最大值），传输长度为 8 位。

4.4 快速以太网控制器（FEC）

4.4.1 介绍

快速以太网控制器（FEC）驱动执行全套 IEEE 802.3/以太网 CSMA/CD 媒体访问控制和通道接口功能。

FEC 需要外部接口适配器和收发器功能来实现与以太网介质的对接。它支持在 10 Mbps、100 Mbps 和 1000 Mbps 相关以太网上进行半双工或全双工操作。

FEC 驱动支持以下功能：

- 全双工/半双工操作
- 链路状态变化检测

- 自动协商（确定网络速度和全双工或半双工操作）
- 传输功能，例如在发生冲突和生成 CRC 时自动重传
- 从设备获取统计信息，例如传输冲突

可通过接口名为 ethx 的 ifconfig 命令访问网络适配器。该驱动自动探测外部适配器（PHY 设备）。

4.4.2 硬件操作

FEC 表示以太网控制器，将系统连接到 LAN 网络。

FEC 支持不同的标准 MAC-PHY（物理）接口，用于连接到外部以太网收发器。FEC 支持 10/100 Mbps MII、10/100 Mbps RMII 和 10/100/1000 Mbps RGMII。此外，FEC 支持 1000 Mbps RGMII，它使用在 125 MHz 运行的 4 位精简 GMII。

此处简要介绍了设备功能。如需了解详细信息，请参见《应用处理器参考手册》的 FEC 章节

在 MII 模式下，有 18 个由 IEEE 802.3 标准定义并由 EMAC 支持的信号。MII、RMII 和 RGMII 模式使用 18 个信号的子集。下表列出了这些信号。

表 40. MII、RMII 和 RGMII 模式下的引脚使用

方向	EMAC 引脚名称	RMII 使用	RGMII 使用（i.MX 6UltraLite 不支持）
进/出	FEC_MDIO	管理数据输入/输出	管理数据输入/输出
出	FEC_MDC	一般输出	管理数据时钟
出	FEC_TXD[0]	数据输出，位 0	数据输出，位 0
出	FEC_TXD[1]	数据输出，位 1	数据输出，位 1
出	FEC_TXD[2]	不使用	数据输出，位 2
出	FEC_TXD[3]	不使用	数据输出，位 3
出	FEC_TX_EN	发送启用	发送启用
出	FEC_TX_ER	不使用	不使用
入	FEC_CRS	不使用	不使用
入	FEC_COL	不使用	不使用
入	FEC_TX_CLK	不使用	同步时钟参考（REF_CLK，可从 PHY 连接）
入	FEC_RX_ER	接收错误	不使用
入	FEC_RX_CLK	不使用	同步时钟参考（REF_CLK，可从 PHY 连接）
入	FEC_RX_DV	接收数据有效并生成 CRS	RXDV XOR RXERR 在 FEC_RX_CLK 的下降沿。
入	FEC_RXD[0]	数据输入，位 0	数据输入，位 0
入	FEC_RXD[1]	数据输入，位 1	数据输入，位 1
入	FEC_RXD[2]	不使用	数据输入，位 2
入	FEC_RXD[3]	不使用	数据输入，位 3

MII 管理接口由两个引脚组成：FEC_MDIO 和 FEC_MDC。FEC 硬件操作可分为以下几部分。如需了解详细信息，请参见《应用处理器参考手册》。

- 传输：以太网发射器的设计为无需软件干预即可工作。一旦 ECR[ETHER_EN]被钳位且数据出现在发送 FIFO 中，以太网 MAC 就能够发送到网络上。当发送 FIFO 填充到水印（由 TFWR 定义）时，MAC 发送逻辑将 FEC_TX_EN 钳位，并开始发送前导（PA）序列、起始帧定界符（SFD），然后是来自 FIFO 的帧信息。然而，如果网络繁忙（FEC_CRS 钳位），控制器会延迟发送。
- 在发送之前，控制器等待载波侦听变为非活跃状态，然后确定载波侦听是否保持非活跃状态达 60 位时间。如果发送在等待额外的 36 位时间（载波侦听最初变为非活跃状态后的 96 位时间）后开始，可能生成缓冲区（TXB）和帧（TXF）中断，具体取决于 EIMR 中的设置。
- 接收：FEC 接收器设计为在几乎无需主机干预的情况下便可工作，可以执行地址识别、CRC 检查、短帧校验和最大帧长校验。当驱动通过钳位 ECR[ETHER_EN]启用 FEC 接收器时，它会立即开始处理接收帧。当 FEC_RX_DV 钳位时，接收器检查有效的 PA/SFD 头。如果 PA/SFD 有效，则将其剥离，由接收器对帧进行处理。如果未找到有效的 PA/SFD，则忽略该帧。在 MII 模式下，接收器检查是否至少有一个字节与 SFD 匹配。可能出现零个或多个 PA 字节，但如果在 SFD 字节之前检测到 00 位序列，则忽略该帧。
- 在接收到帧的前 6 个字节后，FEC 对帧执行地址识别。在接收过程中，以太网控制器检查各种错误情况，一旦将整个帧写入了 FIFO，32 位帧状态字将写入 FIFO。该状态字包含 M、BC、MC、LG、NO、CR、OV 和 TR 状态位以及帧长度。如果由 EIMR 寄存器启用，则可能会产生接收缓冲区（RXB）和帧中断（RXF）。当接收帧完成时，FEC 设置 RxBD 中的 L 位，将其他帧状态位写入 RxBD，并清除 E 位。以太网控制器接下来生成一个可屏蔽中断（EIR 中的 RXF 位，可被 EIMR 中的 RXF 位屏蔽），指示帧已被接收并保存在存储器中。然后，以太网控制器等待新的帧。
- 中断管理：当发生在 EIR 中设置位的事件时，如果中断屏蔽寄存器（EIMR）中的相应位也被设置，则产生中断。如果将 1 写入该位位置，则清除 EIR 中的位；写零没有效果。该寄存器在硬件重置时被清除。这些中断可分为操作中断、收发器/网络错误中断和内部错误中断。正常运行中可能发生的中断包括 GRA、TXF、TXB、RXF、RXB。网络或收发器中检测到的错误/问题导致的中断包括 HBERR、BABR、BABT、LC 和 RL。内部错误导致的中断包括 HBERR 和 UN。一些错误中断在 MIB 块计数器中单独计数。软件可以选择屏蔽这些中断，因为网络管理人员可以通过 MIB 计数器看到这些错误。
- PHY 管理：phylib 用于管理所有 FEC PHY 相关操作，如 PHY 发现、链路状态和状态机。MDIO 总线将在 FEC 驱动中创建并注册到系统中。如需了解详细信息，请参见 Linux OS 源文件目录下的 Documentation/networking/ phy.txt。

4.4.3 软件操作

FEC 驱动支持以下功能：

- 模块初始化——使用设备特定结构对模块进行初始化
- Rx/Tx 传输
- 中断服务例程
- 物理层管理
- FEC 管理，例如初始化/启动/停止
- i.MX 6 FEC 模块使用 little-endian 格式

4.4.4 源代码结构

下表展示了源文件。

这些源文件位于 drivers/net/ethernet/freescale 目录中。

表 41. FEC 驱动文件

文件	说明
drivers/net/ethernet/freescale/fec.h	定义寄存器的头文件
drivers/net/ethernet/freescale/fec_main.c	以太网 LAN 控制器的 Linux 驱动
drivers/net/ethernet/freescale/fec_fixup.c	SoC 和 PHY 专用工具的 Linux 驱动

4.4.5 菜单配置选项

配置内核以提供此模块：

- 为该模块提供了 CONFIG_FEC。此选项位于以下位置：
  - Device Drivers > Network device support > Ethernet (10, 100 or 1000 Mbit) > FEC Ethernet controller。
  - 要通过 FEC 挂载 NFS-rootfs，如需要，请在 menuconfig 中禁用其他网络配置。

4.4.6 编程接口

设备特定的定义被添加到头文件（fec.h）中，并提供公共电路板配置选项。

fec.h 定义寄存器访问的结构和缓冲区描述符的结构。例如

```
/*
*      Define the buffer descriptor structure.
*/
struct bufdesc {
         unsigned short            cbd_datlen;       /* Data length */
         unsigned short            cbd_sc;           /* Control and status info */
         unsigned long             cbd_bufaddr;      /* Buffer address */
};
struct bufdesc_ex {
       struct bufdesc desc;
       unsigned long cbd_esc;
       unsigned long cbd_prot;
       unsigned long cbd_bdu;
       unsigned long ts;
       unsigned short res0[4];
};
/*
*      Define the register access structure.
*/
#define FEC_IEVENT              0x004 /* Interrupt event reg */
#define FEC_IMASK               0x008 /* Interrupt mask reg */
#define FEC_R_DES_ACTIVE        0x010 /* Receive descriptor reg */
#define FEC_X_DES_ACTIVE        0x014 /* Transmit descriptor reg */
#define FEC_ECNTRL              0x024 /* Ethernet control reg */
#define FEC_MII_DATA            0x040 /* MII manage frame reg */
#define FEC_MII_SPEED           0x044 /* MII speed control reg */
#define FEC_MIB_CTRLSTAT        0x064 /* MIB control/status reg */
#define FEC_R_CNTRL             0x084 /* Receive control reg */
#define FEC_X_CNTRL             0x0c4 /* Transmit Control reg */
#define FEC_ADDR_LOW            0x0e4 /* Low 32bits MAC address */
#define FEC_ADDR_HIGH           0x0e8 /* High 16bits MAC address */
```

```
#define FEC_OPD                 0x0ec /* Opcode + Pause duration */
#define FEC_HASH_TABLE_HIGH     0x118 /* High 32bits hash table */
#define FEC_HASH_TABLE_LOW      0x11c /* Low 32bits hash table */
#define FEC_GRP_HASH_TABLE_HIGH 0x120 /* High 32bits hash table */
#define FEC_GRP_HASH_TABLE_LOW  0x124 /* Low 32bits hash table */
#define FEC_X_WMRK              0x144 /* FIFO transmit water mark */
#define FEC_R_BOUND             0x14c /* FIFO receive bound reg */
#define FEC_R_FSTART            0x150 /* FIFO receive start reg */
#define FEC_R_DES_START         0x180 /* Receive descriptor ring */
#define FEC_X_DES_START         0x184 /* Transmit descriptor ring */
#define FEC_R_BUFF_SIZE         0x188 /* Maximum receive buff size */
#define FEC_MIIGSK_CFGR         0x300 /* MIIGSK config register */
#define FEC_MIIGSK_ENR          0x308 /* MIIGSK enable register */
```

4.4.6.1 获取 MAC 地址

MAC 地址可以通过内核命令行、内核设备树 DTS 文件、OCOTP 或引导加载程序设置的 MAC 寄存器（如 U-Boot）来设置。FEC 驱动使用它来配置网络设备的 MAC 地址。一般情况下，使用内核命令行以 fec.macaddr=0x00,0x04,0x9f,0x01,0x30,0xe0 的形式设置 MAC 地址。由于某些引脚冲突（FEC RMII 模式需要使用 GPIO_16 或 RGMII_TX_CTL 引脚作为参考时钟输入/输出通道），两个引脚都无法连接到其他模块使用的分支线，因为分支线对时钟有严重影响。

4.5 FlexCAN

4.5.1 介绍

FlexCAN 是根据 CAN 2.0B 协议规范实施 CAN 协议的通信控制器。

CAN 协议主要设计用作满足该领域特定要求的车辆串行数据总线，如实时处理、车辆在 EMI 环境下的可靠运行、成本效益和所需带宽。支持标准和扩展消息帧。最大消息缓冲区为 64。该驱动是 PF_CAN 协议系列的网络设备驱动。

如需了解详细信息，请参见 lwn.net/Articles/253425 或 Linux 源文件目录中的 Documentation/networking/can.txt。

i.MX 8QuadMax/8QuadXPlus 上的 FlexCAN 支持 CAN FD 协议。

4.5.1.1 软件操作

CAN 驱动是一种网络设备驱动。如需了解软件操作的常见信息，请参见内核源文件目录 Documentation/networking/can.txt 中的文档。

CAN 网络设备驱动接口为设置、配置和监测 CAN 网络设备提供了通用接口。然后，用户可使用“IPROUTE2”实用程序套件中的程序“ip”，通过 netlink 接口配置 CAN 设备，如设置位定时参数。

启动和停止 CAN 网络设备。

可以像往常一样通过命令“ifconfig canX up/down”或“ip link set canX up/down”启动或停止 CAN 网络设备。请注意，在启动真实 CAN 设备之前，*必须*为其定义适当的位定时参数，以避免容易出错的默认设置：

- ip link set canX up type can 比特率 125000

iproute2 工具还为 CAN 总线提供了一些其他配置功能，如位定时设置。如需了解详细信息，请参见内核文档：Documentation/networking/can.txt

4.5.1.2 源代码结构

下表列出了 drivers/net/can 中的驱动源文件

表 42. FlexCAN 驱动文件

文件	说明
drivers/net/can/flexcan.c	FlexCAN 驱动

4.5.1.3 菜单配置选项

为该模块提供了以下内核配置选项。

- CONFIG_CAN：构建对 PF_CAN 协议系列的支持。在 menuconfig 中，此选项位于“Networking > CAN bus subsystem support”。
- CONFIG_CAN_RAW：构建对原始 CAN 协议的支持。在 menuconfig 中，此选项位于“Networking > CAN bus subsystem support > Raw CAN Protocol (raw access with CAN-ID filtering)”。
- CONFIG_CAN_BCM：构建对 Broadcast Manager CAN 协议的支持。在 menuconfig 中，此选项位于“Networking > CAN bus subsystem support > Broadcast Manager CAN Protocol (with content filtering)”。
- CONFIG_CAN_VCAN：构建对虚拟本地 CAN 接口（也在以太网接口中）的支持。在 menuconfig 中，此选项位于“Networking > CAN bus subsystem support > CAN Device Driver > Virtual Local CAN Interface (vcan)”。
- CONFIG_CAN_DEBUG_DEVICES：构建支持，将调试消息生成到驱动的系统日志中。在 menuconfig 中，此选项位于“Networking > CAN bus subsystem support > CAN Device Driver > CAN devices debugging messages”。
- CONFIG_CAN_FLEXCAN：构建对 FlexCAN 设备驱动的支持。在 menuconfig 中，此选项位于“Networking > CAN bus subsystem support > CAN Device Driver > Freescale FlexCAN”。

4.6 I2C

4.6.1 介绍

LPI2C 是一种双向串行总线，提供了一种简单、高效的数据交换方法，最大限度地减少了设备之间的互连。

Linux 操作系统的 LPI2C 驱动由两个部分组成：

- 总线驱动——用于与 LPI2C 总线进行通信的低级接口
- 芯片驱动——其他设备驱动与 LPI2C 总线驱动之间的接口

I2C 总线驱动是用于与 I2C 总线对接的低级接口。此驱动由 I2C 芯片驱动调用，不向用户空间公开。标准 Linux 内核包含一个核心 I2C 模块，芯片驱动使用该模块访问总线驱动，通过 I2C 总线传输数据。该总线驱动支持以下功能：

- 与 I2C 总线标准兼容
- 比特率高达 400 Kbps
- 标准 I2C 主模式
- 通过暂停和恢复 I2C 实现电源管理功能。

4.6.2 LPI2C 总线驱动概述

LPI2C 总线驱动仅由芯片驱动调用，不向用户空间公开。标准 Linux 内核包含一个核心 I2C 模块，芯片驱动使用该模块访问 LPI2C 总线驱动，通过 LPI2C 总线传输数据。芯片驱动使用 Linux 内核中提供的标准内核空间 API 来访问核心 I2C 模块。标准 I2C 内核函数记录在内核源文件树的 Documentation/i2c 下的文件中。该总线驱动支持以下功能：

- 与 I2C 总线标准兼容
- 中断驱动的逐字节数据传输
- 标准 I2C 主模式

4.6.3 I2C 器件驱动概述

I2C 器件驱动实施了与 LPI2C 总线驱动进行通信所需的所有 Linux I2C 数据结构。它向其他设备驱动公开了自定义内核空间 API，以便将数据传输到连接到 LPI2C 总线的设备。在内部，这些 API 函数使用标准的 I2C 内核空间 API 调用 I2C 核心模块。I2C 核心模块查找 LPI2C 总线驱动，并调用 LPI2C 总线驱动中的相应函数进行数据传输。该驱动为其他设备驱动提供以下功能：

- 读取设备寄存器的读取功能
- 写入设备寄存器的写入功能

4.6.4 软件操作

Linux 操作系统的 I2C 驱动由两部分组成：I2C 总线驱动和 I2C 芯片驱动。

4.6.5 I2C 总线驱动软件操作

I2C 总线驱动采用名为 i2c_adapter 的结构。该结构中最重要的字段是 struct i2c_algorithm *algo。该字段是指向 i2c_algorithm 结构的指针，描述了数据如何通过 I2C 总线进行传输。算法结构包含一个指向函数的指针，每当 I2C 芯片驱动想要与 I2C 器件进行通信时，就会调用该函数。

在启动过程中，加载驱动时，I2C 总线适配器将注册到 I2C 核心。某些架构有多个 I2C 模块。在这种情况下，驱动会为每个具有 I2C 核心的 I2C 模块注册单独的 i2c_adapter 结构。卸载驱动时，这些适配器将被注销（删除）。

在正常通信过程中，当传输出现某些错误情况时，例如检测到 NACK 时，它会超时并返回错误。当出现错误情况时，I2C 驱动应停止当前的传输。

4.6.6 I2C 器件驱动软件操作

I2C 驱动控制 I2C 总线上的单个 I2C 器件。i2c_driver 结构描述了 I2C 芯片驱动。此结构中的相关字段是 flags 和 attach_adapter。flags 字段设置为 I2C_DF_NOTIFY 值，以便在加载了驱动后，可告知芯片驱动任何新的 I2C 器件。加载 I2C 总线驱动时，该驱动存储与该总线驱动关联的 i2c_adapter 结构，以便采用适当的方法传输数据。

4.6.7 驱动特性

LPI2C 驱动支持以下功能：

- I2C 通信协议
- I2C 主操作模式

注意

LPI2C 驱动不支持从模式。

4.6.8 源代码结构

下表列出了 drivers/i2c/busses 中的驱动源文件

表 43. I2C 驱动文件

文件	说明
drivers/i2c/busses/imx-lpi2c.c	用于 i.MX 8 和 i.MX 8X 的 LPI2C 总线驱动
drivers/i2c/busses/imx-i2c.c	用于 i.MX 6、i.MX 7 和 i.MX 8M 的 I2C 总线驱动

4.6.9 菜单配置选项

通过 menuconfig 配置内核选项，以启用该模块：

对于 i.MX 6、i.MX 7 和 i.MX 8M，选择“Device Drivers > I2C support > I2C Hardware Bus support > IMX I2C interface”。

对于 i.MX 8 和 i.MX 8X，选择“Device Drivers > I2C support > I2C Hardware Bus support > IMX Low Power I2C interface”。

4.6.10 编程接口

LPI2C 设备驱动可以使用标准 SMBus 接口读取和写入连接到 LPI2C 总线的设备的寄存器。如需了解更多信息，请参见 include/linux/i2c.h。

4.7 媒体本地总线（MediaLB）

4.7.1 介绍

MediaLB 是一种 PCB 或芯片间通信总线，专用于标准化通用硬件接口和软件 API 库。

这种标准化允许一个或多个应用以最小的工作量访问 MOST Network 数据或与其他应用进行通信。MediaLB 支持所有 *MOST Network 数据传输*方法：同步流数据、异步分组数据和控制消息数据。MediaLB 还支持等时数据传输方法。如需了解 MediaLB 的详细信息，请参见《媒体本地总线规范》。

MediaLB 模块实施 MediaLB 规范的物理层和链路层，将 i.MX 连接到 MediaLB 控制器。

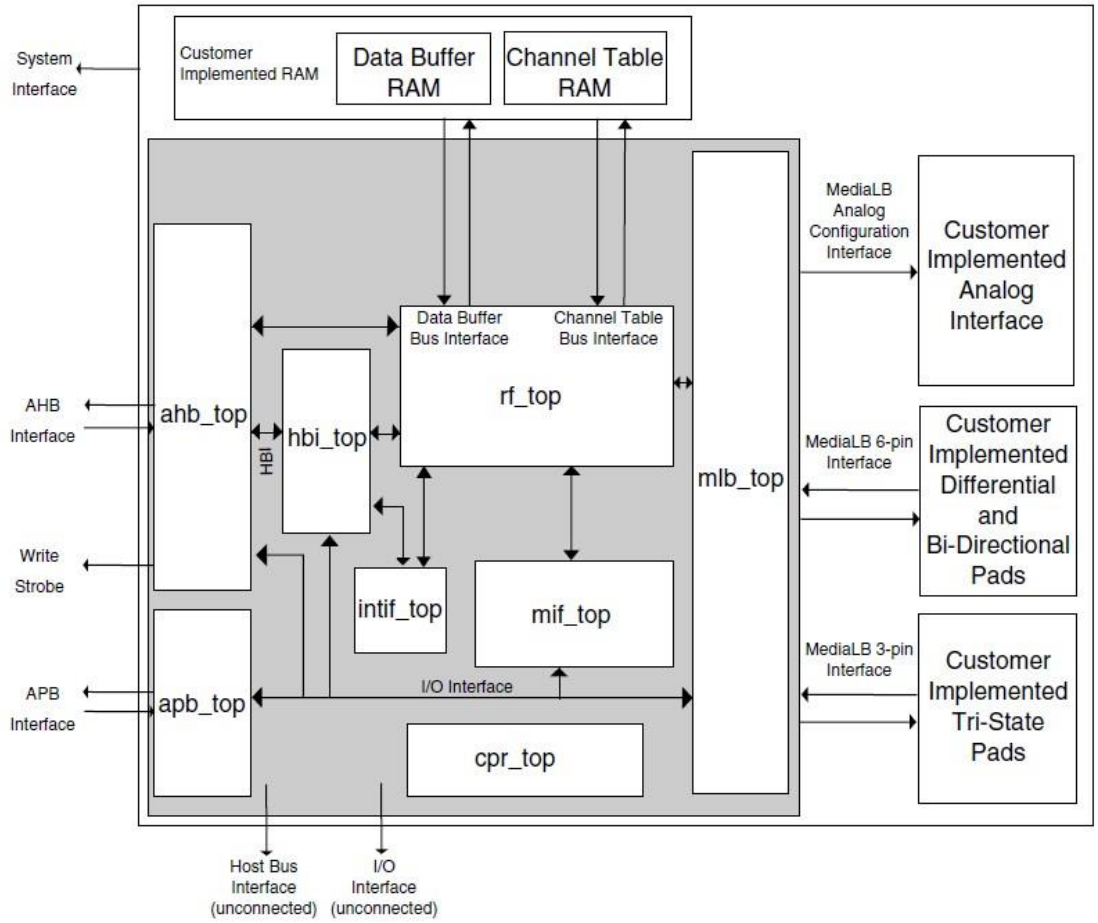


图 11. MLB 设备顶层框图

MLB 实施 3 引脚 MediaLB 模式，运行速度最高可达 1024Fs。它不实现 MediaLB 控制器功能。所有 MediaLB 设备都支持一组物理通道，用于通过 MediaLB 发送数据。每个物理通道的长度为 4 字节（quadlet），并分组为逻辑通道，向每个逻辑通道分配一个或多个物理通道。这些逻辑通道可以是通道类型（同步、异步、控制或等时）和方向（发送或接收）的任意组合。

MLB 最多支持 64 个逻辑通道和 64 个物理通道。每个逻辑通道都使用唯一的通道地址表示，并指示发送数据的 MediaLB 设备和接收数据的 MediaLB 设备之间的单向数据通路。

支持的功能如下所示。

- 同步、异步、控制和等时通道。
- 多达 64 个逻辑通道和 64 个物理通道，最高运行速度为 1024Fs。
- 当作为与逻辑通道地址相关联的发送设备时，发送命令和数据和接收 Rx 状态。
- 当作为与逻辑通道地址相关联的接收设备时，接收命令和数据，并作为接收状态响应进行传输。
- MediaLB 锁定检测。
- 系统通道命令处理。
- 256Fs、512Fs 和 1024Fs 帧速率。

- 异步、控制、同步和等时通道类型。
- MLB 设备模块的以下配置：
  - 帧速率
  - 设备地址
  - 通道地址
- MLB 通道异常获取接口。所有通道异常都由应用发送和处理。

4.7.2 MLB 驱动概述

MLB 驱动是一个通用的 Linux 操作系统字符驱动。它实现了一个异步和一个 Ping-Pong 缓冲操作模式的控制通道设备。支持的帧速率分别为 256、512 和 1024Fs。MLB 驱动使用公共读/写接口来接收/发送数据包，并使用 ioctl 接口来配置 MLB 设备模块。

MLB 驱动架构如下图所示。

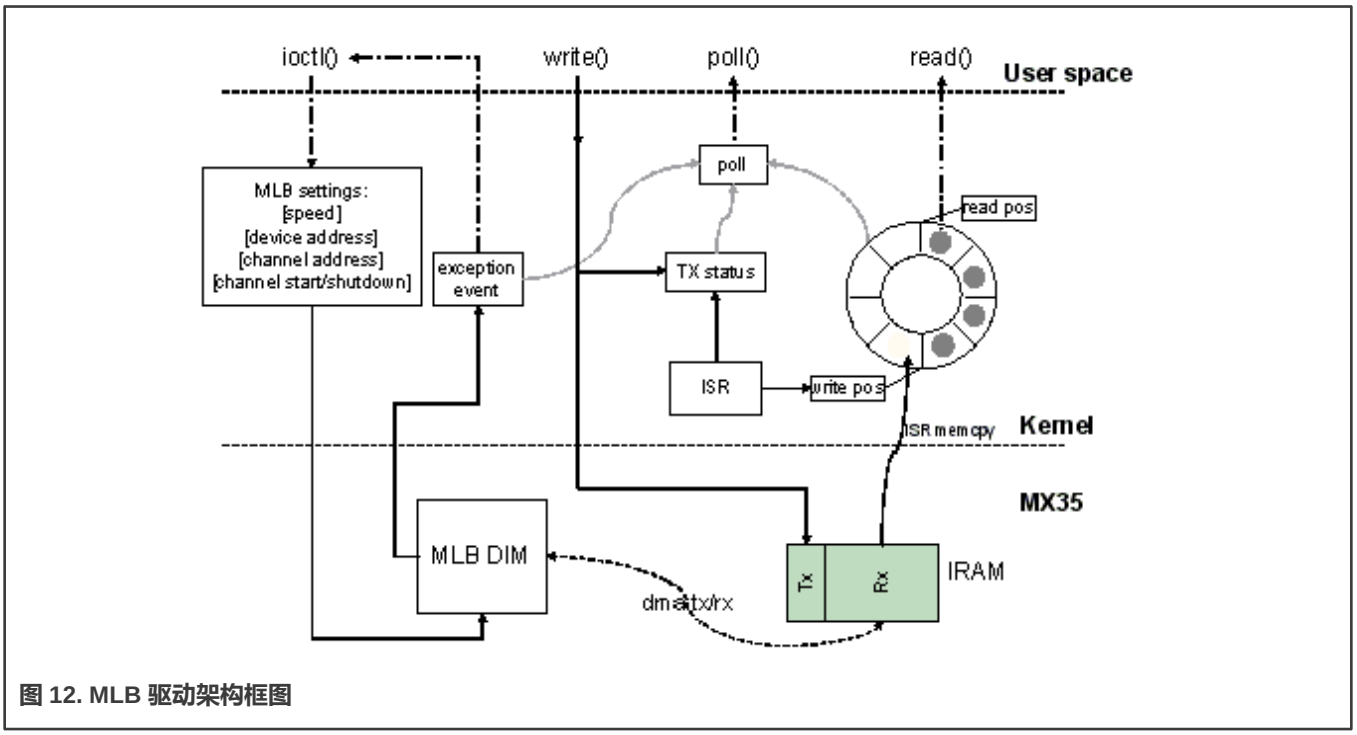


图 12. MLB 驱动架构框图

MLB 驱动创建 4 个次要设备。这 4 个设备支持控制 Tx/Rx 通道、异步 Tx/Rx 通道、同步 Tx/Rx 通道和等时 Tx/Rx 通道。它们的设备文件分别是/dev/ctrl、/dev/async、/dev/sync 和/dev/isoc。每个次要设备具有相同的接口，并处理发送和接收操作。以下描述适用于控制和异步设备。

驱动将 IRAM 作为 MLB 设备模块 Tx/Rx 缓冲区。模块和 IRAM 之间的所有数据传输和接收由 MLB 模块 DMA 处理。驱动负责配置 MLB 模块的缓冲区开始和结束指针。

对于接收，该驱动使用环形缓冲区来缓冲接收到的数据包以供读取。当数据包到达时，MLB 模块将接收到的数据包放入 IRAM Rx 缓冲区，并通过中断通知驱动。然后，驱动将数据包从 IRAM 复制到写入位置指示的一个环形缓冲区节点，并用下一个空节点更新写入位置。最后，通知数据包 reader 应用，它从环形缓冲区的读取位置指示的节点获取一个数据包。读取完成后，它会用下一个可用的缓冲区节点更新读取位置。当读写位置相同时，环形缓冲区中没有接收到的数据包。

对于传输，驱动将 writer 应用给出的数据包写入 IRAM Tx 缓冲区，更新 Tx 状态，并将 MLB 设备模块 Tx 缓冲区指针设置为开始传输。传输完成后，驱动会收到中断通知，并更新 Tx 状态，以接受来自应用的下一个数据包。

驱动支持 NON BLOCK I/O。用户应用可进行轮询，查看是否有要读取的数据包或异常事件，还可以查看是否可以发送数据包。如果存在异常事件，应用可以调用 ioctl 来获取该事件。ioctl 还提供了配置帧速率、设备地址和通道地址的接口。

4.7.3 软件操作

MLB 驱动为应用提供了一个通用接口。

- 数据包读/写：支持 BLOCK 和 NONBLOCK 数据包 I/O 模式。一次只能读取或写入一个数据包。最小读取长度必须大于或等于接收数据包长度，同时写入长度必须小于 1024 字节。
- 轮询：MLB 驱动提供轮询接口，轮询 3 种状态，应用可以使用 select 获取当前 I/O 状态：
  - 数据包可供读取（ready to read）
  - 驱动准备好发送下一个数据包（ready to write）
  - 发生了异常事件（ready to read）
- ioctl-MLB 驱动提供以下 ioctl：

```
MLB_SET_FPS
```

参数类型：unsigned int

设置帧速率，参数必须为 256、512 或 1024。

```
MLB_GET_VER
```

参数类型：unsigned long

获取 MLB 设备模块版本，在 i.MX35 上默认为 0x02000202。

```
MLB_SET_DEVADDR
```

参数类型：unsigned char

设置 MLB 设备地址，供系统通道 MlbScan 命令使用。

```
MLB_CHAN_SETADDR
```

参数类型：unsigned int

设置相应的通道地址[8:1]位。这个 ioctl 结合了 tx 和 rx 通道地址，参数格式为：tx_ca[8:1] << 16 | rx_ca[8:1]。

```
MLB_CHAN_STARTUP
```

启动相应类型的通道进行发送和接收。

```
MLB_CHAN_SHUTDOWN
```

关闭相应类型的通道。

```
MLB_CHAN_GETEVENT
```

参数类型：unsigned long

从 MLB 设备模块获取异常事件，该事件定义为一组枚举：

```
MLB_EVT_TX_PROTO_ERR_CUR
MLB_EVT_TX_BRK_DETECT_CUR
MLB_EVT_RX_PROTO_ERR_CUR
MLB_EVT_RX_BRK_DETECT_CUR
```

4.7.4 源代码结构

下表列出了 MLB 驱动源文件。

表 44. MLB 驱动源文件

文件	说明
drivers/mxc/mlb/mxc_mlb.c	MLB 驱动的源文件
include/linux/mxc_mlb.h	MLB 驱动的 Include 文件

4.7.5 菜单配置选项

在菜单配置中启用以下模块：

Device Drivers > MXC support drivers > MXC Media Local Bus Driver > MLB support.

4.8 PCI Express 根复合体

4.8.1 介绍

i.MX SoC 中包含的 PCI Express 硬件模块可以配置为根复合体或 PCIe 端点。本文档介绍了 i.MX SoC 系列上的 PCI Express 根复合体实施。它还介绍了需要在 i.MX PCI Express 设备上作为根复合体进行配置和操作的驱动。

PCI Express（PCIe）是第三代 I/O 互连，用于低成本、大容量、多平台互连。它具有早期 PCI 和 PCI-X 的概念，并为现有 PCI 软件提供向后兼容性，但有以下区别：

- PCIe 是一种点对点互连
- 设备之间的串行链接
- 基于数据包的通信
- 通过从 X1 到 X16 的聚合通道提供可扩展的性能
- 需要 PCIe 交换机才能在两个以上的 PCIe 设备之间建立连接

4.8.2 术语和凡例

本文档中使用了以下术语和凡例：

- 桥接

  一种功能，可将 PCI/PCI-X 段或 PCI Express 端口与内部组件或与另一个 PCI/PCI-X 总线段或 PCI Express 端口进行虚拟或实际连接。

- 下行
  - 距离根复合体较远的互连/系统元件（端口/组件）的相对位置。交换机上非上行端口的端口都是下行端口。根复合体上的所有端口都是下行端口。链路上的下行组件是距离根复合体较远的组件。

— 信息从根复合体流出的信息流动方向。

- 端点

  几个已定义的系统元件之一。具有 Type 00h Configuration Space 头的函数。

- 主机

  由一个（或多个）中央处理单元（CPU）和资源组成的实体，例如存储器（RAM），可在通过根复合体连接的多个 PCIe 节点之间共享。

- 通道

  两个差分信号对，一对用于传输，一对用于接收。

- 链路

  两个端口及其互连通道的集合。链路是两个组件之间的双单工通信路径。

- PCIe 结构

  由各种 PCI Express 节点（也称为器件）组成的拓扑。结构中的器件可以是根复合体、端点、PCIe-PCI/PCI-X 桥或交换机。

- 端口

  — 在逻辑层面，是组件和 PCI Express 链路之间的接口。

  — 在物理层面，是位于定义链路的同一芯片上的一组发送器和接收器。

- 根复合体

  RC：定义的系统元件，包括主桥、零个或多个根复合体集成端点、零个或多个根复合体事件收集器以及一个或多个根端口。

- 根端口

  根复合体上的 PCI Express 端口，通过关联的虚拟 PCI-PCI 桥映射分层架构的一部分。

- 上行

  — 更接近根复合体的互连/系统元件（端口/组件）的相对位置。在拓扑上最接近根复合体的交换机上的端口是上行端口。仅包含端点或桥接功能的组件上的端口是上行端口。链路上的上行组件更接近根复合体。

结构中相对接近 RC 的任何元件都被视为“上行”。所有靠近 RC 的 PCIe 端点端口（包括网桥的端点）和交换机端口在该设备上称为上行端口。上行流是指向 RC 移动的通信。

4.8.3 i.MX 上的 PCIe 拓扑

MX 上有一个 PCIe 端口。目前，Linux BSP 中只启用了 RC 模式。

下图展示了 i.MX 上的 PCIe RC 端口框图

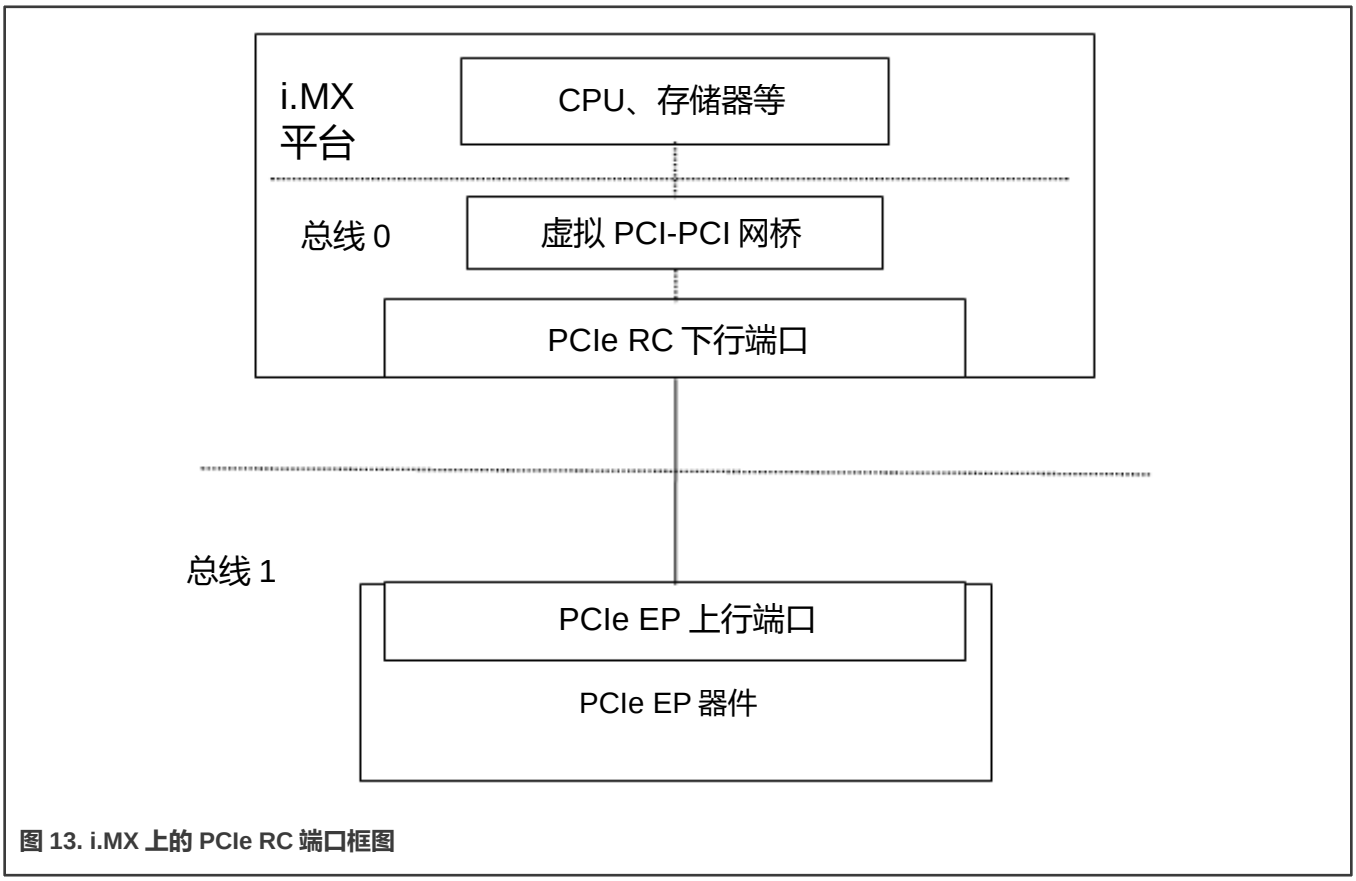


图 13. i.MX 上的 PCIe RC 端口框图

PCI 枚举映射

由于 PCI Express 是点对点拓扑，为了保持与遗留 PCI 总线（用于软件枚举的设备概念）的兼容性，我们引入了以下概念，这些概念允许根据 PCI 设备/功能识别各种节点及其内部器件（例如 PCIe 交换机）：

- 主网桥：集成到 RC 中的网桥，与主机具有 PCI 兼容的连接。该网桥的 PCI 端始终是总线 0。这意味着，该总线上的设备将是主机。
- 虚拟 PCI-PCI 桥：作为 RC 或交换机一部分的每个 PCI Express 端口都被视为一个虚拟 PCI-PCI 桥。这意味着每个端口都有主用和备用 PCI 总线，并且下行被映射到远程配置空间。
- 与根端口相关的虚拟网桥在主用侧有总线 0，在下行有备用总线。
- 每个 PCIe 交换机被视为与下行端口数量一样多的虚拟 PCI-PCI 桥的集合，连接到虚拟 PCI 总线，该总线实际上是交换机上行端口的另一个 PCI-PCI 桥的备用总线。
- 每个 EP 的上行端口可以是虚拟 PCI-PCI 桥的备用总线段的一部分，代表交换机的下行端口或根端口。

4.8.4 特性

以下是 i.MX 作为 PCI Express 根复合体驱动支持的各种功能。

- 符合 Express Base Specification R2.0。
- Gen2 操作，1 个链路支持单向 5 GT/s 的原始传输速率。
- 支持遗留中断（INTx）和 MSI。
- Max_Payload_Size 大小（128 字节）。

- 它适合 Linux PCI 总线框架，支持 PCI 兼容的软件枚举。
- 此外，它为端点驱动提供了接口，用于访问下行检测到的相应设备。
- Linux 操作系统中的 PCI Express 端口总线驱动框架可以使用该接口来处理 AER、ASP 等。
- EP 驱动的中断处理工具可作为遗留中断（INTx）。
- 通过 Linux PCI 子系统中的通用 I/O 附件访问 EP I/O BAR。
- 无缝处理 PCIe 错误。
- 支持 L0、L0s、L1 和 L1.1 ASPM 电源管理。

4.8.5 Linux OS PCI 子系统和 RC 驱动

在 Linux 操作系统中，PCI 实施大致可分为以下主要组件：PCI BIOS 架构特定的 Linux 操作系统实施、主控制器（RC）模块和核心。

- PCI BIOS 架构特定的 Linux 操作系统实施，用于启动 PCI 总线初始化。它与 PCI 主控制器代码以及 PCI 核心对接，以执行总线枚举和资源分配，例如存储器和中断。成功完成 BIOS 执行可确保向系统中的所有 PCI 设备分配了可用 PCI 资源的一部分及其各自的驱动（称为从驱动）。PCI 可以使用 PCI 核心提供的工具进行控制。可以跳过资源分配（如果这些资源是在 Linux 操作系统启动之前分配的，例如 PC 场景）。
- 主控制器（RC）模块处理硬件（SoC + B 电路板）特定的初始化和配置，并调用 PCI BIOS。它应该为 BIOS 和 PCI 核心提供回调函数，这些回调函数将在 PCI 系统初始化和访问 PCI 总线的配置周期期间被调用。它提供可用存储器/IO 空间、INTx 中断线、MSI 的资源信息。它还应该通过 in _x_ () out _x_ () 促进 IO 空间访问（如支持）。可能需要通过 read _x_ () write _x_ () 提供间接存储器访问（如果硬件支持）。
- 核心为系统中的总线设备和网桥创建并初始化数据结构树，处理总线/设备编号，创建设备条目和 proc/sysfs 信息，为 BIOS 和从驱动提供服务，并支持热插拔（可选，取决于硬件是否支持）。它以（EP）驱动接口查询为目标，并对枚举期间找到的相应设备进行初始化。它还提供 MSI 中断处理框架和 PCI Express 端口总线支持。它支持热插拔（如支持）、高级错误报告、电源管理事件支持以及在 PCI Express 端口上运行的虚拟通道（如支持）。

4.8.6 PCIe 驱动源文件

表 45. 源文件

文件	说明
drivers/pci/controller/dwc/pci-imx6.c	i.MX 6 PCIe 源文件

4.8.7 系统资源：存储器布局

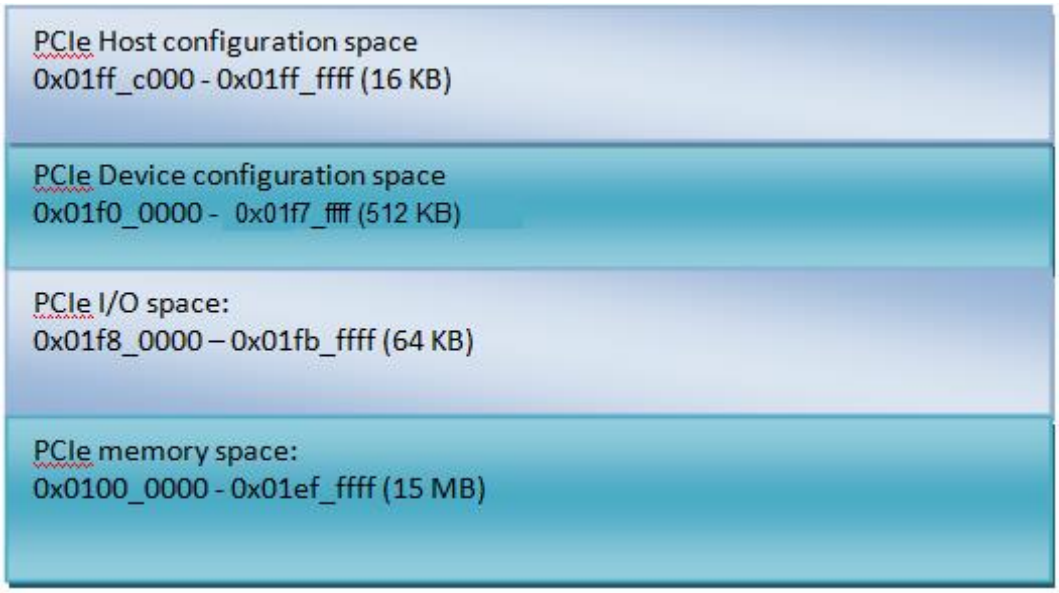


图 14. 存储器布局（i.MX 6Quad/6DualLite/6Solo）

图 15. 存储器布局（i.MX 6SoloX）

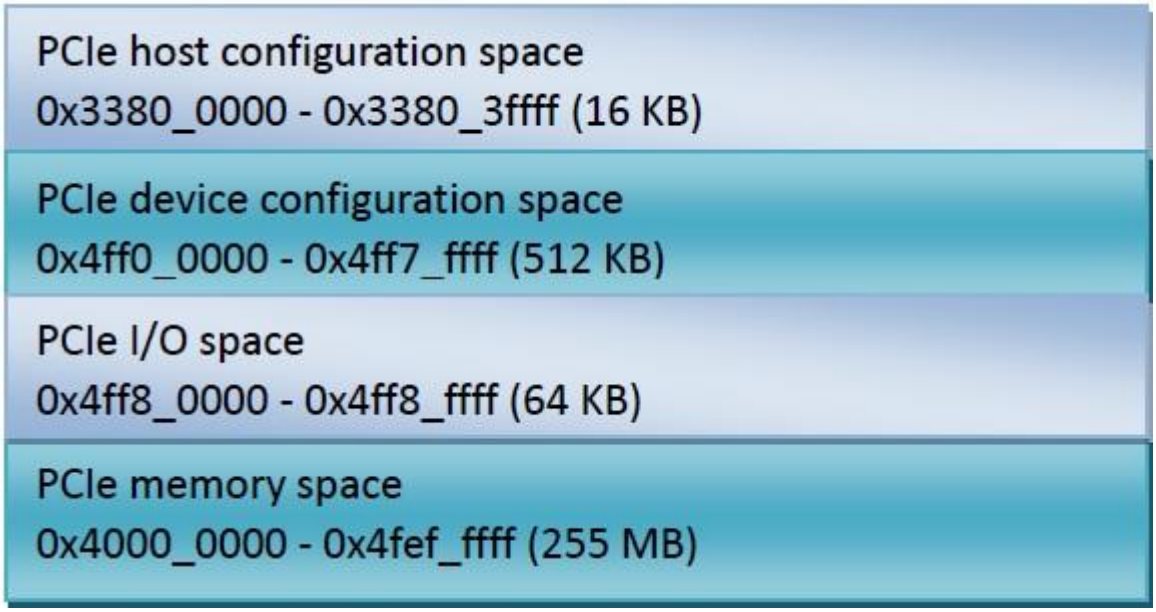


图 16. 存储器布局（i.MX 7Dual）

- IO 和存储器空间是两个地址空间，供设备与在 CPU 上的 Linux 内核中运行的设备驱动进行通信使用。
- 上部 16 KB PCIe 主机配置空间。
  - 此存储器段用于映射 PCIe RC 的配置空间。软件可以通过 DBI 接口访问 PCIe RC 核心配置空间。
- PCIe 设备配置空间。
  - 用于映射插入到 RC 下行端口的 PCIe EP 设备的配置空间。

i.MX 8QuadMax/8QuadXPlus：

i.MX 8QuadMax 有 PCIeA 和 PCIeB，而 i.MX 8QuadXPlus 只有 PCIeB。

- PCIeA
  - PCIe 主机配置空间：0x5f00_0000 – 0x5f00_ffff（64K 字节）
  - PCIe 设备配置空间：0x6ff0_0000 – 0x6ff7_ffff（512K 字节）
  - PCIe IO 空间：0x6ff8_0000 – 0x6ff8_ffff（64K 字节）
  - PCIe 存储器空间：0x6000_0000 – 0x6fef_ffff（255M 字节）
- PCIeB
  - PCIe 主机配置空间：0x5f01_0000 – 0x5f01_ffff（64K 字节）
  - PCIe 设备配置空间：0x7ff0_0000 – 0x7ff7_ffff（512K 字节）
  - PCIe IO 空间：0x7ff8_0000 – 0x7ff8_ffff（64K 字节）
  - PCIe 存储器空间：0x7000_0000 – 0x7fef_ffff（255M 字节）

i.MX 8M Quad:

- PCIe0
  - PCIe 主机配置空间：0x3380_0000 – 0x33bf_ffff (4MB)
  - PCIe 设备配置空间：0x1ff0_0000 – 0x1ff7_ffff（512K 字节）
  - PCIe IO 空间：0x1ff8_0000 – 0x1ff8_ffff（64K 字节）
  - PCIe 存储器空间：0x1800_0000 – 0x1fef_ffff（127M 字节）

- PCIE1
  - PCIe 主机配置空间：0x33c0_0000 – 0x33ff_ffff (4MB)
  - PCIe 设备配置空间：0x27f0_0000 – 0x27f7_ffff（512K 字节）
  - PCIe IO 空间：0x27f8_0000 – 0x27f8_ffff（64K 字节）
  - PCIe 存储器空间：0x2000_0000 – 0x27ef_ffff（127M 字节）

4.8.8 系统资源：中断线

i.MX Root Complex 驱动在 i.MX 6 平台上对 MSI INT 使用中断线 152，在 i.MX 7Dual 平台上对 MSI INT 使用 154。

4.9 USB

4.9.1 介绍

通用串行总线（USB）驱动实施了与 CHIPIDEA USB-HS OTG 控制器对接的标准 Linux 驱动接口。

USB 提供了一个通用链接，可用于各种 PC 到外设的互连。它支持即插即用、端口扩展以及任何使用相同类型端口的新 USB 外设。

CHIPIDEA USB 控制器与增强型主控制器接口（EHCI）兼容。此 USB 驱动具有以下功能：

- 支持高速 OTG 核心
- 支持高速仅主核心（Host1）、高速、全速和低速设备
- 高速芯片间核心（Host2 & Host3）
- 支持高速仅主核心（OTG2）、高速、全速和低速设备。默认情况下，USB2Pci 网桥连接到 OTG2。因此，用户可能无法在此端口上连接其他 USB 设备。
- 高速芯片间核心（Host2）
- 主机模式——支持 HID（人机接口设备）、MSC（大容量存储类）
- 外设模式——支持 MSC 和 CDC（通信设备类）驱动，包括以太网和串行支持
- 嵌入式 DMA 控制器

4.9.2 架构概述

USB 主系统由多个硬件和软件层组成。

下图展示了支持 USB 2.0 的主系统中构建块层的概念框图。

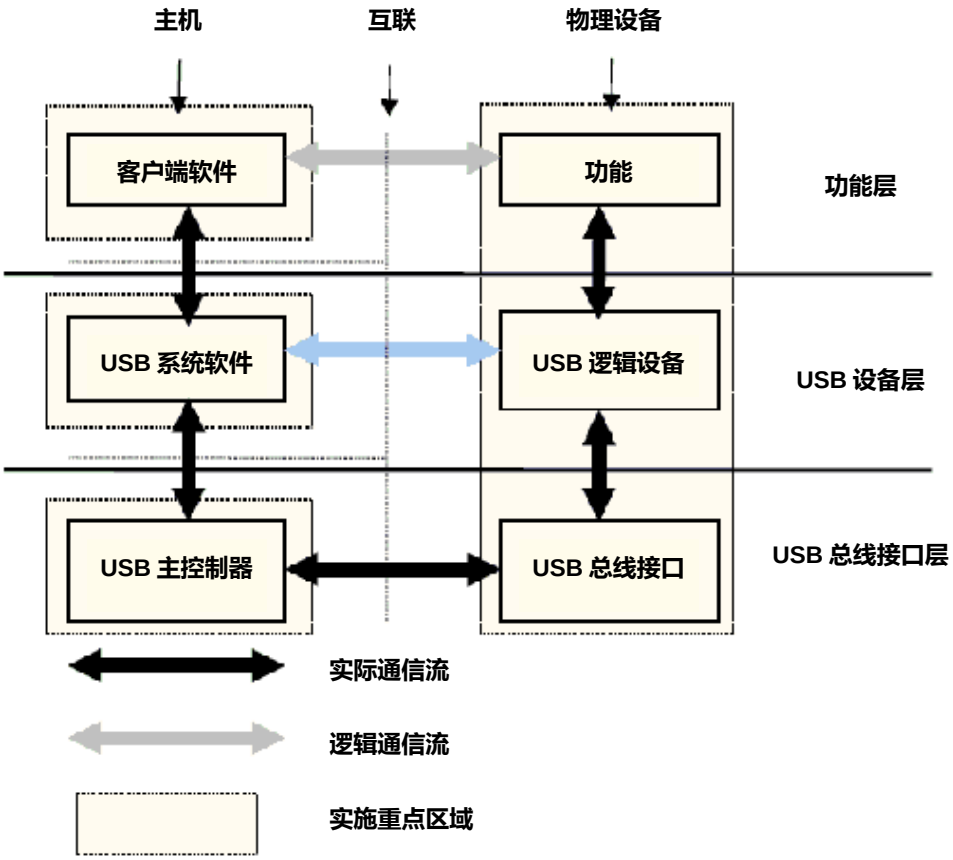


图 17. USB 框图

4.9.3 硬件操作

如需了解硬件操作信息，请参见 EHCI spec.ehci-r10.pdf。

该规范详见《USB 2.0 增强型主控制器接口规范》。

4.9.4 软件操作

Linux 操作系统包含一个实施 USB 协议的 USB 驱动。对于 USB 主机，它只实施硬件指定的初始化功能。对于 USB 外设，它实施设备框架。对于 OTG，支持 ID 动态切换主机/设备模式。目前支持 USB 运行时挂起，即当 USB 不使用时（主机和外设模式），USB 将进入低功耗模式。

4.9.5 源代码结构

下表介绍了 drivers/usb 中的 USB 源文件。

表 46. Chipidea USB 驱动文件

文件	说明
drivers/usb/chipidea/core.c	Chipidea IP 核心驱动

下页继续……

表 46. Chipidea USB 驱动文件（续）

文件	说明
drivers/usb/chipidea/udc.c	Chipidea 外设驱动
drivers/usb/chipidea/host.c	Chipidea 主驱动
drivers/usb/chipidea/otg.c	Chipidea OTG 驱动
drivers/usb/chipidea/otg_fsm.c	Chipidea OTG HNP 和 SRP 驱动
drivers/usb/chipidea/ci_hdrc_imx.c	i.MX 胶层
drivers/usb/chipidea/usbmisc_imx.c	i.MX SoC 抽象层
drivers/usb/phy/phy-mxs-usb.c	i.MX 6 USB 物理驱动

4.9.6 菜单配置选项

在菜单配置中启用以下模块。

Device Drivers > [*] USB support > EHCI HCD (USB 2.0) support and ChipIdea Highspeed Dual Role Controller [*] USB Physical Layer drivers --->

Device Drivers > USB Physical Layer drivers > Freescale MXS USB PHY support

Device Drivers > USB Gadget Support

1. CONFIG_USB：构建主机侧 USB 支持
2. CONFIG_USB_EHCI_HCD EHCI HCD (USB 2.0) support

   默认 y
3. CONFIG_USB_CHIPIDEA：ChipIdea 高速两用控制器

   默认 y
4. CONFIG_USB_CHIPIDEA_UDC：ChipIdea 设备控制器

   默认 y
5. CONFIG_USB_CHIPIDEA_HOST：ChipIdea 主控制器

   默认 y
6. CONFIG_USB_GADGET：USB 设备支持

   默认 y
7. CONFIG_USB_MXS_PHY：恩智浦 MXS USB PHY 支持

   默认值 y

4.9.7 USB 唤醒使用

以下示例适用于 OTG 端口和第一台 EHCI 设备。

控制器唤醒设置，在完成以下设置之后，VBUS 和 ID 将成为唤醒源。

```
echo enabled > /sys/bus/platform/devices/20c9000.usbphy/power/wakeup
echo enabled > /sys/bus/platform/devices/2184000.usb/power/wakeup
echo enabled > /sys/bus/platform/devices/ci_hdrc.0/power/wakeup
```

EHCI 唤醒设置，完成以下设置后，主机将具有唤醒能力，如远程唤醒和连接/断开唤醒

```
echo enabled > /sys/bus/usb/devices/usb1/power/wakeup
echo enabled > /sys/bus/usb/devices/1-1/power/wakeup
```

注意

当 OTG 模式从主机切换到设备时，会删除 EHCI 唤醒，用户需要在系统挂起之前再次进行设置。

4.9.8 如何关闭 USB 子设备电源

以下代码字符串概述了如何关闭 USB 子设备电源：

```
echo auto > /sys/bus/usb/devices/1-1/power/control
echo auto > /sys/bus/usb/devices/1-1.1/power/control (If there is a hub at USB device)
```

4.9.9 更改控制器操作模式

要更改默认设置，用户可以按如下方式修改 DTS 文件：

```
dr_mode =  "host"  /* Set controller as gadget-only mode */
dr_mode =  "peripheral" /* Set controller as host-only mode */
dr_mode =  "otg" /* Set controller as otg mode */
```

4.9.10 可加载模块支持

modprobe 实用程序将自动加载之间具有依赖关系的模块。

加载命令如下所示：

```
modprobe phy_mxs_usb
modprobe ci_hdrc_imx
```

卸载命令如下所示：

```
modprobe -r ci_hdrc_imx
modprobe -r phy_mxs_usb
```

4.9.11 USB 充电器检测

i.MX SoC 具有 USB 充电器检测功能，但没有充电功能。用户可以通过/sys 条目了解 USB 充电器类型、充电电流、以及是否有充电器，如下面的 3 行所示：

```
cat /sys/class/power_supply/imx6_usb_charger/type
cat /sys/class/power_supply/imx6_usb_charger/current_max
cat /sys/class/power_supply/imx6_usb_charger/present
```

目前，i.MX 6 Sabre-SD 电路板不支持 USB 充电器检测功能。i.MX 6 Sabre-Auto 支持该功能。

4.9.12 嵌入式主机认证

4.9.12.1 添加 TPL-Support 属性

要通过嵌入式主 USB 认证，应在 DTS 中添加“tpl-support”，以启用 Targeted Peripheral List (TPL)。例如，要在 i.MX 6UltraLite EVK 板（imx6ul-14x14-EVK.dts）的主端口上启用 TPL：

```
&usbotg2 {
dr_mode = "host";
disable-over-current;
tpl-support;
status = "okay";
};
```

4.9.12.2 VBUS 控制

在 Linux USB 主机功能就绪之前，VBUS 应保持关闭。例如，在 i.MX 6UltraLite EVK 板上，由于引脚复用触摸功能，需要对电路板进行返工，以便选择 GPIO (GPIO1_IO02)用于 VBUS 控制。

在其 DTS 文件（imx6ul-14x14-evk.dts）中禁用触摸功能，如下所示：

```
&tsc {
pinctrl-names = "default";
pinctrl-0 = <&pinctrl_tsc>;
xnur-gpio = <&gpio1 3 0>;
measure_delay_time = <0xffff>;
pre_charge_time = <0xfff>;
status = "disabled";
};
```

添加 VBUS GPIO pinctrl 及其稳压器节点：

```
pinctrl_usb_otg2: usbotg2grp
          { fsl,pins = <
              MX6UL_PAD_GPIO1_IO02  GPIO1_IO02        0xb0
          >;
};
reg_usb_otg2_vbus: regulator@2 {
          compatible = "regulator-fixed";
          reg = <2>;
          pinctrl-names = "default";
          pinctrl-0 = <&pinctrl_usb_otg2>;
          regulator-name = "usb_otg2_vbus";
          regulator-min-microvolt = <5000000>;
          regulator-max-microvolt = <5000000>;
          gpio = <&gpio1 2 GPIO_ACTIVE_HIGH>;
          enable-active-high;
};
&usbotg2 {
  vbus-supply = <&reg_usb_otg2_vbus>;
  dr_mode = "host";
  disable-over-current;
  tpl-support;
  status = "okay";
};
```

4.10 USB3

4.10.1 介绍

对于 i.MX 8 和 i.MX 8X 系列，Cadence 提供了一个支持 USB 3.0 的超高速 USB IP，其中包括一个称为超高速（SS）USB 的新传输速率，传输速率更高，比 USB 2.0 标准快得多。

支持的功能如下所示：

- 主机模式采用 Linux OS 标准 XHCI 驱动实现，支持超高速并经过了测试。
- 对于设备模式，仅支持单个队列。支持大容量存储、ether 和串行。

4.10.2 源代码结构

表 47. USB3 驱动源文件

文件	说明
drivers/usb3/cdns3/cdns3-nxp-reg-def.h	寄存器定义
drivers/usb3/cdns3/core.c	USB3 核心驱动
drivers/usb3/cdns3/core.h	USB3 核心头文件
drivers/usb3/cdns3/dev-regs-macro.h	USB3 Macros
drivers/usb3/cdns3/dev-regs-map.h	USB3 寄存器映射
drivers/usb3/cdns3/gadget.c	USB3 设备
drivers/usb3/cdns3/gadget.h	USB3 设备头文件
drivers/usb3/cdns3/gadget-export.h	USB3 设备导出头文件
drivers/usb3/cdns3/host.c	USB3 主机
drivers/usb3/cdns3/host-export.h	USB3 主机导出头文件
drivers/usb3/cdns3/io.h	USB3 IO

4.11 LPUART

4.11.1 介绍

低级 UART 驱动将 Linux 串行驱动 API 连接到所有 UART 端口。

它具有以下特性：

- 中断驱动和 eDMA 驱动的字符发送/接收
- 标准 Linux 波特率高达 4 Mbps
- 发送和接收长度为 7 位、8 位、9 位或 10 位的字符
- 传输一个或两个停止位
- 支持 TIOCMGET IOCTL 来读取调制解调器控制线。仅在 DTE 模式下支持常量 TIOCM_CTS 和 TIOCM_CAR，加上 TIOCM_RI
- 支持 TIOCMSET IOCTL 设置调制解调器控制线。仅支持常量 TIOCM_RTS 和 TIOCM_DTR
- 奇偶校验
- XON/XOFF 软件流控制。当通信速度不太高且缓冲区溢出的可能性很小时，使用软件流控制的串行通信是可靠的

- CTS/RTS 硬件流控制——中断驱动的软件控制的硬件流和硬件驱动的硬件控制流
- 通过标准 Linux 串行 API 发送和接收中断字符
- 识别帧和奇偶校验错误
- 能够忽略带有中断、奇偶校验和帧错误的字符
- 通过 TIOCGSSERIAL 和 TIOCSSERIAL TTY IOCTL 获取和设置 UART 端口信息。一些程序（如 setserial 和 dip）使用此功能来确保正确设置波特率并获取设备的一般信息。UART 类型应设置为 serial_core.h 头文件中定义的 52。
- 通过暂停和恢复 UART 端口的电源管理功能
- 标准 TTY 层 IOCTL 调用

可以从 device files /dev/ttyLP0 to /dev/ttyLP1 访问所有 UART 端口。

4.11.2 硬件操作

要确定设备上可用的 UART 模块数量，请参见与 SoC 相关的《应用处理器参考手册》。

每个 UART 硬件端口都能够进行标准 RS-232 串行通信。

每个 UART 包含一个 64 字节的发送器 FIFO 和一个 32 半字深的接收器 FIFO。当每个 FIFO 中的数据电平达到编程电平阈值且调制解调器信号的状态发生变化时，每个 UART 还支持各种可屏蔽中断。

4.11.3 软件操作

Linux 操作系统包含一个核心 UART 驱动，用于管理各种平台的 UART 驱动中常见的许多串行操作。

低级 UART 驱动负责向核心 UART 驱动提供 UART 端口信息和一组控制函数。这些函数被用作 Linux OS 和 UART 硬件之间的低级接口。不能从其他驱动或用户应用调用它们。用于控制硬件的控制函数通过名为 uart_ops 的结构传递给核心驱动，端口信息通过名为 uart_port 的结构传递。低级驱动还负责处理 UART 端口的各种中断，并在必要时提供控制台支持。

通过启用 DTS 文件中的 DMA 通道，可以将每个 UART 配置为使用 DMA 进行数据传输。

驱动为需要 DMA 传输的 UART 请求两个 DMA 通道。在接收事务中，驱动将数据从 DMA 接收缓冲区复制到 TTY 翻转缓冲区。

在使用 DMA 进行传输时，驱动将数据从 UART 传输缓冲区复制到 DMA 传输缓冲区，并将该缓冲区发送到 DMA 系统。如需了解更多信息，请参见内核源代码树中有关串行驱动的 Linux 文档。

4.11.4 驱动特性

UART 驱动支持以下功能：

- 波特率高达 4 Mbps
- 仅在中断驱动模式下识别帧和奇偶校验错误；在 DMA 驱动模式下无法识别这些错误
- 发送、接收和适当处理中断字符
- 识别调制解调器控制信号
- 如果要求，可忽略带有帧、奇偶校验和中断错误的字符
- 支持硬件流控制

- 获取和设置 UART 端口信息；某些流控计数信息在硬件驱动的硬件流控模式下不可用
- 电源管理
- 中断驱动和 DMA 驱动的数据传输

4.11.5 源代码结构

下表列出了 UART 驱动源文件。

表 48. UART 驱动文件

文件	说明
drivers/tty/serial/fsl_lpuart.c	LP UART 驱动

对于 i.MX 8，i.MX 8X 和 i.MX 8M 配置选项在 arch/arm64/boot/dts 目录下的设备树中指定。

4.11.6 菜单配置选项

UART 驱动默认启用。

菜单配置选项位于：

Device Drivers > Character devices > Serial drivers > Freescale LPUART serial port support [*] Console on Freescale LPUART serial port

4.11.7 编程接口

UART 驱动实施了 Linux 串行 API 与 UART 端口对接所需的所有方法，并为 Linux 核心 UART 驱动提供了一组控制方法。如需了解驱动中实现的方法的更多信息，请参见 API 文档。

4.11.8 中断要求

UART 驱动接口只生成一个中断。

状态用于确定发生哪种中断，如 RX 或 TX。

4.12 蓝牙

4.12.1 蓝牙无线技术介绍

蓝牙技术是一种低成本、低功耗、短距离的无线技术。它的设计目的是替代线缆和 IrDA 等其他短程技术。蓝牙无线技术在个人区域范围内运行，通常可延伸至 10 米。如需了解蓝牙无线技术的更多信息，请访问 www.bluetooth.com/。

对于 i.MX，多家供应商都支持蓝牙。如需了解详细信息，请参见《*i.MX Linux® 用户指南*》（IMXLUG）中的“蓝牙无线技术和 Wi-Fi 连接”章节。

4.12.2 蓝牙驱动概述

i.MX 使用开源蓝牙驱动。蓝牙软件分为以下 4 个部分：

- 4 线 UART 和 TTY 驱动：它是与蓝牙模块进行通信的接口。
- 蓝牙 HCI 设备驱动：UART (H4) 是用于蓝牙设备和主机之间通信的串行协议。大多数带有 UART 接口的蓝牙设备都需要此协议。
- 蓝牙内核堆栈：蓝牙框架和协议实施。
- 蓝牙用户堆栈：提供多个用户空间实用程序，并为用例集成多个配置文件。

4.12.3 蓝牙驱动文件

蓝牙驱动源文件位于内核源文件目录中。

- 蓝牙 HCI 设备驱动：
  - drivers/bluetooth/hci_h4.c
  - drivers/bluetooth/hci_ldisc.c
- 蓝牙内核堆栈：
  - net/bluetooth/*

4.12.4 蓝牙协议栈

BlueZ 是官方的 Linux 标准蓝牙协议栈，5.x 的最新版本，是基于 Linux 内核的操作系统系列的蓝牙协议栈。其目标是为 Linux 编写蓝牙无线标准规范实施。要使用 Linux 蓝牙子系统，需要几个用户空间实用程序，例如 hciconfig 和 bluetoothd。BlueZ 软件包中提供了这些实用程序和蓝牙内核模块的更新。如需了解详细信息，请访问 www.bluez.org/。

BlueZ 源代码位于 git: git://git.kernel.org/pub/scm/bluetooth/bluez.git。当前的 BSP 包通过了 BlueZ 5.56 测试。

4.12.5 菜单配置选项

为该模块提供了以下 Linux 内核配置选项：

- UART 接口：
  - CONFIG_SERIAL_IMX
  - CONFIG_TTY
- HCI 接口：
  - CONFIG_BT_HCIUART
  - CONFIG_BT_HCIUART_H4
  - CONFIG_BT_HCIUART_BCM
- 蓝牙协议栈
  - CONFIG_BT
  - CONFIG_BT_RFCOMM
  - CONFIG_BT_RFCOMM_TTY
  - CONFIG_BT_BNEP
  - CONFIG_BT_BNEP_MC_FILTER
  - CONFIG_BT_BNEP_PROTO_FILTER
  - CONFIG_BT_HIDP

4.13 Wi-Fi

4.13.1 介绍

i.MX 通过板载芯片解决方案和外部硬件支持蓝牙和 Wi-Fi。如需了解各种板载芯片和外部解决方案，请参见《*i.MX Linux®用户指南*》（IMXLUG）中的“蓝牙无线技术和 Wi-Fi 连接”章节。

4.13.2 软件操作

BSP 支持：

- 从 5.4.47-2.2.0 版开始，Linux BSP 中可用的所有 i.MX 芯片组都支持恩智浦 Wi-Fi 驱动模块。如需获得支持的 Wi-Fi 芯片组清单，请参见每个 i.MX Linux BSP 版本的版本说明。

4.13.3 驱动特性

恩智浦 Wi-Fi 驱动支持 CFG80211 和 NL80211 内核接口。该驱动支持 AP 模式、STA 模式、Wi-Fi 直连模式。

恩智浦 Wi-Fi SoC 需要在上电/复位时加载固件映像。支持的 Wi-Fi SoC 的固件映像位于以下 rootfs 目录：/lib/firmware/nxp。

4.13.4 源代码结构

恩智浦 Onedriver 源代码文件可从 codeaurora.org 获得。

4.13.5 菜单配置选项

为该模块提供了以下 Linux 内核配置选项：

- CONFIG_MAC80211=y
- COCONFIG_NL80211_TESTMODE=y
- CONFIG_CFG80211_WEXT=y
- CONFIG_HOSTAP=y
- CONFIG_CFG80211_INTERNAL_REGDB=y

4.13.6 从用户空间配置 WLAN

在 Station 模式下连接 AP

以下命令组用于将 WLAN 连接到给定的 SSID。

```
modprobe moal mod_para=nxp/wifi_mod_para.conf
head -n 4 /etc/wpa_supplicant.conf  > /etc/wpa_supplicant.conf.tmp
wpa_passphrase ssid password >> /etc/wpa_supplicant.conf.tmp
mv /etc/wpa_supplicant.conf  /etc/wpa_supplicant.conf.bak
mv /etc/wpa_supplicant.conf.tmp  /etc/wpa_supplicant.conf
wpa_supplicant -B -i mlan0 -c /etc/wpa_supplicant.conf -D nl80211
```

以下是一个 wpa_supplicant.conf 示例：

```
ctrl_interface=/var/run/wpa_supplicant
ctrl_interface_group=0
update_config=1
network={
  ssid="NETGEAR73"
  #psk="freshbutter"
  psk=eb0376fc14ee5d1e6ce129ad54da038adab……
}
```

获取 IP 地址

以下命令用于获取 wlan0 的 IP 地址：

```
udhcpc -i mlan0
```

第 5 章
图形

5.1 图形处理单元（GPU）

5.1.1 介绍

图形处理单元（GPU）是一种面向嵌入式 2D/3D 图形应用的图形加速器。3D 图形处理单元（GPU3D）是一个嵌入式引擎，可加速用户级图形应用编程接口（API），如 OpenGL ES 1.1、OpenGL ES 2.0、OpenGL ES 3.0 和 OpenCL 1.1EP。2D 图形处理单元（GPU2D）是一款嵌入式 2D 图形加速器，旨在提高图形用户界面 (GUI) 的渲染能力。VG 图形处理单元（GPUVG）是一个嵌入式矢量图形加速器，用于支持 OpenVG 1.1 图形 API 和功能集。GPU 驱动内核模块源代码位于内核源代码树中，但库仅以二进制形式提供。

图形处理单元	硬件	适用平台
3D	Vivante dual-GC7000XSVX	8QuadMax
3D	Vivante GC7000Lite	8QuadXPlus/8M Quad
3D	Vivante GC7000 Nano Ultra	7ULP 和 8M Mini
3D	Vivante GC7000 UltraLite	8M Plus
3D	Vivante GC7000 Ultra Lite	8M Nano
3D	Vivante GC2000	6Quad/6Dual
3D	Vivante GC2000+	6QuadPlus/6DualPlus
3D	Vivante GC880	6DualLite/6Solo
3D/2D	Vivante GC400T	6SoloX
2D	Vivante GC320	6Quad/6Dual/6DualLite/6Solo
Vector	Vivante GC355	6Quad/6Dual
2D	Vivante GC328	7ULP

注意

- GC400T 不支持 OpenGL ES 3.0。
- GC880/GC400T 不支持 OpenCL 1.1EP. GC2000 和 GC2000+ support OpenCL 1.1 EP。
- GC7000XSVX 支持 OpenCL 1.2 FP、OpenVX 1.0.1 和 Vulkan 1.0。

5.1.2 驱动特性

GPU 驱动使该电路板能够提供以下软件和硬件支持：

- Khronos Group 定义的 EGL（EGL 是 OpenGL ES 或 OpenVG 等 Khronos 渲染 API 与底层原生平台窗口系统之间的接口）1.5 API。
- Khronos Group 定义的 OpenGL ES（OpenGL ES 是一种免版税的跨平台 API，用于嵌入式系统上的全功能 2D 和 3D 图形）1.1 API。

- Khronos Group 定义的 OpenGL ES 2.0 API。
- Khronos Group 定义的 OpenGL ES 3.0/3.1/3.2 API。
- Khronos Group 定义的 OpenVG（OpenVG 是一种免版税的跨平台 API，为 Flash 和 SVG 等矢量图库提供低级硬件加速接口）1.1 API。
- Khronos Group 定义的 OpenCL（OpenCL 是第一个开放的、免版税现代处理器跨平台并行编程标准。）1.1 EP API。
- Khronos Group 定义的 OpenGL 2.1 API。
- 自动 3D 核心减速，当来自热驱动的热通知处于活跃状态时，3D 核心将以 1/64 时钟频率运行。
- Khronos Group 定义的 OpenCL1.1/1.2FP API。
- Khronos Group 定义的 OpenVX 1.0.1 API。
- Khronos Group 定义的 Vulkan 1.0 API。

5.1.3 硬件操作

如需了解详细的硬件操作信息，请参见 SoC 特定的《应用处理器参考手册》中的 GPU 章节。

5.1.4 软件操作

GPU 驱动分为两层。第一层在内核模式下运行，作为整个堆栈的基本驱动。这一层提供必要的硬件访问、设备管理、存储器管理、命令队列管理、上下文管理和电源管理。第二层在用户模式下运行，实施协议栈逻辑，为上层应用提供如下 API：

- OpenGL ES 1.1、2.0 和 3.0 API
- EGL 1.5 API
- OpenGL ES11/20/30/31/32
- OpenCL 1.1/1.2 FP
- OpenVX 1.0.1
- Vulkan 1.0
- OpenGL 4.0
- WebGL 1.0.2
- OpenVG 1.1 API
- OpenCL 1.1 EP API

5.1.5 源代码结构

下表列出了 GPU 驱动内核模块源代码结构：

```
drivers/mxc/gpu-viv
```

表 49. GPU 驱动文件

文件	说明
Kconfig Kbuild config	内核配置文件和 makefile

下页继续……

表 49. GPU 驱动文件（续）

文件	说明
hal/kernel/arch	GC2000、GC880、GC400T 和 GC320 的硬件特定驱动代码
hal/kernel/archvg	GC355 的硬件特定驱动代码
hal/kernel	内核模式 HAL 驱动
hal/os/linux/kernel	OS 层 HAL 驱动

注意

如果替换该目录中的全部内容，则可以升级 GPU 内核驱动。

5.1.6 库结构

下表列出了 GPU 驱动用户模式库结构：

<ROOTFS>/usr/lib

表 50. GPU 库文件

文件	说明
libCLC.so	OpenCL 前端编译器库
libEGL.so**	EGL1.4 库
libGAL.so	GAL 用户模式驱动
libGLES_CL.so	OpenGL ES 1.1 通用精简库（没有 EGL API，无浮点支持 API）
libGL.so**	OpenGL 2.1 通用库
libGLES_CM.so	OpenGL ES 1.1 通用库（没有 EGL API，含浮点支持 API）
libGLESv1_CL.so**	OpenGL ES 1.1 通用精简库（有 EGL API，无浮点支持 API）
libGLESv1_CM.so**	OpenGL ES 1.1 通用库（有 EGL API，含浮点支持 API）
libGLESv2.so**	OpenGL ES 2.0/3.0/3.1/3.2 库
libGLSLC.so	OpenGL ES 着色器语言编译库
libVSC.so	OpenGL 前端编译库
libVivanteOpenCL.so	Vivante

下页继续……

表 50. GPU 库文件（续）

文件	说明
libOpenCL.so	OpenCL ICD wrapper 库
libOpenVG.so*	OpenVG 1.1 库
libVDK.so	VDK wrapper 库。
libVIVANTE.so	Vivante 用户模式驱动。
xorg/modules/drivers/vivante_drv.so	用于 X11 加速的 EXA 库。
libwayland-viv.so	用于 Vivante 的 EGL 驱动的 Wayland 服务器侧库
libgc_wayland_protocol.so	Vivante Wayland 协议扩展库
libOpenVX.so*	OpenVX 1.0 库
libvulkan..so*	Vulkan 1.0 库

**SONAME 用于 libEGL.so、libGLESv2.so、libGLESv1_CM.so、libGLESv1_CL.so、libGL.so。

*对于 libOpenVG.so，OpenVG 功能有两个库。libOpenVG.3d.so 是基于 GC7000XSVX/GC2000+/GC2000/ GC880/GC400T 的 OpenVG 库。libOpenVG.2d.so 是基于 gc355 的 OpenVG 库。

- 对于 i.MX 6DualPlus/QuadPlus 和 i.MX 6Dual/Quad，可以使用 libOpenVG.3d.so 和 libOpenVG.2d.so。
- 对于 i.MX 6DualLite 和 i.MX 6SoloX，只能使用 libOpenVG.3d.so。
- 如果没有 SOC 限制，对于 x11 后端，默认链接 libOpenVG.3d.so。
- 如果没有 SOC 限制，对于 framebuffer、directFB 和 Wayland 后端，默认的 openVG 库链接到 libOpenVG.2d.so。

可使用以下命令序列来完成：

```
cd <ROOTFS>/usr/lib
sudo ln -s libOpenVG_355.so libOpenVG.so
```

5.1.7 API 参考

如需了解详细规范，请访问以下网站：

- OpenGL ES 1.1、2.0 和 3.0 API：www.khronos.org/opengles/
- OpenCL 1.1 EP www.khronos.org/opencl/
- EGL 1.4 API：www.khronos.org/egl/
- OpenVG 1.1 API：www.khronos.org/openvg/
- OpenGL ES API：www.khronos.org/opengles/
- OpenCL API：www.khronos.org/opencl/
- OpenVX API：www.khronos.org/openvx/
- Vulkan API：www.khronos.org/vulkan/
- OpenGL API：www.khronos.org/opengl/
- WebGL API：www.khronos.org/webgl/

5.1.8 菜单配置选项

在菜单配置中为 GPU 驱动启用以下模块：

CONFIG_MXC_GPU_VIV 是 GPU 驱动的配置选项。在 menuconfig 中，此选项位于“Device Drivers > MXC support drivers > MXC Vivante GPU support > MXC Vivante GPU support “。

在显示的屏幕上，选择 Configure the kernel（“配置内核”），选择”Device Drivers > MXC support drivers > MXC Vivante GPU support > MXC Vivante GPU support “，然后退出。出现下一个屏幕时，选择以下选项以启用 GPU 驱动：

- Package list > imx-gpu-viv
- 该软件包提供专有的二进制库，以及从 GPU 构建的帧缓冲测试代码

5.2 Wayland

5.2.1 介绍

Wayland 是一个用于合成器与其客户端进行对话的协议，也是该协议的 C 库实施。合成器可以是在 Linux 内核 modesetting 和 evdev 输入设备上运行的独立显示服务器，也可以作为 X 应用或 Wayland 客户端。客户端可以是传统应用、X 服务器或其他显示服务器。

Wayland 项目的一部分也是 Wayland 合成器的 Weston 参考实现。Weston 合成器是一个最小且快速的合成器，适用于许多嵌入式和移动用例。

本章介绍如何在 i.MX 系列设备上启用 Wayland/Weston 支持。

5.2.2 软件操作

此版本基于 Wayland 1.16 版本和 Weston 5.0.0 版本。

5.2.3 Yocto 构建说明

Yocto Project 构建说明如下：

1. 准备一个 Yocto 构建目录并遵循面向 DISTRO Wayland 的《*i.MX Yocto Project 用户指南*》（IMXLXYOCTOUG）中的设置说明。
2. 在构建目录中为 Wayland 设置 Yocto：

```
$ MACHINE = <your-machine> DISTRO=fsl-imx-xwayland source imx-setup-release.sh -b build-wayland
```

3. 构建镜像

```
$ bitbake imx-image-multimedia
```

5.2.4 自定义 Weston

i.MX Weston 包含两个合成器。一个是 EGL3D 合成器，由 3D 核心加速。另一个是由 2D BLT 引擎加速的 G2D 合成器。

Weston 选项可以在文件“/etc/init.d/weston”中更新。

表 51. Weston 的常用选项

Weston 选项	说明
tty	默认为当前 tty。
device	“/dev/fb0”，默认帧缓冲区，G2D 合成器支持多显示。

下页继续……

表 51. Weston 的常用选项（续）

Weston 选项	说明
use-gl	EGL 加速，默认为“1”。
use-g2d	G2D 加速，默认为“0”。
idle-time	以秒为单位的空闲时间。

Weston 支持的多显示器

仅 G2D 合成器支持多显示。添加这些选项以启动 Weston：

```
weston --tty=1 --device=/dev/fb0,/dev/fb2 --use-g2d=1 &
```

Weston 支持的多缓冲区

Weston 服务器支持单缓冲和多缓冲。在单缓冲中，损坏区域被渲染到屏幕外表面，光点被渲染到前缓冲区。屏幕外表面用于避免闪烁。默认情况下，Weston 服务器以单缓冲启动。

在多缓冲中，不渲染到屏幕外，而是将损坏区域渲染到后缓冲区并进行翻转，但帧速率将限制为显示速率。最多支持 3 个缓冲区。

在启动 Weston 服务器之前，导出 FB_MULTI_BUFFER 以控制要使用的缓冲区数量。

单缓冲的环境变量：

```
export FB_MULTI_BUFFER=1
```

双缓冲的环境变量：

```
export FB_MULTI_BUFFER=2
```

5.2.5 运行 Weston

执行以下操作以运行 Weston：

1. 启动 i.MX 设备。
2. 要运行客户端，顶部栏中的第二个按钮将运行 weston-terminal，您可以从中运行客户端。Weston 构建目录中提供了一些演示客户端，但这些客户端都非常简单，主要用于测试 Wayland 协议中的特定功能：
   - 'weston-terminal' 是一个简单的终端仿真器，不是很兼容，但对于 bash 来说已经足够。
   - 'weston-flower' 在屏幕上画一朵花，测试帧协议。
   - 'weston-smoke' 测试 SHM 缓冲区共享。
   - 'weston-image' 加载通过命令行传递的图像文件并显示。

5.3 X Windows 加速

5.3.1 介绍

X-Windows 系统（又名 X11 或 X）是一种基于客户端-服务器的便携式图形显示系统。仅 i.MX 6 支持 X11。

X-Windows 系统可以使用默认的帧缓冲驱动运行，该驱动处理对主显示器的所有绘图操作。由于有可用的 2D GPU（图形处理单元），可以加速一些绘图操作。高级 X 操作可以分解为低级绘图操作，这些操作为 X-Windows 系统加速。

5.3.2 硬件操作

带有 GPU 的 i.MX 上的 X-Windows 系统利用 Vivante GC320 2D GPU 进行加速。

加速也取决于帧缓冲存储器。

5.3.3 软件操作

X.org X 服务器 1.11.x 版及支持 EXA 接口 2.5 版的更高版本支持 X-Windows 加速。

下表汇总了 X11 加速的操作类型。所有操作都涉及屏幕上或屏幕外的帧缓冲存储器：

- 矩形的实体填充。
- 将系统存储器中的图像上传到视频存储器中。
- 复制具有相同像素格式的矩形，源目标矩形可能重叠。
- 使用以下选项复制支持大多数 XRender 合成操作的矩形：
  - 像素格式转换。
  - 重复模式源。
  - Porter-Duff 混合了源与目标。
  - 源 alpha 屏蔽。

以下列表包括 X-Windows 加速支持的其他功能：

- X 像素图直接分配到帧缓冲存储器中。
- EGL 交换缓冲区，其中 EGL 窗口表面是 X-window。
- X-window 可以合成 X 像素图，可以直接用作任何 EGL 表面。

5.3.4 X-Windows 加速架构

下面的框图展示了 X-Windows 系统加速所涉及的组件：

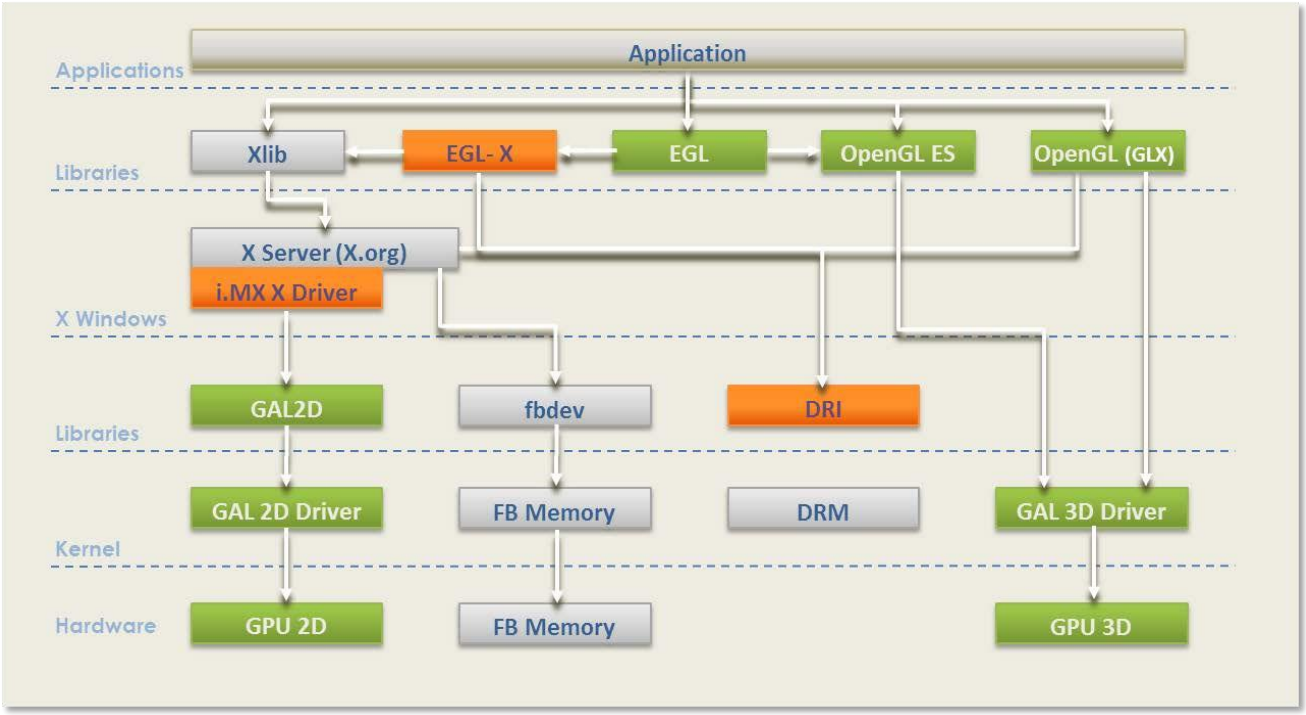


图 18. X 驱动架构

绿色显示的组件是作为 Vivante 2D/3D GPU 驱动支持的一部分提供的组件，包括 OpenGL/ES 和 EGL。浅灰色显示的组件是没有加速的 X-Windows 系统中的标准组件。橙色显示的组件是为支持 X-Windows 系统加速而添加的组件，此处进行了简要介绍。

i.MX X 驱动库模块（`vivante-drv.so`）由 X 服务器加载，包含用于包含 GC320 2D GPU 核心的 i.MX 平台的 X-Windows 加速接口的高级实施。`/dev/fb0` 中的整个线性连续帧缓冲存储器用于在屏幕上和屏幕外为 X 分配像素图。该驱动支持自定义 X 扩展，允许 X 客户端查询存储在帧缓冲存储器中的任何 X 像素图的 GPU 地址。

libGAL.so 库模块（`libGAL.so`）包含与 GC320 GPU 模块对接的寄存器级编程接口。其中包括将寄存器编程命令存储到可以流式传输到设备的数据包中。libGAL.so 库中的函数由 i.MX X 驱动代码调用。

EGL-X 库模块（`libEGL.so`）包含低级别 EGL 平台特定支持功能的 X-Windows 实现。允许将 X 窗口和 X 像素图对象用作 EGL 窗口和像素图表面。EGL-X 库在其实现中使用 Xlib 函数调用以及 i.MX X 驱动模块的 X 扩展来查询存储在帧缓冲存储器中的 X 像素图的 GPU 地址。

5.3.5 X-Windows 系统的 i.MX 驱动

i.MX X 驱动称为 vivante-drv.so，实施 X 服务器的 EXA 接口以提供加速。

Vivante X Driver 简称 vivante-drv.so，实施 X 服务器的 EXA 接口以提供加速。

下面列出了该实施的特定细节：

- 该实施建立在 X 的 fbdev 帧缓冲驱动的源代码之上，因此当加速被禁用时，它可以作为回退。
- 该实施基于 X 服务器 EXA 2.5.0 版。

- EXA 实心填充操作被加速，但包含少于 300x300 像素的源/目标可绘制对象除外，在这种情况下，回退到软件渲染。
- EXA 复制操作会加速，但包含少于 400x120 像素的源/目标可绘制对象除外，在这种情况下，回退到软件渲染。
- EXA putimage（上传到视频存储器）会加速，但包含小于 400x400 像素的源可绘制对象除外，在这种情况下，回退到软件渲染。对于 EXA 实心填充和复制操作，只会加速实心平面掩码和 GXcopy 光栅操作。
- 对于 EXA 复制操作，光栅操作（GXandInverted、GXnor、GXorReverse、GXorInverted 和 GXnand）不会加速。
- EXA 复合允许许多选项和源/掩码/目标组合，以进行渲染。
- 大多数（常用的）EXA 复合操作都得到了加速。

以下类型的 EXA 符合操作将加速：

- 包含至少 640 像素的源/目标可绘制对象的复合操作。如果小于 640 像素，则合成路径由软件决定。
- 当源/目标可绘制对象包含超过 200x200 像素时，使用简单的源复合操作（不支持使用掩码的操作）。
- 恒定源（带或不带 alpha 掩码）与目标合成。
- 重复模式源（带或不带 alpha 掩码）与目标合成。
- 只有这些混合函数：SOURCE、OVER、IN、IN-REVERSE、OUT-REVERSE 和 ADD（其中一些是支持加速的组件-alpha 混合所必需的）。
- 通常，以下类型（不太常用）的 EXA 复合操作不会被加速：
  - 转换（即缩放、旋转）的源和掩码
  - 梯度源
  - 带有重复模式的 Alpha 掩码

该实现通过 EXA 回调接口处理 X 的所有像素图分配。首先，将存储器分配到可通过物理 GPU 地址访问的位置。如果剩余的 GPU 可访问存储器不足，该操作可能会失败，但当为像素图请求的每像素比特数小于八（8）时，该操作也可能失败。如果从 GPU 可访问存储器的分配操作失败，则从系统分配存储器。如果从系统分配像素图存储器，那么这个像素图不能包含在 GPU 加速选项中。用于访问像素图存储器的间距字节数可能会有所不同，具体取决于它是从 GPU 可访问存储器还是从系统进行分配的。一旦为 X 像素图分配了存储器，无论是从 GPU 可访问存储器还是从系统分配，像素图都会被锁定，永远无法迁移到其他类型的存储器。不需要将像素图从 GPU 可访问存储器迁移到系统存储器，因为系统虚拟地址始终可用于 GPU 可访问存储器。目前尚未实现从系统存储器到 GPU 可访问存储器的像素图迁移，但只有在初始分配时 GPU 可访问存储器不足但稍后会有更多存储器可用（通过取消分配）的情况下，像素图才会有所帮助。Vivante 2D GPU 的 GPU 可访问存储器间距（水平）对齐为 8 像素。因为存储器可以从 GPU 可访问存储器进行分配，所以这些像素可以在 EGL 中用于 OpenGL/ES 绘图操作。根据 EXA 中使用的存储器管理器，为/dev/fb0 分配的所有存储器都可供内部线性屏幕外存储器管理器使用。屏幕存储器之外的存储器部分可用于分配 X 像素图，该存储器区域可由 GPU 访问。分配给/dev/fb0 的存储器需要比屏幕所需的多几 MB。实际需要的量取决于使用的 X-Windows 和像素图的数量、能否将 X 像素图作为纹理（texture），以及 X-Windows 是否使用了 XComposite 扩展。提供了 X 扩展，即图 1 所示的 VIVEXT，以便 X 客户端可以查询与 X 像素图关联的物理 GPU 地址，前提是该 X 像素图是在 GPU 可访问存储器中分配的。

5.3.6 面向 X-Windows 系统的 i.MX 直接渲染基础设施（DRI）

直接渲染基础设施（DRI）是一个允许以安全有效的方式直接访问 X Window 系统下的图形硬件的框架。其中包括对 X 服务器、多个客户端库和内核（直接渲染管理器（DRM））的更改。DRI 最重要的活动是创建直接渲染到帧缓冲存储器的快速 OpenGL 和 OpenGL ES 实施。如果没有 DRI，OpenGL 驱动必须依赖 X 服务器进行最终渲染（间接渲染），这会显著降低整体性能。

Vivante 的 DRI OpenGL 实施的组件如下所示：

- 直接渲染管理器（DRM）是一个内核模块，为 userland 提供 API，以同步对硬件的访问并管理不同类别的视频存储器缓冲区。Vivante 的 DRI 实施使用选定的 DRM API 来打开/关闭 DRI 设备，以及锁定/解锁 FB。大多数其他缓冲区管理和 DMA 管理功能由 Vivante 的特定内核模块（galcore.ko）进行处理。
- EXA 驱动是一个支持 DRI 的 DDX 2D 驱动，在 X 服务器启动时进行 DRM 初始化。由于所有 X Window 像素图缓冲区都是由 EXA 驱动从 GPU 存储器分配的，如果缓冲区信息从 X 服务器进程正确地传递到 X 客户端进程（GL 或 GLES 应用），GPU 可以直接渲染到这些缓冲区中。
- Vivante 特定的 X 扩展“vivext”将缓冲区信息从 X 服务器传递到 X 客户端。这个 Vivante X 扩展包括以下 3 个接口：
  - DrawableFlush，它允许 X 客户端通知 X 服务器刷新 GPU 缓存以获得可绘制的表面。
  - DrawableInfo，它允许 X 客户端从 X 服务器查询可绘制信息（位置、大小、物理地址、跨距、cliplist 等）。
  - PixmapPhysAddr，它使 X 客户端能够从 X 服务器查询像素图缓冲区的物理地址和跨距。

GL/GLES 应用窗口与 Ubuntu Unity2D 桌面的集成通过以下步骤实现：

- GL/GLES 应用将帧渲染到 EXA 驱动中分配的像素图缓冲区中。
- 在 SwapBuffers 实施中，驱动通过 Xdamage 和 Xfixes API 通知 X 服务器像素图缓冲区已损坏。
- 然后 X 服务器将向 Unity2D 桌面呈现最新的像素图缓冲区，同时保持相对于桌面上其他窗口的正确窗口重叠特性。

在合成 X 桌面上，例如 Ubuntu Unity 2D，GLES/GL 应用总是渲染到窗口的完整矩形后缓冲区中。不需要裁剪窗口。因此 Vivante DRI 实施可以利用 GPU 的解析功能，直接渲染到窗口后缓冲区中。

在遗留 X 窗口桌面上（如 Gnome、Xwin 等），GLES/GL 应用必须直接渲染到帧缓冲区表面。因此，DRI 驱动使用 VIVEXT 扩展中的 DrawableInfo 接口获取窗口的 cliplist，然后根据 cliplist 将渲染目标的子区域复制到帧缓冲区。这将确保 GLES/GL 窗口与桌面上的其他窗口正确重叠。然而，渲染目标子区域到帧缓冲区的复制必须由 CPU 完成，因为子区域的起始地址和对齐可能无法满足 GPU 复制要求。

Vivante DRI 实施可以在运行时检测 X 窗口管理器的类型（是合成桌面管理器还是遗留桌面管理器），并为 GLES/GL 应用使用适当的 DRI 渲染路径。

5.3.7 EGL-X 库

在 X Window 系统中使用时，EGL-X 库实施低级 EGL 接口。下面列出了此实施的特定细节：

- eglDisplay 本机显示类型在 X 中为“Display*”。
- eglWindowSurfacenative 窗口表面类型在 X 中为“Window”。
- eglPixmapSurface 本机像素图表面类型在 X 中为“Pixmap”。

创建 eglWindowSurface 时，用于双缓冲的后台缓冲区可以具有与窗口表面不同的表示形式（基于选定的 eglConfig）。尝试使用在调用 eglSwapBuffers 时向窗口表面提供最高效的后台缓冲区内容的表示来创建每个后台缓冲区。

通过创建必要大小的 X 像素图来分配后台缓冲区。使用 Vivante X 驱动模块的 X 扩展来查询此 X 像素图的物理帧缓冲区地址（如果已在屏幕外的帧缓冲区存储器中分配）。

5.3.8 面向 i.MX 的 xorg.conf

必须正确配置/etc/X11/xorg.conf 文件才能使用 i.MX 6 X 驱动。

必须正确配置/etc/X11/xorg.conf 文件才能使用 Vivante X 驱动。此配置在在文件的“设备”章节介绍，其中包含一些必备条目和一些可选条目。下面的示例展示了使用 Vivante X 驱动的首选配置：

```
Section "ServerLayout"
Identifier    "Default    Layout"
Screen      "Default     Screen"
EndSection
Section "Module"
Load            "dbe"
Load            "extmod"
Load            "freetype"
Load            "glx"
Load            "dri"
EndSection
Section "InputDevice"
Identifier      "Generic Keyboard"
Driver          "kbd"
Option          "XkbLayout" "us"
Option          "XkbModel" "pc105"
Option          "XkbRules" "xorg"
EndSection
Section "InputDevice"
Identifier      "Configured Mouse"
Driver          "mouse"
Option          "CorePointer"
EndSection
Section "Device"
Identifier      "Your Accelerated Framebuffer Device"
Driver          "vivante"
Option          "fbdev"         "/dev/fb0"
Option          "vivante_fbdev" "/dev/fb0"
Option          "HWcursor"      "false"
EndSection
Section "Monitor"
Identifier      "Configured Monitor"
EndSection
Section "Screen"
Identifier      "Default Screen"
Monitor         "Configured Monitor"
Device          "Your Accelerated Framebuffer Device"
DefaultDepth    24
EndSection
Section "DRI"
Mode 0666
EndSection
```

必要字符串

Vivante X 驱动识别的一些重要条目如下所述。

“设备标识符”（Device Identifier）和“屏幕设备”（Screen Device）字符串

“设备”章节中的必要“标识符”（Identifier）条目指定要与此图形设备关联的唯一名称。

```
Section "Device"
Identifier      "Your Accelerated Framebuffer Device"
```

以下条目将特定图形设备绑定到屏幕。“设备标识符”（Device Identifier）字符串必须与 xorg.conf 文件的“屏幕”章节中的“设备”（Device）字符串匹配。 例如：

```
Section "Screen"
Identifier      "Default Screen"
<other entries>
Device          "Your Accelerated Framebuffer Device"
<other entries>
EndSection
```

“设备驱动”（Device Driver）字符串

必要“驱动”（Driver）条目指定可加载的 Vivante X 驱动的名称。

Driver "vivante"

“设备 fbdevPath”（Device fbdevPath）字符串

必要条目 fbdev 和 vivante_dev 指定要使用的帧缓冲设备的路径。

```
Section "Device"
Identifier      "Your Accelerated Framebuffer Device"
Driver          "vivante"
Option          "fbdev"       "/dev/fb0"
Option          "vivante_fbdev" "/dev/fb0"
<other entries>
EndSection
```

5.3.9 在 Yocto 上设置 X-Windows 系统加速

前提条件：

- xserver-xorg-video-imx-viv-(ver).tar.gz，这是基于 GPU 驱动的 Vivante EXA 插件源代码
- drm-update-arm.patch，这是一个为 libdrm xf86drm.h 添加 Arm 锁定实施的补丁。请注意，来自 libdrm 的原始 xh86drm.h 头文件没有用于支持 Arm 架构的锁定。该补丁位于社区 Yocto Project 层 Yocto_build/sources/meta-freescale/recipes-graphics/drm/libdrm/mx6，如下所示：drm-update-arm.patch：

```
+#elif defined(  arm  )
+       #undef DRM_DEV_MODE
+       #define DRM_DEV_MODE     (S_IRUSR|S_IWUSR|S_IRGRP|S_IWGRP|S_IROTH|S_IWOTH)
+
+       #define DRM_CAS(lock,old,new,  ret)             \
+       do {                                            \
+               asm     volatile   (                    \
+                       "1: ldrex %0, [%1]\n"           \
+                       "   teq %0, %2\n"               \
+                       "   strexeq %0, %3, [%1]\n"     \
+               : "r" (  ret)                          \
+               : "r" (lock), "r" (old), "r" (new)      \
+               : "cc","memory");                       \
```

```
+       } while (0)
+
#endif /* architecture */
#endif /*   GNUC_  >= 2 */
```

构建和安装说明：

- 在 Yocto 环境中使用正确的配方在适当的位置安装必备模块或补丁。
- 使用正确的 drm 头文件（xf86drm.h）构建 XServer。目的是创建正确的 dri 模块
- 使用命令“bitbake xf86-video-imxfb-vivante”构建 GPU EXA 模块。创建成功会生成 vivante_drv.so，然后将它与 xorg 和 libdri 库一起安装在/usr/lib/xorg/modules/中的目标电路板 rootfs 中
- 在目标电路板 rootfs 中安装预 Yocto 构建的 imx-gpu-viv 二进制文件。为了加速 X11，需要 X11 后端
- 现在准备好在目标电路板上运行 X11 应用。

注意

如果不使用 Arm 核心版本 xf86drm.h，x11 应用将关闭。

5.3.10 设置 X Window 系统加速

- 安装适合您平台的任何软件包。
- 验证设备文件/dev/galcore 是否存在。
- 验证文件/etc/X11/xorg.conf 是否包含上一节所述的条目。
- 假设已执行上述步骤，请执行以下操作，以验证 X Window System 加速是否在运行。
- 检查日志文件/var/log/Xorg.0.log 并确认存在以下行。

```
[    41.752] (II) Loading /usr/lib/xorg/modules/drivers/vivante_drv.so
       [    41.752] (II) VIVANTE(0): using default device
       [    41.752] (II) VIVANTE(0): Creating default Display subsection in Screen section
"Default Screen" for depth/fbbpp 24/32
       [    41.752] (**) VIVANTE(0): Depth 24, (--) framebufferbpp 32
       [    41.752] (==) VIVANTE(0): RGB weight 888
       [    41.752] (==) VIVANTE(0): Default visual is TrueColor
       [    41.753] (==) VIVANTE(0): Using gamma correction (1.0, 1.0, 1.0)
       [    41.753] (II) VIVANTE(0): hardware: DISP3 BG (video memory: 8100kB)
       [    41.753] (II) VIVANTE(0): checking modes against framebuffer device...
       [    41.753] (II) VIVANTE(0): checking modes against monitor...
       [    41.753] (--) VIVANTE(0): Virtual size is 1920x1080 (pitch 1920)
       [    41.753] (**) VIVANTE(0):  Built-in mode "current": 148.5 MHz, 67.5 kHz, 60.0 Hz
       [    41.753] (II) VIVANTE(0): Modeline "current"x0.0  148.50  1920 2008 2052 2200  1080
1084 1089 1125 +hsync +
      vsync -csync (67.5 kHz)
       [    41.753] (==) VIVANTE(0): DPI set to (96, 96)
       [    41.753] (II) Loading sub module "fb"
       [    41.753] (II) LoadModule: "fb"
       [    41.754] (II) Loading /usr/lib/xorg/modules/libfb.so
       [    41.755] (II) Module fb: vendor="X.Org Foundation"
       [    41.755] compiled for 1.10.4, module version = 1.0.0
       [    41.755] ABI class: X.Org ANSI C Emulation, version 0.4
       [    41.755] (II) Loading sub module "exa"
       [    41.755] (II) LoadModule: "exa"
       [    41.756] (II) Loading /usr/lib/xorg/modules/libexa.so
       [    41.756] (II) Module exa: vendor="X.Org Foundation"
       [    41.756] compiled for 1.10.4, module version = 2.5.0
       [    41.756] ABI class: X.Org Video Driver, version 10.0
```

```
       [    41.756] (--) Depth 24 pixmap format is 32 bpp
       [    41.797] (II) VIVANTE(0): FB Start = 0x33142000  FB Base = 0x33142000  FB Offset =
(nil)
       [    41.797] (II) VIVANTE(0): test Initializing EXA
       [    41.798] (II) EXA(0): Driver allocated offscreenpixmaps
       [   41.798] (II) EXA(0): Driver registered support for the following operations:
       [    41.798] (II)         Solid
       [    41.798] (II)         Copy
       [    41.798] (II)         Composite (RENDER acceleration)
       [    41.798] (II)         UploadToScreen
       [    42.075] (==) VIVANTE(0): Backing store disabled
       [    42.084] (==) VIVANTE(0): DPMS enabled
```

5.3.11 故障排除

1. 可以通过环境变量指定帧缓冲设备。这在有多个帧缓冲设备时尤为重要。

   ```
   export FB_FRAMEBUFFER_0=/dev/fb2
   ```

2. 如果上述方法无法解决问题：
   - 如果 DRM 正常启动，请检查/var/log/X11.n 日志文件（n 代表实例编号）以获取更多信息。
   - 如果 DRM 没有正常启动，请检查您的内核模式驱动安装情况。（参见上文第 6.4.2 和 6.4.3 节）。
3. 创建了窗口，但没有绘制任何内容
   - 如果运行 OpenGL 应用，发现创建了一个窗口，但没有绘制任何内容，请尝试导出 ${ GL_DEV_FB}环境变量：

     ```
     export   GL_DEV_FB=$FB_FRAMEBUFFER_0.
     ```

4. “无法打开显示器”消息
   - 如果有类似于“Cannot open Display”（“无法打开显示器”）的消息，请使用以下命令检查 X 是在:0 还是在:1 实例运行，请使用：

     ```
     $ ps –ef|grep X
     ```

   - 然后根据返回的实例编号，添加以下环境变量

     ```
     export DISPLAY=:n
     ```

   - 然后再次运行。
5. UART 终端无法使用 lightdm 运行 GPU 应用
   - 改用 ssh 终端。
6. EXA 构建脚本失败
   - 检查日志文件，确保系统时间设置正确。
7. “Invalid MIT-MAGIC-COOKIE-1 Key”错误信息
   - 某些 GPU 应用不允许使用 root 运行。请改用备用帐户。
8. 运行 GPU 应用时出现段故障
   - 检查 dev/galcore 的属性是否应更新为 666。
   - 要在系统启动时自动更新此属性，
   - 找到并编辑文件/etc/udev/rules.d/<bsp-specific.rules>。

- 添加：“KERNEL==”galcore”，MODE=”0666””
- 最后，确保内核和 GPU 驱动相匹配。

9. 检查 Compiz 是否正在运行
   - 如果您的主机或目标在安装表 6 中的 OpenGL 开发包后出现问题，请使用以下命令检查 compiz 是否正在运行：

```
$ ps -ef|grep compiz
```

   - 如果 compiz 正在运行，则 Ubuntu 默认使用 Unity3D。要将默认窗口管理器设置为 Unity2D：
   - 找到并编辑文件/var/lib/AccountsService/users/<username>。
   - 将 ubuntu 更改为 ubunto-2d。

第 6 章
视频

6.1 采集概述

6.1.1 介绍

i.MX 采集驱动程序的支持是通过带有相机传感器控制器和接口的 V4L2 接口实现的。应用程序不能直接使用相机驱动程序。相反，应用程序使用 V4L2 采集驱动程序来打开和关闭摄像头以进行预览和图像采集、控制摄像头、从摄像头获取图像以及启动摄像头预览。

采集控制器列表如下所示：

- 摄像头串行接口——CSI
- IPU-CSI
- 视频接口单元——VIU
- 图像传感器接口——ISI
- 图像传感器处理——ISP

传输图像数据的采集接口列表如下所示：

- Parallel-CSI
- MIPI-CSI2
- HDMI RX
- TV Decoder

本章介绍各种控制器和接口之间的差异。

注意

带 IPU 的 i.MX 6 将 internaldev 作为 V4L2 接口，而其他芯片则将 subdev 作为 V4L2 接口。

下表列出了不同的控制器和接口组合。

表 52. 摄像头控制器和接口

SoC	控制器	接口
6SLL	CSI	Parallel CSI
6SoloX	VIU	Parallel CSI 和 TV Decoder
6UltraLite/6ULL	CSI	Parallel CSI
6DualLite/Solo	IPU-CSI	Parallel CSI internaldev IPU
6QuadPlus/Quad/Dual	IPU-CSI	Parallel CSI internaldev IPU
7Dual/Solo	CSI	使用三星的 MIPI-CSI2 接口和 Parallel CSI 接口
8M Plus/8M Nano	ISI	MIPI-CSI2
8M Plus	ISP	MIPI-CSI2
8QuadMax	ISI	使用 Mixel 的 MIPI-CSI2 和使用 Cadence 的 HDMI Rx 接口

下页继续.....

表 52. 摄像头控制器和接口（续）

SoC	控制器	接口
8QuadXPlus	ISI	使用 Mixel 的 MIPI-CSI2 和使用 IMX8 的 Parallel CSI 接口
8M Quad	CSI	使用 Mixel 的 MIPI-CSI2 接口
8M Mini	CSI	使用三星的 MIPI-CSI2 接口

下面列出了一些额外的详细信息：

- ISP 是用于 i.MX 8M Plus 的新控制器。
- ISI 控制器是用于 i.MX 8 系列中部分芯片的新控制器。
- 不带 IPU 的 i.MX 6 SoC、i.MX 7Dual 和 i.MX 8M 使用相同的 CSI 控制器驱动。
- i.MX 8 和 i.MX 8X 系列使用较新的 i.MX 8 CSI 驱动。
- 带 IPU 的 i.MX 6 使用与 IPU 硬件对接的自定义 CSI。
- 每个 SoC 都可以支持上表所述的一个或多个接口。这些接口与 Video for Linux V4L2 API 一致。
- 在某些情况下，采集控制器不是与摄像头连接，而是连接到视频输入单元。有些还连接到 HDMI 接收器。

6.1.2 Omnivision Camera

Omnivision Camera 支持多种接口和摄像头类型。Omnivision Camera 是一种低功耗的小型摄像头传感器和镜头模块。

Omnivision Camera 使用串行摄像头控制总线（SCCB）接口来控制传感器操作，作为 I2C 客户端执行控制操作。该摄像头支持 CSI、MIPI-CSI2 和 Parallel-CSI 接口的传输模式。使用 MIPI 模式时，OV5640 通过 MIPI CSI-2 接口连接到 i.MX 芯片。MIPI 接收传感器数据并将其传输给 CSI。

下表列出了不同的 Omnivision 摄像头以及支持的接口。

表 53. 摄像头控制器和接口

摄像头	控制器接口
OV5640	CSI 控制器/MIPI-CSI2/Parallel CSI
OV5642	Parallel CSI
OV10635	MIPI-CSI2/Parallel CSI

Omnivision 菜单配置有多个基于支持的选项

顶层选择如下所示：

Device Drivers > Multimedia support (MEDIA_SUPPORT [=y]) > V4L platform devices (V4L_PLATFORM_DRIVERS)

下一级选择基于每个 SoC 的不同接口

- 对于带 IPU 的 i.MX 6，同时选择“> MXC Camera/V4L2 PRP Features support”和“> OmniVision ov5640 camera support (MXC_CAMERA_OV5640)”。
- 对于不带 IPU 的 i.MX 6，选择“> OmniVision ov5640 camera support (MXC_CAMERA_OV5640_V2)”。
- 对于 i.MX 7，选择“> OmniVision ov5640 camera support using MIPI (MXC_CAMERA_OV5640_MIPI_V2)”。

- 对于 i.MX 8，选择“> IMX8 Camera ISI/MIPI Features support (VIDEO_MX8_CAPTURE) > IMX8 Camera Controller (IMX8_CAPTURE_DRIVER) and Maxim OV5640_V3 driver support (Device Drivers > Statging Drivers > Media Staging Drivers > i.MXqQXP/QM Camera ISI/MIPI Features support)”。
- 对于 i.MX 8M，选择“> OmniVision ov5640 camera support (MXC_CAMERA_OV5640_V2) and OmniVision ov5640 camera support using MIPI (MXC_CAMERA_OV5640_MIPI_V2)”。

下表介绍了每个接口支持的摄像头功能。

表 54. 采集接口功能

接口	功能
IPU-CSI	• Parallel CSI 有 2 个端口，分别为 20 位+ 8 位 • 播放 1080i/p + D1@30fps @ 30fps • 录制 1080p@30fps • 双向 720@30fps • 反交错高质量运动自适应算法。 • 调整大小——非常灵活 • 旋转/反转支持 • 颜色转换——非常灵活 • 存储器接口——AXI 分割事务 64 位 266 MHz • 存储器总线—— 选择性读取以进行组合 • 控制能力——显示和 DMA 控制器，内部同步 • 同步——采用内部存储器进行双/三重缓冲，逐帧或紧密子帧
CSI	• MIPI-CSI-2，2 个端口，4 通道 x 1.5 Gbps • 播放—— 两个 1080p30 • 录制——1080p30x2 • 双向——1080p30x2 • 反交错——简单的 bob 和 weave • 存储器接口吞吐量——64 位 • 控制器功能——DMA • 同步——双缓冲
ISI-8QuadXPlus	• Parallel CSI，1 个端口，24 位 • MIPI-CSI-2，1 个端口，4 通道 x 1.5 Gbps • 播放——1080p30x2 • 录制——1080p30x2 • 双向——1080p30x2 • 反交错——简单的 bob 和 weave • 调整大小

下页继续.....

表 54. 采集接口功能（续）

接口	功能
	• JPEG 编码/解码 • 存储器接口吞吐量——124 位，400 MHz • 控制器功能——DMA • 同步——双缓冲
ISI-8QuadMax	• MIPI-CSI-2，2 个端口、4 通道 x 1.5 Gbps • HDMI 接收器——1 个端口 HDMI 1.4 4K30 • 播放——4K60x1、4K30x1 或 1080px30x4 • 录制——4K30x1 或 1080px30x4 • 双向——4K30x1 或 1080px30x4 • 反交错——简单的 bob 和 weave • 调整大小 • JPEG 编码/解码 • 存储器接口吞吐量——124 位，400 MHz • 控制器功能——DMA • 同步——双缓冲
ISI-8M Plus	• MIPI-CSI-2，2 个端口，4 通道 x 1.5 Gbps • 支持一个 4K 分辨率信号源，每秒 30 帧（24 bpp） • 在每个通道上在 60 fps (24 bpp)下支持两个 2K 分辨率的视频源 • 反交错——简单的 bob 和 weave • 调整大小
ISI-8M Nano	• MIPI-CSI-2，1 个端口，4 通道 x 1.5 Gbps • 在 60 fps (24 bpp) 下支持一个分辨率高达 2K 的信号源 • 反交错——简单的 bob 和 weave • 调整大小

6.1.3 Parallel CSI

Parallel CSI 驱动可直接连接到外部 CMOS 传感器和 CCIR656 视频源。CSI 和传感器驱动在 Video for Linux 2（V4L2）驱动框架中实施，该框架由图像采集驱动和视频输出驱动组成。

驱动初始化 CSI 接口并配置和操作 CSI 模块的硬件寄存器。支持以下功能：

- 可配置的接口逻辑，支持最常用的 CMOS 传感器。
- 完全控制 8 位/像素、10 位/像素或 16 位/像素数据格式到 32 位 Rx FIFO 打包。
- 128x32 FIFO 用于存储接收到的图像像素数据。

- 接收 FIFO 溢出保护机制。
- 嵌入式 DMA 控制器通过 AHB 总线传输来自 Rx FIFO 或统计 FIFO 的数据。
- 支持在外部存储器中双缓冲两个帧。
- 从可屏蔽中断源选取单个中断源到中断控制器：帧开始、帧结束等。
- 可配置的主时钟频率输出到传感器。

V4L2 CSI 采集设备包含两个接口：采集接口和覆盖层接口。采集和覆盖层接口使用 CSI 嵌入式 DMA 控制器，使用 V4L2 API 实施该功能。以下是采集和覆盖层的数据流。

1. 摄像头通过 8 位/10 位数据端口将数据发送到 CSI Rx FIFO。
2. 嵌入式 DMA 控制器通过 AHB 总线将数据从 Rx FIFO 传输到外部存储器。
3. 数据保存到用户空间存储器或直接输出到帧缓冲区。

带 IPU 的 i.MX 6 使用直接与 IPU 对接的 IPU-CSI 驱动。i.MX Quad Plus/Quad/Dual 支持两个 IPU-CSI 传感器。不带 IPU 的 i.MX 6 和 i.MX 7Dual/Solo 使用单独的 CSI 传感器驱动直接连接到传感器。

6.1.4 MIPI 摄像头串行接口（MIPI CSI）

MIPI CSI-2 D-PHY 中有 4 个模块：PHY 适配层、数据包分析器、图像数据接口和寄存器 bank。

MIPI CSI-2 是一款 MIPI 摄像头串行接口主控制器，具有用于移动应用的高性能串行互连总线，可将摄像头传感器连接到主系统。CSI-2 主控制器是一个数字核心，可实现 MIPI CSI-2 规范中定义的所有协议功能。在此过程中，它在系统和 MIPI D-PHY 之间提供了一个接口，并允许与符合 MIPI CSI-2 规范的摄像头传感器进行通信。

MIPI CSI2 驱动用于管理 MIPI D-PHY，并允许其与 MIPI 传感器和 IPU CSI 一起工作。MIPI CSI2 驱动实现如下功能：

- MIPI CSI-2 低级接口，用于管理 mipi D-PHY 寄存器和时钟
- MIPI CSI-2 通用 API，用于 MIPI 传感器和 MIPI D-PHY 之间的通信

通过调用 MIPI 通用 API，MIPI 传感器可向 MIPI CSI2 驱动设置传感器的某些信息（如数据类型、通道号等），以配置 D-PHY。为了正确配置 IPU CSI 模块驱动，需要接收适当的数据并进行正确的处理，该驱动必须从 MIPI CSI2 驱动接收有关传感器的信息（如数据类型、虚拟通道、IPU ID、CSI ID 等）。

功能和操作如下所示：

- PHY 适配层负责管理 D-PHY 接口，包括 PHY 错误处理；
- 数据包分析器负责数据通道合并（如果需要），以及报头解码、错误检测和纠正、帧大小错误检测和 CRC 错误检测。
- 图像数据接口分离 CSI-2 数据包头信息并根据存储器存储格式对数据进行重新排序。它还可以生成定时精确的视频同步信号。还在帧级和行级执行若干错误检测；
- 可通过标准 AMBA-APB 从接口访问寄存器 bank，并提供对 CSI-2 主控制器寄存器的访问，以进行配置和控制。还有一个完全可编程的中断发生器，可在发生某些事件时通知系统；

Linux 操作系统的 MIPI CSI2 驱动分为两个部分：MIPI CSI2 驱动初始化操作，用于初始化 mipi_csi2_info 结构；MIPI CSI2 通用 API，用于导出 CSI 模块驱动和 MIPI 传感器驱动的 API。

6.1.5 HDMI

HDMI 视频与图像传感器接口（ISI）对接。

在 i.MX 8QuadMax 上，HDMI 接收器视频接口支持单端口 HDMI 1.4 4K30。

6.1.6 软件操作

V4L2 采集支持模式、图片格式和图片大小随每个采集接口而变化。

imx-test repo 在 mxc_v4l2_test 中包含对这些接口的单元测试。如需了解如何运行测试的详细信息，请参见 README（读我文件）。

6.1.7 V4L2 Capture

Video for Linux Two (V4L2) 是一个 Linux 标准。如需获得 API 规范，请访问 https://www.kernel.org/doc/html/latest/userspace-api/media/v4l/v4l2.html。

V4L2 capture 设备包含两个接口：采集接口和覆盖层接口，使用用于采集和覆盖层设备的 V4L2 API。

下面列出了 V4L2 capture API 的一些示例用例：

1. 使用 IOCTL VIDIOC_S_FMT 设置采集像素格式和大小。
2. 使用 IOCTL VIDIOC_S_CTRL 设置控制信息，用于旋转。
3. 使用 IOCTL VIDIOC_REQBUFS 请求缓冲区。
4. 存储器将缓冲区映射到其用户空间。
5. 执行 IOCTL VIDIOC_DQBUF。
6. 将需要后处理的数据传递到缓冲区。
7. 使用 IOCTL 命令 VIDIOC_QBUF 对缓冲区进行排队。
8. 通过执行 IOCTL VIDIOC_STREAMON 启动流。
    - 无法同时启用 VIDIOC_STREAMON 和 VIDIOC_OVERLAY。

下表列出了 i.MX Capture 驱动中使用的 V4L2 capture ioctl。如需了解详细信息，请参见“V4I2”章节。

表 55. V4L2 Capture API IOCTL

IOCTL	说明
VIDIOC_QUERYCAP	查询设备能力
VIDIOC_G_FMT VIDIOC_S_FMT	获取或设置数据格式
VIDIOC_S_DEST_CROP	设置裁剪矩形
VIDIOC_REQBUFS	启动存储器映射
VIDIOC_QueryBUF	查询缓冲区状态
VIDIOC_QBUF,VIDIOC_DQBUF	与驱动交换缓冲区
VIDIOC_STREAMON,VIDIOC_STREAMOFF	开始或停止流式传输
VIDIOC_G_CTRL,VIDIOC_S_CTRL	获取或设置控件的值
VIDIOC_CROPCAP	查询裁剪功能
VIDIOC_G_CROP,VIDIOC_S_CROP	获取或设置裁剪
VIDIOC_OVERLAY	开始或停止视频覆盖

下页继续.....

表 55. V4L2 Capture API IOCTL（续）

IOCTL	说明
VIDIOC_G_FBUF, VIDIOC_S_FBUF	获取或设置帧缓冲区覆盖参数
VIDIOC_G_PARM,VIDIOC_S_PARM	获取或设置流传输参数
VIDIOC_G_STD,VIDIOC_S_STD	获取或设置视频标准
VIDIOC_G_OUTPUT,VIDIOC_S_OUTPUT	获取或设置视频输出
VIDIOC_G_INPUT,VIDIOC_S_INPUT	获取或设置视频输入
VIDIOC_ENUMSTD	枚举视频标准
VIDIOC_ENUMOUTPUT,VIDIOC_ENUMINPUT	枚举输出和输入
VIDIOC_ENUM_FMT	枚举图像格式
VIDIOC_ENUM_FRAMESIZE,VIDIOC_ENUM_ FRAMEINTERVALSS	枚举帧大小和间隔
VIDIOC_DBG_G_CHIP_IDENT	芯片识别

6.1.8 源代码结构

下表列出了采集驱动源文件。对于 i.MX 6 和 i.MX 7，源文件位于 drivers/media/platform/mxc/ capture。对于 i.MX 8 系列，源文件位于 drivers/media/platform/imx8。对于 MIPI-CSI，源文件位于 drivers/mxc/mipi。

表 56. Omnivision V4L2 Camera 驱动文件

文件	说明
• drivers/media/platform/imx8/mipi-csi2.c • drivers/media/platform/imx8/mipi-csi2.h • drivers/media/platform/imx8/mipi-csi2-yav.c	i.MX 8 MIPI-CSI2 Capture 接口驱动
• drivers/staging/media/imx/parallel-csi.c • drivers/staging/media/imx/parallel-csi.h	i.MX 8 MIPI-CSI2 Parallel-CSI 接口驱动
• drivers/staging/media/imx/mxc-isi-core.c • drivers/staging/media/imx/mxc-isi-cap.c • drivers/staging/media/imx/mxc-isi-hw.c	i.MX 8 ISI Capture 控制器驱动
• drivers/media/i2c/ov5640.c • drivers/staging/media/imx/gmsl-max9286.c	i.MX 8 Omnivision Camera V3 Camera 接口
• drivers/media/platform/imx8/mxc-jpeg-hw.c • drivers/media/platform/imx8/mxc-jpeg-hw.c	i.MX 8 JEPG 硬件接口
• drivers/mxc/mipi/mxc_mipi_csi2.c • drivers/mxc/mipi/mxc_mipi_csi2.h	i.MX 6 和 i.MX 7 MIPI-CSI2 接口核心驱动

下页继续.....

表 56. Omnivision V4L2 Camera 驱动文件（续）

文件	说明
• drivers/media/platform/mxc/capture/ipu_bg_overlay_sdc.c • drivers/media/platform/mxc/capture/ipu_csi_enc.c • drivers/media/platform/mxc/capture/ipu_fg_overlay_sdc.c • drivers/media/platform/mxc/capture/ipu_prp_enc.c • drivers/media/platform/mxc/capture/ipu_prp_vf_sdc_bg.c • drivers/media/platform/mxc/capture/ipu_prp_vf_sdc.c • drivers/media/platform/mxc/capture/ipu_still.c • drivers/media/platform/mxc/capture/v4l2-int-device	i.MX 6 IPU V4L2 插件
• drivers/media/platform/mxc/capture/mx6s_capture.c • drivers/media/platform/mxc/capture/ov5640.c • drivers/media/platform/mxc/capture/ov5640_camera_int.c	CSI Omnivision Camera V4L2 插件
• drivers/media/platform/mxc/capture/ov5640_v2.c • drivers/media/platform/mxc/capture/ov5640_camera_v2.c	Paralllel CSI Omnivision Camera V4L2 插件
• drivers/media/platform/mxc/capture/ov5640_mipi.c • drivers/media/platform/mxc/capture/ov5640_camera_mipi_int.c	MIPI-CSI Omnivision Camera V4L2 插件
• drivers/media/platform/mxc/capture/ov5640_mipi_v2.c • drivers/media/platform/mxc/capture/ov5640_camera_mipi_v2.c	MIPI-CSI2 Omnivision Camera V4L2 插件
• drivers/media/platform/mxc/capture/adv7180.c • drivers/media/platform/mxc/capture/adv7180_tvin.c	TV 解码器 ADV7180 V4L2
• drivers/staging/media/imx/hdmirx/cdns-hdmirx-audio.c • drivers/staging/media/imx/hdmirx/cdns-hdmirx-hdcp.c • drivers/staging/media/imx/hdmirx/cdns-hdmirx-hw.c • drivers/staging/media/imx/hdmirx/cdns-hdmirx-phy.c • drivers/staging/media/imx/hdmirx/cdns-hdmirx-phy.h • drivers/staging/media/imx/hdmirx/cdns-hdmirx.c • drivers/staging/media/imx/hdmirx/cdns-mhdp-hdmirx.c • drivers/staging/media/imx/hdmirx/cdns-mhdp-hdmirx.h	i.MX 8 HDMI RX

6.2 显示概述

6.2.1 介绍

i.MX 显示系统使用显示控制器来优化视频数据移动以显示接口和图形处理。每个显示控制器都通过 Linux 驱动程序实现并进入显示框架（帧缓冲区或 DRM）。在某些情况下，显示控制器包括确保安全视频管道的身份验证。在其他情况下，显示控制器将包括在传输期间用于缩放、去隔行、平铺和颜色转换的附加功能。对于 i.MX 8，支持多个显示器是通过使用两个控制器一起工作来完成的。本章提供 i.MX 显示控制器和接口的高级概述，以及帧缓冲区和 DRM 显示驱动程序之间的区别。使用以下显示控制器。

- IPU
- PXP
- eLCDIF
- DPU
- DCSS——仅在 i.MX 8M 上

显示接口将与显示控制器、显示屏以及编码器显示桥（在某些情况下）对接。支持以下显示接口。

- EPDC——支持 EInk 显示器
- Parallel——支持 LCD 显示器
- LVDS——支持 LVDS 显示器
- HDMI——支持片上和外部 HDMI
- 显示端口——支持 eDP 屏
- MIPI-DSI——支持 MIPI 显示器

注意

不再支持模拟显示。模拟接口用于 i.MX 37 和 i.MX 5 系列。

支持以下 HDMI 显示桥接器/编码器。

- Parallel 到 HDMI——使用 Silicon Image si902x
- LVDS 到 HDMI——使用 ITE it6263
- MIPI-DSI 到 HDMI——使用 Analog Devices adv7535

每个 SOC 支持不同的显示功能。其中一些功能在位于 arch/arm/boot/dts 和 arch/arm64/boot/dts 的设备树中进行配置。如需了解以下方面的更多详细信息，请参见硬件参考手册。

- 吞吐量
  - 输出数量
  - 像素时钟速率
  - 最大显示器数量和相应的分辨率
  - 60 Hz 时的分辨率。
- 接口
  - Parallel——端口数和位大小
  - LVDS——通道数和信道数。
  - MIPI-DSI——端口数、通道数、信道数和速度
- 处理

— 动态组合，包括高分辨率显示器

— 离线组合速度

6.2.2 帧缓冲区

i.MX 6 和 i.MX 7 支持帧缓冲驱动，但 i.MX 8 不支持。使用 drivers/video/fbdev 中的 imxfb 驱动支持帧缓冲驱动。帧缓冲内核 fbdev 结构在此处或 kernel.org 上的此处定义。如需了解 i.MX V4LS 的更多信息，请参见“V4L2”章节。

这些屏由 video/fbdev/mxc 文件夹中的 TRULY 和 EInk 屏的帧缓冲驱动提供支持。在 imx_v7_defconfig 中搜索 PANEL，查看支持的屏。只有 MIPI DSI 接口支持 Trully 屏。只有 EPDC 接口支持 Eink 屏。

6.2.3 直接渲染模型（DRM）

直接渲染模型（DRM）是适用于 i.MX 的新显示驱动。i.MX DRM 驱动位于 drivers/gpu/drm/imx 中。其他组件具有 DRM 接口，例如 GPU 和 DCSS。DRM 框架记录在 kernel.org 的此处。

i.MX DRM 驱动使用以下驱动实现。

- 硬件库支持文件
- 核心 DRM 驱动
- 硬件相关的 DRM 驱动
- 支持 hdp HDMI/Display Port 的 HDMI DRM 驱动

DRM 驱动对 i.MX 8QuadMax 和 i.MX 8QuadXPlus 使用 DPU，对 i.MX 8M Quad 和 i.MX 8M Mini 使用 LCDIF，对 i.MX 8ULP 使用 DCNANO。

i.MX DRM 框架还包括 driver/gpu/panel 中的屏驱动。支持的 DRM 屏包括 Simple panel、Raydium RM67191、Raydium RM68200 和 Raydium RM67199。

6.2.4 显示分辨率

显示分辨率计算使用以下因子。

- 帧宽
- 帧高
- 帧速率（fps）
- 隐藏间隔——在显示器的 DS 中高达 35%(1.35)，使用最小值

像素时钟[MHz]是根据帧宽 x 帧高 x 帧速率 x 隐藏间隔计算的，需要考虑的事项如下所示：

- 数据格式（每时钟像素）
- 显示器的源时钟（DI#_CLK_EXT 位）
- 显示控制器（DC）上的负载

6.2.5 身份验证

显示身份验证允许硬件处理，以确保显示内容不受损害。通过使用身份验证硬件的显示身份验证 CRC 来完成。该硬件是通过 i.MX 6 上的帧缓冲显示框架集成的 DCIC，以及在 i.MX 8 的 DRM 显示框架中实现的 DPU。

以下 SoC 支持显示身份验证 CRC。

- i.MX 6 Solox 支持使用 DCIC 对 1 个显示器进行身份验证。

- i.MX 6 QuadPlus/Quad/Dual 支持使用 DCIC 对 2 个显示器进行身份验证。
- i.MX 8QuadXPlus 可以使用 DPU 验证 2 个显示器。
- i.MX 8QuadMax 可以使用 DPU 验证 4 个显示器。

6.2.6 平铺

通过硬件平铺提供优化的视频数据显示。这通过不同的硬件块实现。最新功能是显示预取解析（DPR），它提高了 i.MX 6 QuadPlus、i.MX 8QuadMax 和 i.MX 8QuadXPlus 的性能。

已在以下器件上启用平铺支持：

- i.MX 6Quad/Dual 支持使用视频数据顺序适配器（VDOA）进行平铺。
- i.MX 6QuadPlus 支持平铺 VDOA 和显示预取解析（DPR）版本 1
- i.MX 8QuadXPlus 和 i.MX 8QuadMax 支持使用显示预取解析（DPR）版本 2 进行平铺

6.3 显示控制器

6.3.1 显示处理单元（DPU）

6.3.1.1 介绍

显示处理单元（DPU）设计用于支持视频和图形处理功能，并与视频和静态显示传感器和显示器连接。DPU 驱动提供内部内核级 API 来操控逻辑通道。逻辑通道代表完整的 DPU 处理流程。例如，完整的 DPU 处理流程（逻辑通道）可能包括从存储器中读取 YUV 缓冲区并将其显示到外部接口。DPU API 由一组适用于所有通道的通用函数组成。它的功能包括初始化通道、设置缓冲区、启用和禁用通道以及设置中断。

典型的逻辑通道包括：

- CSI 直接到存储器
- 存储器到同步帧缓冲背景
- 存储器到同步帧缓冲前景

更高级别的驱动负责存储器分配并提供用户级 API。DPU 接口可用于 V4L2 框架中的采集并使用 DRM 显示框架进行显示。DPU 与 LVDS、MIPI-DSI、HDMI 和 Parallel 显示接口对接。

DPU 显示控制器支持 32 位显示合成引擎，包含以下内容：

- 独立屏上的 2 个显示输出流。
- 两层结构
- 使用区域 CRC 检查器进行 CRC 匹配，进行自动安全流卡死检测

DPU 显示控制器支持 2D 合成引擎，可提供高效率、性能和安全性。DPU 2D 图形引擎支持减轻了 GPU 的负担，它只做 3D GPU。覆盖原生视频和图形的视频效率可最大限度地减少系统存储器访问。高能效特性允许 3D 引擎关闭，以便像 Android 硬件合成器一样打开 GUI 窗口。

DPU 还支持以下身份验证功能。

- 具有 8 个可堆叠区域的 CRC 检查器，可屏蔽，独占的自上而下优先级
- 可以在后处理 pipeline 的任何阶段之后插入 CRC 检查
- CRC 故障会产生软件中断，或将帧发生器切换到安全流或恒定平面

DPU 显示接口缓存支持以下功能。

- 每个显示平面都有一个多行缓存
- 每个平面 8 行像素
- 支持 RGB、YUV 等格式
- 支持视频和 GPU 平铺格式
- 提前从存储器获取内容以填充缓存
- 支持水平和垂直提取
- 不支持 Warp 提取，需要旁路

6.3.1.2 DRM

显示处理单元（DPU）与支持视频显示的 DRM 驱动对接。

6.3.1.3 源代码结构

DPU 驱动分为 DRM、blitting 和主处理。drivers/gpu/drm/imx/dpu 和 drivers/gpu/imx/dpu-blit 中提供了常用函数，而主驱动位于于 drivers/gpu/imx/dpu。下表列出了源文件。

表 57. DPU 驱动源文件

文件	说明
DRM 源文件	
drivers/gpu/drm/imx/dpu/dpu-plane.	DRM DPU Plane
drivers/gpu/drm/imx/dpu/dpu-crtc	DRM DPU CRTC
drivers/gpu/drm/imx/dpu/dpu-blit	DRM DPU blitter
drivers/gpu/drm/imx/dpu/dpu-kms	DRM DPU KMS
DPU Blitter 源文件	
drivers/gpu/imx/dpu-blit/dpu-blit	DPU Bliter
drivers/gpu/imx/dpu-blit/dpu-blit-registers.h	DPU Blit 寄存器
DRM Core 源文件	
drivers/gpu/imx/dpu/dpu-vscaler.c	DPU VScaler
drivers/gpu/imx/dpu/dpu-fetchwarp.c	DPU Fetchwarp
drivers/gpu/imx/dpu/constframe.c	DPU Const Frame
drivers/gpu/imx/dpu/dpu-prv.h	DPU Private 头
drivers/gpu/imx/dpu/dpu-disengcfg.c	DPU 显示配置
drivers/gpu/imx/dpu/dpu-fetchunit.c	DPU 提取单元
drivers/gpu/imx/dpu/dpu-framegen.c	DPU 帧发生器
drivers/gpu/imx/dpu/dpu-hscaler.c	DPU HScaler
drivers/gpu/imx/dpu/dpu-extdst.c	DPU 外部目的
drivers/gpu/imx/dpu/dpu-common.c	DPU Common
drivers/gpu/imx/dpu/dpu-fetchlayer.c	DPU 提取层

下页继续.....

表 57. DPU 驱动源文件（续）

文件	说明
drivers/gpu/imx/dpu/dpu-tcon.c	DPU TCon
drivers/gpu/imx/dpu/dpu-layerblend.c	DPU 层混合
drivers/gpu/imx/dpu/dpu-fetcheco.c	DPU 提取编码
drivers/gpu/imx/dpu/dpu-fetchdecode.c	DPU 解码

6.3.1.4 菜单配置选项

为 DPU 模块提供了以下 Linux 内核配置选项。

Device Drivers -> i.MX DPU core support

6.3.2 图像处理单元（IPU）

6.3.2.1 介绍

图像处理单元（IPU）旨在支持视频和图形处理功能，并与视频和静态图像传感器及显示器连接。IPU 驱动提供内核级 API 来操纵逻辑通道。一个逻辑通道代表一个完整的 IPU 处理流程。例如，完整的 IPU 处理流程（逻辑通道）可能涉及的操作包括从存储器读取 YUV 缓冲区、执行后处理，以及将 RGB 缓冲区写入存储器。逻辑通道映射 1 到 3 个 IDMA 通道，并映射到零个或一个 IC 任务。1 个逻辑通道可以有一个输入、一个输出和一个备用输入 IDMA 通道。IPU API 由一组适用于所有通道的通用函数组成。它的功能包括初始化通道、设置缓冲区、启用和禁用通道、为实现自动帧同步将通道链接起来以及设置中断。

IPU 是一个显示控制器，支持帧缓冲区显示框架支持的以下显示接口。该访问仅通过帧缓冲区 fbdev 应用框架公开。

- Parallel
- LVDS
- HDMI
- MIPI-DSI

典型的逻辑通道包括：

- CSI 直接到存储器
- CSI 到取景器（viewfinder）预处理到存储器
- 存储器到取景器（viewfinder）预处理到存储器
- 存储器到取景器（viewfinder）旋转到存储器
- 存储器到视频反交错和取景器预处理到存储器的前一个场通道
- 存储器到视频反交错和取景器预处理到存储器的当前场通道
- 存储器到视频反交错和取景器预处理到存储器的下一个场通道
- CSI 到编码器预处理到存储器
- 存储器到编码器预处理到存储器
- 存储器到编码器旋转到存储器
- 存储器到后处理旋转到存储器
- 存储器到同步帧缓冲区背景
- 存储器到同步帧缓冲区前景

- 存储器到同步帧缓冲区 DC
- 存储器到同步帧缓冲区掩码

IPU API 具有一些附加功能，这些功能并非在所有通道中都通用，并且特定于 IPU 子模块。IPU 子模块的功能类型如下所示：

- 同步帧缓冲功能
- 屏接口初始化
- 设置前景位置
- 设置本地/全局 alpha 和颜色键
- 设置伽玛校正
- CSI 功能
- 传感器接口初始化
- 设置传感器时钟
- 设置采集大小
- 使用 PRE/PRG 启用或禁用预取线性帧
- 使用 PRE/PRG 启用或禁用解析平铺帧

更高级别的驱动负责存储器分配、通道链接，并提供用户级 API。

6.3.2.2 硬件操作

《应用处理器参考手册》中介绍了 IPU 的硬件操作详情。下图展示了 IPU 硬件模块。

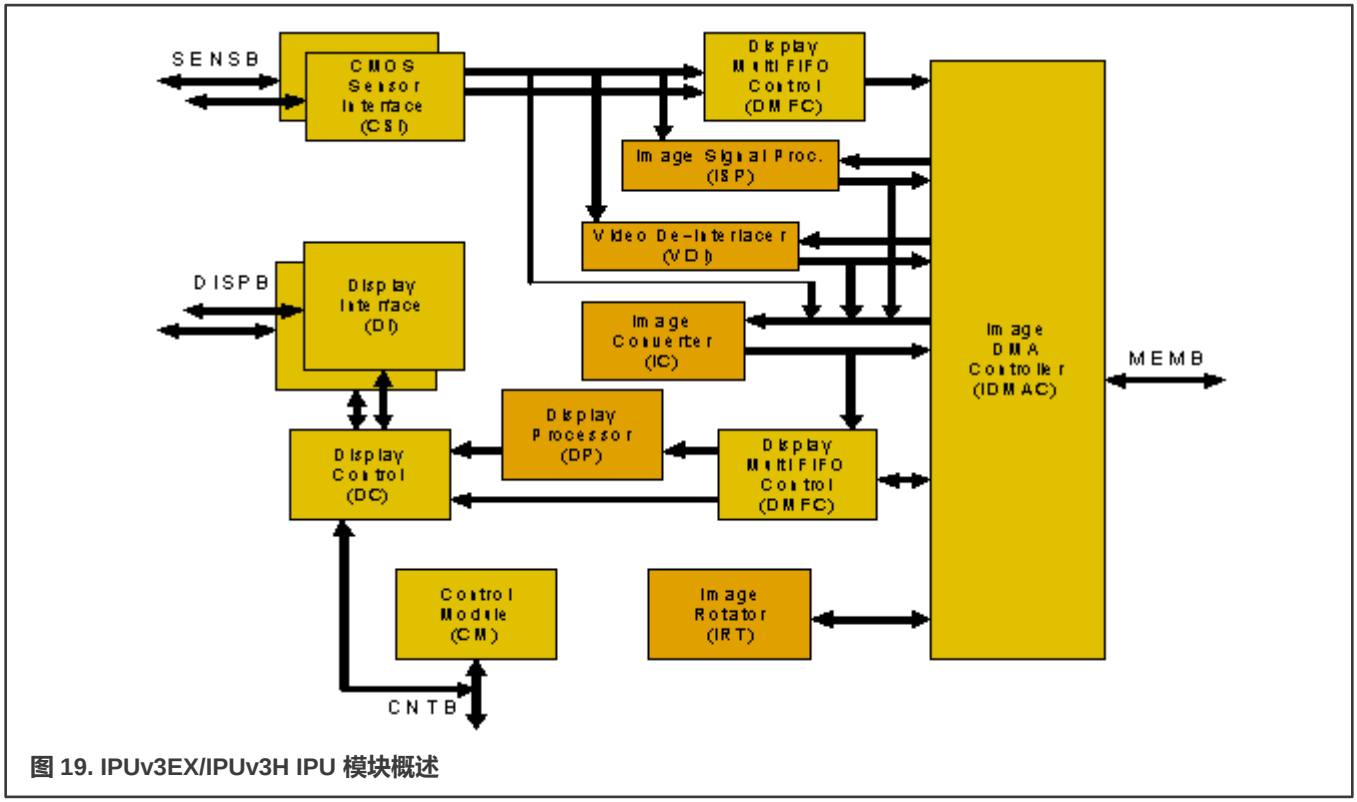


图 19. IPUv3EX/IPUv3H IPU 模块概述

6.3.2.3 软件操作

IPU 驱动是 Linux 内核中一个独立的驱动模块。

它由以下块的自定义内核级 API 组成：

- 同步帧缓冲驱动
- 显示接口（DI）
- 显示处理器（DP）
- 图像 DMA 控制器（IDMAC）
- CMOS 传感器接口（CSI）
- 图像转换器（IC）
- 预取/解析引擎/Gasket (PRE/PRG)

下图展示了不同图形/视频驱动与 IPU 之间的交互。

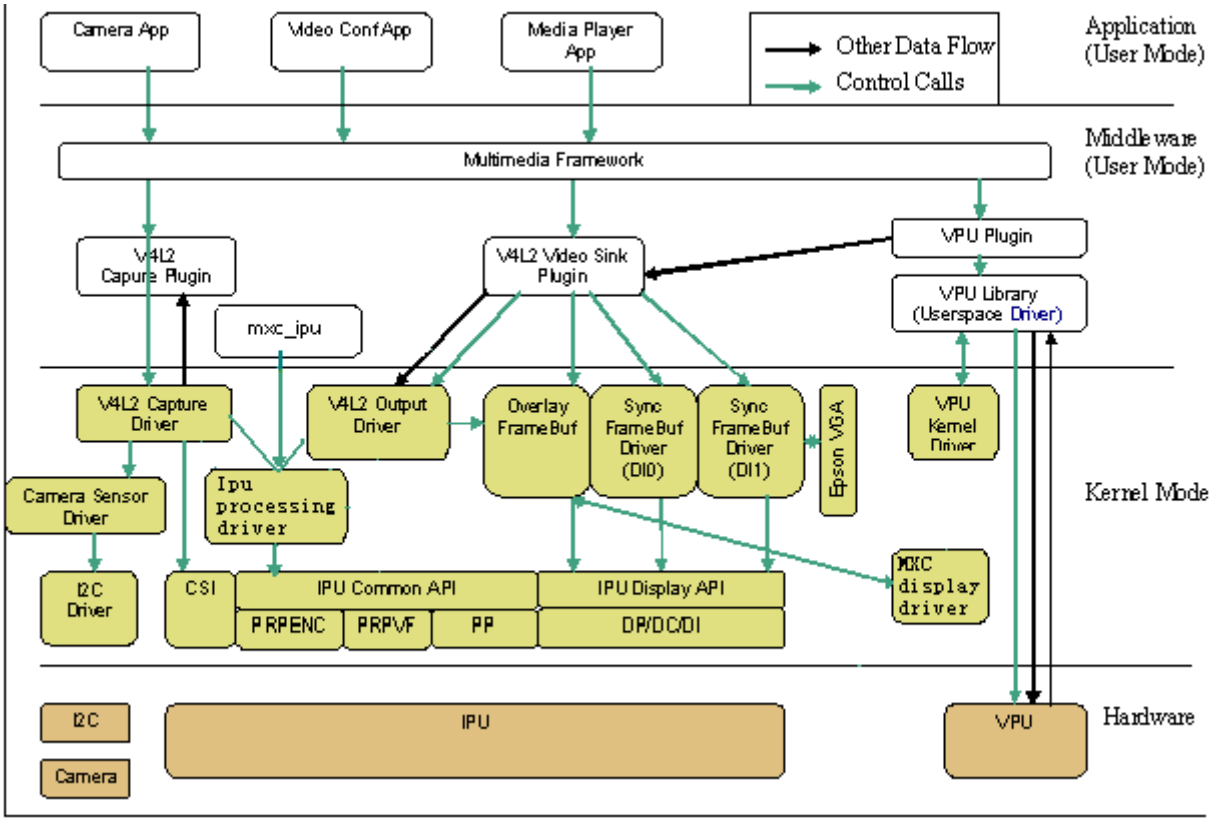


图 20. IPUv3 的图形/视频驱动软件交互

IPUv1 的驱动简称为 ipu。IPUv3 的驱动名称中包含 3 或 v3。IPU 驱动细分如下所示的驱动：

- 设备驱动——包括用于同步帧缓冲的帧缓冲驱动、用于显示器的帧缓冲驱动、用于 IPU 预处理的 V4L2 采集驱动、用于 IPU 后处理的 V4L2 输出驱动以及为用户空间或 V4L2 驱动提供系统接口的 IPU 处理驱动。帧缓冲设备驱动位于 `drivers/video/mxc`。V4L2 设备驱动位于 `drivers/media/platform/mxc`。
- MXC 显示驱动作为一个简单的框架，用于管理 IPU 和显示设备驱动（例如 LCD、LVDS、HDMI、MIPI 等）之间的交互

- 底层库例程——与 IPU 硬件寄存器对接的接口。它们从高级设备驱动获取输入，并与 IPU 硬件进行通信。底层库位于 Linux 内核的目录中。

6.3.2.4 IPU 帧缓冲驱动概述

帧缓冲设备为图形硬件提供了一种抽象。它代表帧缓冲视频硬件，并允许应用软件通过定义明确的接口访问图形硬件，因此软件不需要了解任何关于底层硬件寄存器的信息。

通过在内核配置的图形参数下选择帧缓冲选项来启用该驱动。为了补充帧缓冲驱动，内核构建器可能支持字体和启动标识。该设备依靠虚拟终端（VT）控制台从串行模式切换到图形模式。通过位于/dev 目录中的特殊设备节点（如/dev/fb*）访问该设备。fb0 通常是主帧缓冲区。

除了物理存储器分配和 LCD 屏配置外，通用内核视频 API 还用于设置颜色、调色板注册、图像块传输和存储器映射。IPU 从帧缓冲存储器读取原始像素数据，并将其发送至屏进行显示。

6.3.2.5 IPU 帧缓冲硬件操作

帧缓冲区与 IPU 硬件驱动模块进行交互。

6.3.2.6 IPU 帧缓冲软件操作

帧缓冲设备是一种存储器设备，例如/dev/mem，它具有与存储设备类似的功能。用户可以读取、写入、查找其中的某个位置，并对其执行 mmap()操作（主要用途）。区别在于，出现在特殊文件中的存储器不是整个存储器，而是某些视频硬件的帧缓冲区。

/dev/fb*还与若干 IOCTL 进行交互，允许用户查询和设置硬件相关信息。色图也通过 IOCTL 进行处理。如需了解存在哪些 IOCTL 及其使用的数据结构的更多信息，请参见 `include/uapi/linux/fb.h`。下面列出了一些 IOCTL 功能：

- 请求硬件的一般信息，例如屏幕存储器的名称、组织（平面、压缩像素等），以及屏幕存储器的地址和长度。
- 请求和更改有关硬件的可变信息，例如可见和虚拟几何、深度、色图格式、时间等。如果此信息发生变更，驱动会建议满足硬件功能的值（如果不可能，则硬件返回 EINVAL）。
- 获取和设置部分色图。通信是每像素 16 位（红、绿、蓝、透明度的值），以支持所有现有硬件。驱动执行将选项应用于硬件所需的所有计算（舍入到更少的位，可能丢弃透明度值）。

硬件抽象使应用程序实施更容易、更可移植。必须内置到应用程序中的唯一组件是屏幕组织（位平面或块状像素等），因为它直接处理帧缓冲区图像数据。

MXC 帧缓冲驱动（drivers/video/mxc/mxc_ipuv3_fb.c）与通用 Linux 帧缓冲驱动（`drivers/video/fbdev/core/fbmem.c`）密切交互。

6.3.2.7 同步帧缓冲驱动

同步帧缓冲屏幕驱动为同步 LCD 屏或无存储器的屏实施 Linux 标准帧缓冲驱动 API。同步帧缓冲屏幕驱动是与内核和用户级应用进行交互的高层内核视频驱动。通过选择内核配置中的图形支持设备驱动下的同步屏帧缓冲（Synchronous Panel Framebuffer）选项启用该驱动。为了补充帧缓冲驱动，内核构建器可能还支持字体和启动标识。具体取决于 VT 控制台从串行模式切换到图形模式。

除了物理存储器分配和 LCD 屏配置外，通用内核视频 API 还用于设置颜色、调色板注册、图像块传输和存储器映射。IPU 从帧缓冲存储器读取原始像素数据，并将其发送至屏进行显示。

帧缓冲驱动支持不同的屏作为内核配置选项。通过为屏设置结构定义新值，可增加对新屏的支持。

帧缓冲使用自定义 API 与 IPU 驱动进行交互，允许以下操作：

- 屏接口设置初始化
- LCD 刷新的 IPU 通道设置初始化
- 更改帧缓冲地址以支持双缓冲

支持以下功能：

- 可配置的屏幕分辨率
- 可配置的 RGB 16、24 或 32 位每像素帧缓冲
- 可配置的屏接口信号时序和极性
- 调色板/颜色转换管理
- 电源管理
- LCD 电源关闭/打开
- 启用/禁用 PRE/PRG 功能

用户应用利用通用视频 API（标准 Linux 帧缓冲驱动 API）来执行帧缓冲功能。其中包括：

- 获取屏幕信息，例如分辨率或扫描长度
- 使用 mmap 分配用户空间存储器，以执行直接 blitting 操作

另一个帧缓冲驱动支持第二个视频/图形平面。

6.3.2.8 IPU 背光驱动

IPU 驱动还控制背光。IPU 背光驱动实施屏的 IPU PWM 背光控制。它将/sys/class/backlight/pwm-backlight.0/brightness 下的 sys 控制文件导出到用户空间。默认背光强度值为 128。

6.3.2.9 IPU 设备驱动

IPU（处理）设备驱动提供图像处理功能：基于 IPUv3 中的 IC/IRT 模块调整大小/旋转/CSC/组合/反交错。

IPU 设备驱动基于任务，用户只需准备任务设置、对任务排队，然后阻塞等待任务完成。该驱动现在只支持阻塞方法，以后将增加非阻塞方法。任务结构如下所示：

```
struct ipu_task {
struct ipu_input input;
struct ipu_output output;
bool overlay_en;
struct ipu_overlay overlay;
#define IPU_TASK_PRIORITY_NORMAL 0
#define IPU_TASK_PRIORITY_HIGH  1
u8      priority;
#define IPU_TASK_ID_ANY 0
#define IPU_TASK_ID_VF  1
#define IPU_TASK_ID_PP  2
#define IPU_TASK_ID_MAX 3
```

```
u8      task_id;
int     timeout;
};
struct ipu_input
{ u32 width;
u32 height;
u32 format;
struct ipu_crop crop;
dma_addr_t paddr;
struct ipu_deinterlace deinterlace;
dma_addr_t paddr_n; /*valid when deinterlace enable*/
};
struct ipu_overlay
{ u32 width;
u32 height;
u32 format;
struct ipu_crop crop;
struct ipu_alpha alpha;
struct ipu_colorkey colorkey;
dma_addr_t paddr;
};
struct ipu_output
{ u32 width;
u32 height;
u32 format;
u8 rotate;
struct ipu_crop crop;
dma_addr_t paddr;
};
```

要准备任务，用户只需要填写 task.input、task.overlay（如果需要组合）和 task.output 参数，如果是在内核级别（例如 V4L2 驱动），则通过 `int ipu_queue_task(struct ipu_task *task)`对任务进行排队；如果是在应用级别，则通过/dev/mxc_ipu 下的 IPU_QUEUE_TASK ioctl。

6.3.2.10 源代码结构

与 IPU、Sensor、V4L2 和 Panel 驱动有关的源文件位于以下文件夹中。

- drivers/mxc/ipu3
- drivers/video/mxc
- drivers/video/fbdev/mxc
- drivers/video/backlight

如需了解有关 IPU V4L2 驱动文件的更多信息，请参见 V4L2 章节

表 58. IPU 驱动文件

文件	说明
driveers/mxc/ipu3/ipu_common.c	IPU 常用库函数
driveers/mxc/ipu3/ipu_common.c	IPU 常用库函数
drivers/mxc/ipu3/ipu_ic.c	IPU IC 基础驱动
drivers/mxc/ipu3/ipu_device.c	IPU 驱动设备接口和 fops 功能。
drivers/mxc/ipu3/ipu_capture.c	IPU CSI capture 基础驱动

下页继续.....

表 58. IPU 驱动文件（续）

文件	说明
drivers/mxc/ipu3/ipu_disp.c	IPU 显示功能
drivers/mxc/ipu3/ipu_calc_stripes_sizes.c	ipu_device.c 的多条带方法函数
drivers/mxc/ipu3/pre.c	i.MX 6 QuadPlus 预取/解析 Engine 驱动
drivers/mxc/ipu3/prg.c	i.MX 6 QuadPlus 预取/解析 Gasket 驱动
drivers/mxc/ipu3/mxc_ipuv3_fb.c	同步帧缓冲驱动
drivers/mxc/ipu3/vdoa.c	VDOA 后处理驱动，由 ipu_device.c 使用
drivers/video/fbdev/mxc/mxc_lcdif.c	支持 CLAA-WVGA 和 SEIKO-WVGA LCD 的显示驱动
drivers/video/fbdev/mxc/mxc_hdmi.c	HDMI 接口的显示驱动
drivers/video/fbdev/mxc/ldb.c	片上 LVDS 同步帧缓冲驱动
drivers/video/fbdev/mxc/mxc_dispdrv.c	用于同步帧缓冲的显示驱动框架
drivers/video/fbdev/mxc/mxc_edid.c	EDID 驱动

表 59 列出了与 IPU 和 Panel 驱动有关的头文件。

表 59. IPU 全局头文件

文件	说明
drivers/mxc/ipu3/ipu_param_mem.h	用于 IPU 参数存储器访问的 Hellper 函数
drivers/mxc/ipu3/ipu_prv.h	用于预处理驱动的头文件
drivers/mxc/ipu3/ipu_regs.h	IPU 寄存器定义
drivers/mxc/ipu3/pre-regs.h	预取/解析 Engine 寄存器定义
drivers/mxc/ipu3/prg-regs.h	预取/解析 Gasket 寄存器定义
drivers/mxc/ipu3/vdoa.h	VDOA 驱动的头文件
drivers/video/fbdev/mxc/mxc_dispdrv.h	显示驱动的头文件
include/linux/uapi/mxcfb.h	同步帧缓冲驱动的头文件
include/linux/uapi/ipu.h	IPU API 的头文件

6.3.2.11 菜单配置选项

为 IPU 模块提供了以下 Linux 内核配置选项。

在菜单配置中启用以下模块：

- CONFIG_MXC_IPU_V3——包括图像处理单元支持。在 menuconfig 中，此选项位于：Device Drivers > MXC support drivers > Image Processing Unit Driver

  默认情况下，对于所有架构，此选项都是 Y。

  如果 ARCH_MXC 为 true，则设置 CONFIG_MXC_IPU_V3。

- CONFIG_MXC_IPU_V3_PRG——它支持 IPUv3 预取 gasket 引擎，以支持 IPUv3 和预取引擎（PRE）之间的双缓冲握手控制，侦听 AXI 接口，将显示刷新请求发送到存储器，并修改请求地址以获取 OCRAM 中的双缓冲块行。

Device Drivers > MXC support drivers > i.MX IPUv3 prefetch gasket engine

此选项取决于 CONFIG_MXC_IPU_V3 和 CONFIG_MXC_IPU_V3_PRE。

- CONFIG_MXC_IPU_V3_PRE——支持 IPUv3 预取引擎，以提高系统存储器性能。该引擎能够将平铺像素格式的帧缓冲区解析为线性。

  Device Drivers > MXC support drivers > i.MX IPUv3 prefetch engine

  此选项取决于 CONFIG_MXC_IPU_V3。启用此选项会选择 CONFIG_MXC_IPU_V3_PRG。

- CONFIG_MXC_CAMERA_OV5640_MIPI——适用于 OV 5640 mipi 传感器驱动和用例驱动的选项。此选项取决于 VIDEO_MXC_CAPTURE 选项。在 menuconfig 中，此选项位于以下位置：

  Device Drivers > Multimedia support > V4L platform devices > MXC Video For Linux Video Capture > MXC Camera/V4L2 PRP Features support > OmniVision 5640 Camera support using mipi

- CONFIG_MXC_CAMERA_OV5640——适用于 OV5640 传感器驱动和用例驱动的选项。此选项取决于 VIDEO_MXC_CAPTURE 选项。在 menuconfig 中，此选项位于以下位置：

  Device Drivers > Multimedia platform > V4L platform devices > MXC Video For Linux Video Capture > MXC Camera/V4L2 PRP Features support > OmniVision ov5640 camera support

  一次只能安装一个传感器。

- CONFIG_MXC_IPU_PRP_VF_SDC——适用于 IPU 的选项（此处的 > 符号表示硬件块之间的数据流方向）：

  CSI > IC > MEM MEM > IC (PRP VF) > MEM

  用于非智能传感器的用例驱动

  CSI > IC(PRP VF) > MEM

  用于智能传感器。在 menuconfig 中，此选项位于以下位置：

  Multimedia devices > Video capture adapters > MXC Video For Linux Camera > MXC Camera/V4L2 PRP Features support> Pre-Processor VF SDC library

  默认情况下，此选项对所有驱动均为 M。

- CONFIG_MXC_IPU_PRP_ENC——适用于 IPU 的选项：

  对于非智能传感器的用例驱动

  CSI > IC > MEM MEM > IC (PRP ENC) > MEM

  对于智能传感器为

  CSI > IC(PRP ENC) > MEM

  在 menuconfig 中，此选项位于以下位置：

  Device Drivers > Multimedia Devices > Video capture adapters > MXC Video For Linux Camera > MXC Camera/V4L2 PRP Features support > Pre-processor Encoder library

  默认情况下，此选项对所有驱动均为 M。

- CONFIG_VIDEO_MXC_CAMERA——这是适用于 V4L2 capture 驱动的配置选项。此选项取决于以下表达式：

  VIDEO_DEV && MXC_IPU && MXC_IPU_PRP_VF_SDC && MXC_IPU_PRP_ENC

  在 menuconfig 中，此选项位于以下位置：

  Device Drivers > Multimedia devices > Video capture adapters > MXC Video For Linux Camera

  默认情况下，此选项对所有驱动均为 M。

- CONFIG_VIDEO_MXC_OUTPUT——这是适用于 V4L2 Output 驱动的配置选项。此选项取决于 VIDEO_DEV && MXC_IPU 选项。在 menuconfig 中，此选项位于以下位置：

Device Drivers > Multimedia devices > Video capture adapters > MXC Video for Linux Video Output

默认情况下，此选项对所有驱动均为 Y。

- CONFIG_FB——这是在 Linux 内核中包含帧缓冲支持的配置选项。在 menuconfig 中，此选项位于以下位置：

  Device Drivers > Graphics support > Support for frame buffer devices

  默认情况下，此选项对所有架构均为 Y。

- CONFIG_FB_MXC——这是适用于 MXC 帧缓冲驱动的配置选项。此选项取决于 CONFIG_FB 选项。在 menuconfig 中，此选项位于以下位置：

  Device Drivers > Graphics support > MXC Framebuffer support

  默认情况下，此选项对所有架构均为 Y。

- CONFIG_FB_MXC_SYNC_PANEL——这是选择同步屏帧缓冲的配置选项。此选项取决于 CONFIG_FB_MXC 选项。在 menuconfig 中，此选项位于以下位置：

  Device Drivers > Graphics support > MXC Framebuffer support > Synchronous Panel Framebuffer

  默认情况下，此选项对所有架构均为 Y。

- CONFIG_FB_MXC_LDB——此配置选项选择 i.MX 6 芯片上的 LVDS 模块。此选项取决于 CONFIG_FB_MXC_SYNC_PANEL 和 CONFIG_MXC_IPUV3 || FB_MXS 选项。在 menuconfig 中，此选项位于以下位置：

  Device Drivers > Graphics support > MXC Framebuffer support > Synchronous Panel Framebuffer > MXC LDB

- CONFIG_FB_MXC_SII9022——此配置选项选择 SII9022 HDMI 芯片。 此选项取决于 CONFIG_FB_MXC_SYNC_PANEL 选项。 在 menuconfig 中，此选项位于以下位置：

  Device Drivers > Graphics support > MXC Framebuffer support > Synchronous Panel Framebuffer > Si Image SII9022 DVI/HDMI Interface Chip

6.3.3 Pixel Pipeline（PxP）

6.3.3.1 介绍

PxP 是一个显示控制器，与 EPDC 显示接口一起工作。Pixel Pipeline（PxP）DMA 引擎驱动提供了一个独特的 API，作为 dmaengine 客户端，可以减轻不同硬件分流引擎实施的细节。通常，PxP DMA-ENGINE 驱动的用户包括 EPDC 驱动、V4L2 Output 驱动和 PxP 用户空间库。

PxP 驱动使用 PxP 寄存器与硬件进行交互。如需了解硬件操作的详细信息，请参见与 SoC 有关的《应用处理器参考手册》文档。

6.3.3.2 软件操作

PxP IP 有不同的版本。为了简化 i.MX 7Dual 上使用的新版 PxP 的维护，该版本主要具有面向 EPDC 的新功能，如硬件冲突检测、硬件中的 E Ink Gen II 波形算法（REAGL/-D）处理和硬件抖动支持等，含有不同的驱动（drivers/dma/pxp/pxp_dma_v3.c）。然而，每个版本都使用 DMA Engine 框架。

6.3.3.3 关键数据结构

PxP DMA Engine 驱动实施取决于 DMA Engine 框架。DMA Engine 框架中有 3 个重要的结构，它们通过 PxP 驱动进行了扩展：struct dma_device、struct dma_chan、struct dma_async_tx_descriptor。PxP 驱动实施了几个回调函数，当 DMA 从机（客户端）与 DMA Engine 进行交互时，DMA Engine 框架（或 DMA 从机）会调用这些函数。

PxP 驱动在 struct dma_device 中实施以下回调函数：

device_alloc_chan_resources / allocate resources and descriptors */*

device_free_chan_resources / release DMA channel's resources */*

device_tx_status / poll for transaction completion */*

device_issue_pending / push pending transactions to hardware */*

and,

device_prep_slave_sg / prepares a slave DMA operation */*

device_terminate_all/ manipulate all pending operations on a channel, returns zero or error code */*

前 4 个函数由 DMA Engine 框架使用，后两个函数由 DMA 从机（DMA 客户端）使用。特别是，*device_issue_pending* 用于触发 PxP 操作启动。

PxP DMA 驱动在 *struct dma_async_tx_descriptor* 中实现接口 *tx_submit*，用于准备将由引擎执行的描述符。在 pxp_tx_submit 中接收到任务时，不会立即配置和执行任务。而是将任务添加到任务队列中，并允许函数调用立即返回。

6.3.3.4 通道管理

虽然 ePxP 在硬件中没有多个通道，但驱动支持虚拟通道。这为多实例/客户端设计提供了灵活性。在任何时候，用户都可以调用 *dma_request_channel*()来获得一个空闲通道，然后用几个描述符配置该通道。每个输入平面和输出平面都需要一个描述符。当不再使用 PxP 时，应调用 *dma_release_channel*()来释放通道。通道管理的具体元件由驱动进行处理，对客户端透明。

6.3.3.5 描述符管理

DMA Engine 根据描述符处理任务。一个 DMA 通道通常与多个描述符相关联。描述符是在分流引擎驱动的控制下回收的资源，在操作完成后会被重用。扩展的 TX 描述符包（pxp_tx_desc）允许用户将 PxP 配置信息传递给驱动。其中包括 PxP 执行处理任务所需的一切。

6.3.3.6 完成通知

应用可通过两种方式接收 PxP 操作已完成的通知。

- 调用 dma_wait_for_async_tx()。此调用导致 CPU 在轮询操作是否完成时旋转。
- 指定完成回调。

建议采用后一种方法。PxP 操作完成后，可以检索 PxP 输出缓冲区数据。

如需了解 DMA Engine 框架的一般信息，请参见 Linux 内核源代码树中的 *Documentation/dmaengine.txt*。

6.3.3.7 限制

- 驱动目前不支持以传统方式使用的散点列表对象。驱动目前没有使用 scatterlist 参数对象来提供一系列存储器源和目标，而是使用它为一次传输提供输入和输出缓冲区（以及覆盖缓冲区，如需要）。
- PxP 驱动可能无法正确执行一系列快速排队的传输。建议等待每次传输完成后再提交新的传输请求。

6.3.3.8 菜单配置选项

对该模块提供了以下 Linux 内核配置选项：

对于 i.MX 7Dual，选择“Device Drivers > DMA Engine support > [*] MXC PxP V3 support > [*] MXC PxP Client Device”

对于 i.MX 6，选择“Device Drivers > DMA Engine support > [*] MXC PxP V2 support > [*] MXC PxP Client Device”

6.3.3.9 源代码结构

PxP 驱动源代码位于 drivers/dma/pxp。

表 60. Pxp 源代码

文件	说明
drivers/dma/pxp/pxp_device.c	PxP 设备
drivers/dma/pxp/pxp_dma_v2.c	用于 i.MX 6 的 PxP DMA
drivers/dma/pxp/pxp_dma_v3.c	用于 i.MX 7 的 PxP DMA
drivers/dma/pxp/regs-pxp_v2.h	用于 i.MX 6 的 PxP 寄存器
drivers/dma/pxp/regs-pxp_v3.h	用于 i.MX 7 的 Pxp 寄存器
drivers/dma/pxp/regs-pxp_v3.h	用于 i.MX 7 的 Pxp 寄存器
include/linux/drivers/dma/pxp/pxp_dma_v3.c	用于 i.MX 7 的 PxP 设备
include/linux/pxp_dma.h	PxP DMA 内核头文件
include/linux/pxp_device.h	PxP Device 内核头文件
include/linux/uapi/pxp_dma.h	PxP DMA 用户空间头文件
include/linux/uapi/pxp_device.h	PxP Device 用户空间头文件
drivers/media/platform/mxc/output/mxc_pxp_v4l2.c	PxP V4L2
drivers/media/platform/mxc/output/mxc_pxp_v4l2.c	PxP V4L2 头文件
drivers/media/platform/mxc/output/mxc_vout.c	i.MX V4L2 Output 驱动

6.3.4 eLCDIF 帧缓冲

6.3.4.1 介绍

eLCDIF 是一个与 Parallel LCD 接口一起工作的显示控制器。该驱动被用作一个显示子系统驱动，可以是帧缓冲或 DRM，控制通用 LCD 底层操作，从而实现底层硬件控制。只测试了 ELCDIF 的 DOTCLK 模式，因此理论上 ELCDIF 帧缓冲驱动可以与同步 LCD 屏驱动配合工作，以支持帧缓冲设备。同步 LCD 驱动以灵活、可扩展的方式组织，并从任何特定的同步 LCD 屏支持中抽象出来。如需支持其他同步 LCD 屏，用户可以参考现有的同步 LCD 驱动编写同步 LCD 驱动。

6.3.4.2 软件操作

对于作为帧缓冲驱动的 eLCDIF，帧缓冲设备是类似于/dev/mem 的存储器设备。可以对其进行读取、写入操作，也可以使用 mmap()查找和映射其中的某个位置。区别在于，用户可用的存储器不是整个分配的存储器，而是视频硬件的帧缓冲区。用户通过特殊的设备节点访问该设备，这些节点通常位于/dev 目录/dev/fb*中。/dev/fb*还有若干 IOCTL 作用于它，还可通过它们查询和设置有关硬件的信息。色图处理也通过 IOCTL 操作。如需获取关于存在哪些 IOCTL 以及使用了哪些数据结构的更多信息，请参见 linux/fb.h。

i.MX ELCDIF 帧缓冲驱动实施是从实际硬件中抽象出来的。默认屏驱动由平台数据中定义的视频模式拾取，或在探测期间通过“video=mxc_elcdif_fb:resolution，bpp=bits_per_pixel”内核启动命令传入。分辨率应采用通用帧缓冲视频模式，bits_per_pixel 应为帧缓冲区的颜色深度。

6.3.4.3 菜单配置选项

以下菜单选项将配置 MXC ELCDIF 帧缓冲驱动。此选项取决于 FB 和（ARCH_MXS || ARCH_MXC）。

Frame buffer Devices > MXS LCD framebuffer support (CONFIG_FB_MXS)

6.3.4.4 源代码结构

帧缓冲区的源代码位于 drivres/video/fbdev/mxc 中，drm 驱动位于 drivers/gpu/drm/imx/lcdif。

表 61. ELCIF 源代码

文件	说明
drivers/video/fbdev/mxsfb.c	ELCDIF 帧缓冲驱动
drivers/video/fbdev/mxc/mxc_lcdif.c	ELCDIF 帧缓冲驱动
drivers/gpu/drm/imx/lcdif/lcdif-crtc.c	ELCDIF DRM 身份验证
drivers/gpu/drm/imx/lcdif/lcdif-kms.c	ELCDIF DRM KMS
drivers/gpu/drm/imx/lcdif/lcdif-kms.h	ELCDIF DRM KMS 头文件
drivers/gpu/drm/imx/lcdif/lcdif-plane.c	ELCDIF DRM 平面
drivers/gpu/drm/imx/lcdif/lcdif-plane.h	ELCDIF DRM 平面头文件

6.3.5 显示控制子系统（DCSS）

6.3.5.1 介绍

显示控制子系统（DCSS）是 i.MX 8M Quad 的显示控件，通过 DRM 显示框架进行集成。DCSS 提供了一种将存储器中的帧缓冲区显示给 UltraHD 或 HDTV 的机制，能够将 3 层图形或视频覆盖组合到 HDMI 输出中。DCSS 控制器的主要功能包括：

- 最多支持 3 层图形或视频
  - 任意偏移量
  - 1 个平面可以是支持 8 位 alpha 的图形
  - 将 1920x1080p60 视频或图形升级到 3840x2160p60
  - 将 3840x2160p30 视频缩小到 1920x1080p30 或 1280x720p30
- HDR 支持：
  - 具有 2084 和 2020 颜色空间的 HDR10
  - Dolby Vision 单层和双层格式
  - HLG
- HDMI 2.0a 支持一台显示器：
  - 分辨率：640x480p60、720x480p60、1280x720p60、1920x1080p60、3840x2160p60、4096x2160p60
  - HDCP 2.2 和 HDCP 1.4
- 像素时钟高达 596 MHz
- 输出也可转至 MIPI DSI 输出
- 帧缓冲区压缩——缓冲区的无损压缩

6.3.5.2 源代码结构

DCSS drm 驱动位于 drivers/gpu/drm/imx/dcss 中，DCSS 核心驱动位于 drivers/gpu/imx/dcss 中

表 62. DCSS 驱动源代码

文件	说明
drivers/gpu/drm/imx/dcss/dcss-plane	DRM DCSS 平面
drivers/gpu/drm/imx/dcss/dcss-kms	DRM DCSS KMS
drivers/gpu/drm/imx/dcss/dcss-crtc	DRM DCSS CRTC 头文件
drivers/gpu/drm/imx/dcss/dcss-dec400d.c	DCSS dec400d
drivers/gpu/drm/imx/dcss/dcss-scaler	DCSS Scaler
drivers/gpu/drm/imx/dcss/dcss-ss.c	DCSS ss
drivers/gpu/drm/imx/dcss/dcss-hdr10.c	DCSS hdr10
drivers/gpu/drm/imx/dcss/dcss-wtsc1.c	DCSS wtsc1
drivers/gpu/drm/imx/dcss/dcss-dtg.c	DCSS dtg
drivers/gpu/drm/imx/dcss/dcss-ctx1d.c	DCSS ctx1d
drivers/gpu/drm/imx/dcss/dcss-dtrc.c	DCSS DTRC
drivers/gpu/drm/imx/dcss/dcss-dpr.c	DCSS ctx1d
drivers/gpu/drm/imx/dcss/dcss-blkctr.c	DCSS ctx1d

6.3.6 DCNANO

6.3.6.1 介绍

LCDIF 是一种高性能图形核心，可用于从帧缓冲区读取渲染的图像。除了提供硬件光标模式外，显示控制器还执行格式转换、抖动和伽马校正。显示控制器的主要功能包括：

- 视频时序生成
  - HSYNC、VSYNC、DE 信号
  - 可编程定时器
- MIPI 显示协议
  - 显示像素接口 2 (DPI-2) 格式
  - 支持 DPI 24 位、18 位（2 个配置）和 16 位（3 个配置）
  - （可选）显示总线接口 2.0 (DBI-2)
- 显示接口
  - 具有 24 位 Data、HSync、VSync、Data 使能的并行像素输出
  - 轻松适应外部串行逻辑，例如 HDMI
- 显示器
  - 显示器尺寸为 1024x480
  - 同步和消隐信号
  - 伽玛和抖动表

- 输入格式
  - ARGB2101010/ARGB8888/ARGB1555/RGB565/ARGB4444
  - YUV422 封装和半平面（YUY2, UYVY）
- 格式转换
  - 接受多种 RGB 格式的像素输入
  - 像素输出为多种格式的 24 位 RGB
- 输出格式
  - RGB888/DPI_D16CFG1/DPI_D16CFG2/DPI_D16CFG3/DPI_D18CFG1/ DPI_D18CFG2/ DPI_D24
- 硬件光标
  - 支持 ARGB888 和 Mask 光标格式
- 颜色
  - 使用坐标生成器进行覆盖
  - Alpha 混合：8 种 Porter Duff 混合模式
- 抖动和伽马校正
  - 单独的抖动查找表
  - 单独的伽马校正查找表

6.3.6.2 源代码结构

DCNANO drm 驱动位于 drivers/gpu/drm/imx/dcnano 中。

表 63. DCNANO 驱动源代码

文件	说明
drivers/gpu/drm/imx/dcnano/dcnano-crtc.c	DRM DCNANO CRTC
drivers/gpu/drm/imx/dcnano/dcnano-drv.c	DRM DCNANO 核心
drivers/gpu/drm/imx/dcnano/dcnano-drv.h	DRM DCNANO 头文件
drivers/gpu/drm/imx/dcnano/dcnano-kms.c	DRM DCNANO KMS
drivers/gpu/drm/imx/dcnano/dcnano-plane.c	DRM DCNANO 平面
drivers/gpu/drm/imx/dcnano/dcnano-reg.h	DCNANO 寄存器头文件

6.4 显示接口

6.4.1 Parallel LCD 接口

6.4.1.1 介绍

Parallel 接口支持 LCD 显示。通过显示控制器支持 Parallel Display 接口，并使用显示框架实施，对于 i.MX 6 和 i.MX 7 为 fbdev 框架，对于 i.MX8 为 drm 框架。

以下控制器支持 Parallel 接口

- 带 IPU 的 i.MX 上的 IPU
- 所有 i.MX8 上的 DPU
- 带 PxP 的 i.MX 上的 ElCDIF

Parallel 接口支持启用 Parallel 接口的 i.MX SoC 上的至少一个端口，并支持带 IPU 的 i.MX 的两个端口。启用的 SoC 的每端口比特率从 18 位到 24 位不等。在带有 IPU 的 i.MX 6 上，Parallel 接口还支持用于显示刷新的同步模式和用于存储器的异步模式，并且非常灵活，可以无胶连接到无 RAM 显示器、显示控制器和电视编码器。

6.4.2 MIPI DSI Interface

6.4.2.1 介绍

MIPI 显示接口（MIPI DSI）是一个驱动接口，用于与显示屏上的 MIPI 设备控制器进行通信。MIPI DSI 显示屏驱动提供了一个接口，可通过 MIPI DSI 对显示屏进行配置。

MIPI DSI Interface 是一个带有多通道 D-PHY 的数字核心，它实施 MIPI DSI 规范中定义的所有协议功能，在系统和符合 MIPI DSI 的显示器之间提供了一个接口。本文概要介绍了 MIPI DSI，但规范仅适用于 MIPI 成员。

MIPI DSI 模块在主处理器和显示模块之间提供了高速串行接口。与并行总线相比，它性能更高、功耗更低、EMI 更少，引脚也更少。它旨在与标准 MIPI DSI 协议兼容，并基于现有的 MIPI DPI-2、MIPI DBI-2 和 MIPI DCS 标准构建。该模块向外设发送像素或命令，并从外设读取状态或像素信息。MIPI DSI 序列化所有像素数据、命令和事件，包含两种基本模式：命令模式和视频模式。它使用命令模式将寄存器和存储器写入显示控制器，同时读取显示模块状态信息。它还通过视频模式以高速模式将实时像素流从主机传输到外设，并在出现错误时生成中断。

对于 i.MX，各种驱动都支持 MIPI DSI，这些驱动将在以下章节中介绍。MIPI DSI 驱动支持以下功能：

- MIPI DSI 通信协议
- MIPI DSI 命令模式和视频模式
- MIPI DCS 命令操作

MIPI DSI 驱动对 i.MX 6 和 i.MX 7 使用帧缓冲驱动，对 i.MX8 使用 drm 驱动。这两种驱动都支持以下内容。

- 驱动不向用户接口公开，而是通过 drm 或帧缓冲接口。
- MIPI DSI IP 驱动——底层接口，用于与显示屏上的 MIPI 设备控制器进行通信
- MIPI DSI 显示屏驱动提供了一个接口，可通过 MIPI DSI 配置显示屏

该驱动启用与平台相关的调节器和时钟。它请求与操作系统有关的系统资源，并注册缓冲区事件通知程序以执行隐藏/恢复操作。该驱动根据 MIPI DSI 显示屏初始化 MIPI D-PHY 并配置 MIPI DSI IP。

MIPI DSI 驱动支持以下功能：

- 兼容 MIPI 联盟 DSI 规范版本 1.01.0r11
- 兼容 MIPI 联盟 D-PHY 规范版本 1.00.00
- 支持 1 到 4 个 D-PHY 数据通道，具体取决于 SoC 功能。
- 通过数据通道 0 支持双向通信和逃生模式
- 可编程显示分辨率
- 视频模式像素格式、16bpp (565 RGB)、18bpp (666 RGB)压缩、18 bpp (666 RGB) 松散、24bpp (888 RGB)。
- 支持所有通用命令的传输
- 支持 ECC 和校验和功能
- 支持传输结束数据包（EoTp）
- 支持超低功耗模式
- 支持 PLL 的 PMS 控制接口，以配置字节时钟频率
- 支持预分频器从字节时钟生成输出时钟（escape clock）

端口和通道的数量在位于 arch/arm/boot/dts 和 arch/arm64/boot/dts 的设备树中指定。

6.4.2.2 软件操作

MIPI DSI 驱动包含两个部分：MIPI DSI IP 驱动和 MIPI DSI 显示屏驱动。

MIPI DSI IP 驱动有一个名为 mipi_dsi_info 的私有结构。MIPI DSI IP 连接的实例在 int dev_id 字段中描述，而 IPU 内的 DI 实例在 int disp_id 字段中描述。

在启动过程中，加载驱动时，通过字段 struct mxc_dispdrv_handle 将 MIPI DSI IP 驱动注册到帧缓冲驱动。它还向帧缓冲核心注册帧缓冲事件通知程序，以执行显示屏隐藏/恢复操作。通过显示屏回调接收字段 struct fb_videomode *mode 和 struct mipi_lcd_config *lcd_config。MIPI DSI IP 需要此信息才能配置 MIPI DSI 硬件寄存器。

在初始化 MIPI DSI IP 控制器和显示模块后，MIPI DSI IP 通过 DPI-2 接口从 IPU 获取像素流，并通过高速数据链路对像素数据和视频事件进行序列化，以供显示。当发生帧缓冲区隐藏/恢复事件时，将调用已注册的通知程序进入/离开低功耗模式。

MIPI DSI IP 驱动为 MIPI DSI 显示屏驱动提供 3 个 API，用于配置显示模块。

驱动使用 MIPI DSI IP 驱动提供的 API 读取/写入显示模块寄存器。通常，显示屏上集成了一个 MIPI DSI 从控制器。上电复位后，需要根据制造商的规范，通过标准 MIPI DCS 命令或 MIPI DSI Generic 命令配置 MIPI DSI 显示屏。

6.4.2.3 源代码结构

下表列出了 drivers/video/fbdev/mxc 中的 MIPI DSI 驱动源文件。

表 64. MIPI DSI 驱动文件

文件	说明
drveirs/video/fbdev/mxc/mipi_dsi.c	MIPI DSI IP 帧缓冲驱动源文件
drivers/video/fbdev/mxc/mipi_dsi.h	MIPI DSI IP 帧缓冲驱动头文件
drivers/video/fbdev/mxc/mxcfb_hx8369_wvga.c	MIPI DSI 帧缓冲显示屏驱动源文件
drivers/video/fbdev/mxc/mipi_dsi_samsung.c	MIPI DSI 帧缓冲三星源文件
drivers/video/fbdev/mxc/mipi_dsi_northwest.c	MIPI DSI 帧缓冲 Northwest 源文件
drivers/video/fbdev/mxc/mxcfb_hx8363_wvga.c	i.MX 7 帧缓冲 Truly WVGA Panel TFT3P5581E
drivers/video/fbdev/mxc/mxcfb_hx8369_wvga.c	i.MX 6 帧缓冲 Truly WVGA 同步屏
drivers/video/fbdev/mxc/mxcfb_otm808b_wvga.c	Truly 帧缓冲 WVGA Panel TFT3P5079E
drivers/gpu/drm/imx/sec_mipi_dsmi-imx.c	三星 DRM 驱动
drivers/gpu/drm/imx/nwl_dsi-imx.c	Northwest DRM 驱动

6.4.2.4 菜单配置选项

在菜单配置中启用以下模块：

Device Drivers > Graphics support > MXC Framebuffer support > Synchronous Panel Framebuffer > MXC MIPI_DSI

Device Drivers > Graphics support > MXC Framebuffer support > Synchronous Panel Framebuffer > MXC MIPI_DSI_SAMSUNG

Device Drivers > Graphics support > DRM Support for Freescale i.MX > Support for Northwest Logic MIPI DSI displays

Device Drivers > Graphics support > DRM Support for Freescale i.MX > Support for Samsung MIPI DSIM displays

6.4.3 LVDS 接口

6.4.3.1 介绍

低压差分信号（LVDS）支持高带宽和高清晰度图形，以及低功耗的快速帧速率。该实施使用了一组导线，其中一对导线中的每根导线都携带另一根导线的反向信号。这会减少干扰和噪音。LVDS 接口使用 4、6 或 8 对导线，其他的导线带有时钟线和地线。

LVDS 接口的目的是支持同步 RGB 数据通过 LVDS 接口从显示控制器流向外部显示设备。

该支持涵盖了这些活动的各个方面：

1. 与相关设备的连接——带有 LVDS 接收器的显示器。
2. 外接显示接收器和 LVDS 显示标准要求的数据排列。
3. 同步和控制功能。

LVDS 接口支持下面所示的多个控制器。

- LDB——带 IPU 的 i.MX 6 上的两个（double）
- i.MX 8QuadMax 上的 Mixel
- i.MX 8QuadXPlus 上的 Mixel Combo

LVDS 驱动与支持的显示框架配合工作，对于 i.MX 6 和 i.MX 7 为帧缓冲框架，对于 i.MX 8 为 drm 框架。

LVDS 接口具有以下支持结构

- 通道——通常是 2 个通道
- 每个通道支持多个数据对
- 数据像素率可能因每个数据对而异
- HSYNC、VSYNC、DE 的控制信号

LVDS 接口支持以下显示。

- IT6263 LVDS 到 HDMI 桥——使用我们的 LDB 驱动实现
- LVDS 双通道屏

LVDS 的相关标准如下所示。

- PHY 标准：ANSI EIA-644A
- 显示协议标准：
  - SPWG－标准屏工作组规范 3.8（2007 年 5 月）
  - VESA PSWG－屏标准化工作组，一组使用 LVDS 的屏标准。
  - JEIDA/JEITA DISM 标准 JEIDA-59-1999
  - OpenLDI（国家标准）－1999 年 5 月 13 日 0.95 版。*仅*支持不平衡操作模式（与绝大多数 LCD 供应商一致）。

通过 i.MX 6 和 i.MX 7 上的帧缓冲框架以及 i.MX 8 上的 drm 框架支持 LVDS 接口。

6.4.3.2 软件操作

如果驱动是内置的且设备树状态设置为“OK”，则 LVDS 驱动可以正常工作。

正确探测 LVDS 设备驱动后，驱动将为 LVDS 配置时钟。LVDS 驱动探测功能将默认模式设置为 1080p60。LVDS 通道映射模式和位映射模式设置为使用 30 位 JEIDA 模式。

驱动按以下步骤启用 LVDS 通道：

1. 开启 LVDS 的电源。
2. 设置 ldb_di_clk 的父时钟和父时钟的速率。
3. 设置 ldb_di_clk 的速率。
4. 启用 ldb_di_clk 及其父时钟。
5. 将 LVDS 设置为合适的模式，包括显示信号的极性、通道映射模式和位映射模式。
6. 启用相关的 i.MX LVDS 通道。

6.4.3.3 源代码结构

表 65. LVDS 源文件

文件	说明
drivers/video/fbdev/mxc/imx_lvds.c	LVDS 驱动
drivers/gpu/drm/imx/imx*-ldb.c	LDB 驱动，带 Mixel 信息

6.4.3.4 菜单配置选项

在菜单配置中启用以下模块：

Device Drivers > Graphics support > DRM Support for Freescale i.MX > Support for LVDS displays

6.4.4 LVDS 显示桥（LDB）

6.4.4.1 介绍

本节介绍 LVDS 显示桥（LDB）驱动，该驱动控制 LDB 模块，通过 LVDS 接口与外部显示设备连接。LDB 的目的是支持同步 RGB 数据通过 LVDS 接口从 IPU 或 LCDIF 流到外部显示设备。

支持以下内容：

- 与相关设备的连接——带有 LVDS 接收器的显示器。
- 根据外部显示接收器和 LVDS 显示标准的要求排列数据。
- 同步和控制功能。

6.4.4.2 软件操作

如果驱动是内置的，则 LDB 驱动可以正常工作。

当正确探测了 LDB 设备后，该驱动使用平台数据信息配置 LDB 参考电阻模式和 LDB 调压器。LDB 驱动探测功能尝试将外部显示设备的视频模式与 LVDS 接口相匹配。LDB 的显示信号极性控制位根据匹配的视频模式进行设置。根据用户设置的 LDB 设备树节点，设置 LDB 的 LVDS 通道映射模式和位映射模式。如果驱动将带有 LVDS 接口的显示设备识别为主用显示设备，则 LDB 在探测功能中完全启用。

驱动启用 LVDS 通道的步骤如下所示：

1. 设置 ldb_di_clk 的父时钟和父时钟的速率。
2. 设置 ldb_di_clk 的速率。
3. 启用 ldb_di_clk 及其父时钟。

4. 将 LDB 设置为合适的模式，包括显示信号的极性、LVDS 通道映射模式、位映射模式、参考电阻模式。
5. 启用相关的 LVDS 通道。

6.4.4.3 源代码结构

表 66. LDB 源文件

文件	说明
drivers/video/fbdev/mxc/ldb.c	LDB 帧缓冲驱动

6.4.4.4 菜单配置选项

为该模块提供了以下 Linux 内核配置选项。

在菜单配置中启用以下模块：

Device Drivers -> Graphics support -> MXC Framebufer support ->Synchronous Panel Framebuffer -> MXC LDB

6.4.5 EPDC 接口

6.4.5.1 介绍

电泳显示控制器（EPDC）是一种直接驱动有源矩阵 EPD 控制器，旨在用于驱动支持多种 TFT 背板的 E Ink EPD 屏。EPDC 帧缓冲驱动作为标准 Linux 帧缓冲设备。该驱动支持一组自定义 API 扩展，可从用户空间（通过 IOCTL）或其他内核模块（通过直接函数调用）访问，以便为用户提供访问 EPD 特定功能的权限。EPDC 驱动是从任何特定的 E Ink® 屏类型中抽象出来的，支持灵活地使用多种 E Ink 屏和规格。

EPDC 驱动支持以下功能：

- EPDC 驱动作为可加载模块或内置模块。
- RGB565、RGB24、RGB32 和 Y8 帧缓冲格式。
- 更新整个或部分 EPD 屏幕。
- 最多 256 种屏特定波形模式。
- 对给定更新自动选择最佳波形。
- 通过等待特定更新请求完成实现同步。
- 从备用（覆盖）缓冲区的屏幕更新。
- 自动碰撞处理。
- 64 个同步更新区域。
- Y8 帧缓冲格式的 Ppixel 反转。
- 90、180 和 270 度硬件加速帧缓冲旋转。
- 平移（仅限 y 轴方向）。
- 通过 Linux fb_deferred_io 机制自动更新全屏和部分屏幕。
- 3 种 EPDC 驱动显示更新方案：快照、队列和队列与合并。
- 通过一次性指定的 API 调用设置环境温度，或每次更新时设置环境温度。
- 用户控制完成所有更新和关闭 EPDC 之间的延迟。

6.4.5.2 EPDC 帧缓冲驱动概述

帧缓冲设备为图形硬件提供抽象。它代表帧缓冲视频硬件，允许应用软件通过定义明确的接口访问图形硬件，从软件中抽象出如何管理底层硬件寄存器。EPDC 驱动支持此模式，但有一个关键说明：帧缓冲的内容不会自动更新到 E Ink 显示器。相反，需要一个自定义 API 函数调才能触发对 E Ink 显示器的更新。 “EPDC 帧缓冲驱动扩展”中说明了该过程的详细信息。

在内核配置的图形参数下选择 frame buffer（帧缓冲）选项，启用帧缓冲驱动。为了补充帧缓冲驱动，内核构建器可能还支持字体和启动标识。帧缓冲设备依靠虚拟终端（VT）控制台从串行模式切换到图形模式。通过位于/dev 目录中的特殊设备节点（如/dev/fb*）访问该设备。fb0 通常是主用帧缓冲区。

帧缓冲设备是一种存储器设备，如/dev/mem，其功能与存储设备类似。用户可以读取、写入、查找其中的某个位置，以及对其进行 mmap()操作（主要用途）。区别在于，特殊文件中出现的存储器不是整个存储器，而是某些视频硬件的帧缓冲区。

EPDC 帧缓冲驱动（i.MX 6DualLite 上的 drivers/video/fbdev/mxc/mxc_epdc_fb.c 或 i.MX 7Dual 上的第二代 EPDC 的 drivers/video/fbdev/mxc/ mxc_epdc_v2_fb.c）与通用 Linux 帧缓冲驱动（drivers/ video/fbmem.c）进行密切交互。

如需了解帧缓冲设备的更多详细信息，请参见 Documentation/fb/framebuffer.txt 中的 Linux 内核文档。

6.4.5.3 EPDC 帧缓冲驱动扩展

E Ink 显示技术与 EPDC 相结合，具有几个独特的功能，使其与基于 LCD 的标准帧缓冲设备不同。这些差异导致需要对帧缓冲接口进行 API 扩展。EPDC 异步刷新 E Ink 显示器，并支持部分屏幕更新。因此，EPDC 要求用户在帧缓冲区内容已被修改以及需要更新哪个区域时发送通知。EPDC 对 E Ink 显示器的更新的另一个独有特征是较长的屏幕更新延迟（在 300-980 毫秒之间），这就需要一种机制来允许用户等待给定的屏幕更新完成。

帧缓冲设备的自定义 API 扩展既可以从用户空间应用访问，也可以从内核空间访问。标准设备 IOCTL 接口允许用户空间应用访问自定义 API。IOCTL 扩展以及相关的数据结构和定义位于 include/linux/mxcfb_epdc.h。如需获得这些 IOCTL 的完整描述，请参见 “软件操作”章节的“编程接口”部分。

对于自定义 API 扩展的内核模式访问，应旁路 IOCTL 接口，直接访问底层函数。

6.4.5.4 EPDC 屏配置

EPDC 驱动旨在灵活地支持具有各种屏分辨率、定时参数和波形模式的 E Ink 屏。通过使用 EPDC 屏模式结构 imx_epdc_fb_mode（可在 include/linux/mxcfb_epdc.h 中找到），EPDC 驱动与屏无关。

```
struct imx_epdc_fb_mode {
struct fb_videomode *vmode;
int vscan_holdoff;
int sdoed_width;
int sdoed_delay;
int sdoez_width;
int sdoez_delay;
int gdclk_hp_offs;
int gdsp_offs;
int gdoe_offs;
int gdclk_offs;
int num_ce;
};
```

imx_epdc_fb_mode 结构由 fb_videomode 结构参考和一组 EPD 定时参数组成。fb_videomode 结构定义了屏分辨率和基本定时参数（像素时钟频率、hsync 和 vsync 裕量），imx_epdc_fb_mode 中的附加定时参数定义了 EPD 特定的定时参数，例如时钟源和栅极驱动时钟。如需了解如何配置 E Ink 屏定时参数的详细信息，请参见《*i.MX 6DualLite 应用处理器参考手册*》（IMX6DLRM）或《*i.MX 7Dual 应用处理器参考手册*》（IMX7DRM）中的“EPDC 编程模式”章节。

除了 EPDC 屏模式数据外，还可以将函数传递给 EPDC 驱动，以定义在启用或禁用 EPDC 驱动时应如何处理 EPDC 引脚。出于节能目的，这些功能应禁用 EPDC 引脚。

6.4.5.5 引导命令行参数

EPDC 驱动的其他配置通过引导命令行参数提供。命令行选项的格式为

```
epdc video=mxcepdcfb:[panel_name],bpp=16
```

EPDC 驱动解析这些选项，并尝试将 panel_name 与 imx_epdc_fb_mode 屏模式结构中指定的视频模式名称相匹配。如果未找到匹配项，则 EPDC 驱动将使用平台数据中提供的第一个屏模式。此命令行中的 bpp 设置为帧缓冲区设置每像素初始位。设置为 32 或 24 将选择 RGB888 像素格式，设置为 16 则选择 RGB565 像素格式，而设置为 8 将选择 8 位灰度（Y8）格式。

6.4.5.6 EPDC 波形加载

EPDC 驱动需要波形文件才能正常运行。该波形文件包含生成驱动 E Ink 屏更新的波形所需的波形信息。在执行第一次更新之前，指向波形文件数据的指针被编址到 EPDC 中。

有两个选项可用于选择波形文件：

1. 选择此 BSP 版本中包含的默认波形文件之一。
2. 使用正在使用的 E Ink 屏特定的新波形文件。

EPDC 驱动使用 Linux 固件 API 加载波形文件。

6.4.5.7 使用默认波形文件

要开始使用 E Ink 屏和 EPDC 驱动，最快、最简便的方法是使用 Linux BSP 中提供的默认波形文件之一。这样可以在没有屏特定波形文件的情况下更新几种不同类型的 E Ink 屏。缺点是达不到最佳质量。通常，对 E Ink 屏使用非屏特定的波形文件会导致更多的伪像，且色彩质量整体较差。

BSP 中包含的以下默认波形文件位于/lib/firmware/imx/epdc 中：

- epdc_E60_V110.fw – 6.0 英寸 V110 E Ink 屏的默认波形。
- epdc_E60_V220.fw – 6.0 英寸 V220 E Ink 屏的默认波形（支持动画模式更新）。
- epdc_E97_V110.fw – 9.7 英寸 V110 E Ink 屏的默认波形。
- epdc_E060SCM.fw – 6.0 英寸 Pearl E Ink 屏的默认波形（支持动画模式更新）。
- epdc_ED060XH2C1.fw – 6.0 英寸 E Ink 屏的默认波形（默认不支持 Reagl/-D。如需了解 Reagl/-D 支持的信息，请联系 NXP support。）

EPDC 驱动尝试在 rootfs 中的/lib/firmware/imx/epdc 目录下加载名为“epdc_[panel_name].fw”的波形文件，其中 panel_name 指的是 fb_videomode `name` 字段中指定的字符串。这个 panel_name 信息应该通过前一章中介绍的内核命令行参数提供给 EPDC 驱动。例如，要为 Pearl 屏加载 epdc_E060SCM.fw 默认固件文件，EPDC 内核命令行参数设置应如下所示：

```
video=mxcepdcfb:E060SCM,bpp=16
```

6.4.5.8 使用自定义波形文件

为确保最佳 E Ink 显示质量，请使用所用 E Ink 屏特定的波形文件。需要将原始波形文件类型（.wbf）转换为 EPDC 可以理解和读取的格式。此转换脚本不包含在 BSP 中。因此，请联系恩智浦获取此转换脚本。

在原始波形文件上运行波形转换脚本后，应对转换后的波形文件重命名，以便 EPDC 驱动可以找到并加载该文件。驱动将在 rootfs 中的/lib/firmware/imx/epdc 目录下搜索名为“epdc_[panel_name].fw”的波形文件，其中 panel_name 指的是 fb_videomode `name` 字段中指定的字符串。例如，如果屏名为“E60_ABCD”，则转换后的波形文件应命名为 epdc_E60_ABCD.fw。

注意

如果 EPDC 驱动搜索与其中一个默认波形文件的名称相匹配的固件波形文件（请参见前一章），则它将选择内置在 BSP 中的默认固件文件，而不是已添加在固件搜索路径中的任何固件文件。因此，如果让 BSP 使用默认固件文件，请确保使用的屏名称与和默认固件文件关联的屏名称不同，因为将优先选择这些默认波形文件，而不选择放置在固件搜索路径中的新波形文件。

6.4.5.9 EPDC 屏初始化

加载帧缓冲驱动模块时，帧缓冲驱动通常不会执行任何硬件初始化步骤（如需了解例外情况，请参见下面的注释）。相反，必须进行后续用户模式调用，以请求驱动为特定的 EPD 屏进行自我初始化。要初始化 EPDC 硬件和 E Ink 屏，必须调用 FBIOPUT_VSCREENINFO ioctl，并设置 fb_var_screeninfo 参数的 xres 和 yres，以匹配支持的 E Ink 屏类型的 X 和 Y 分辨率。为确保 EPDC 驱动收到初始化请求，应将 fb_var_screeninfo 参数的 activate 字段设置为 FB_ACTIVATE_FORCE。

注意

当内核中包含 FB Console 驱动时例外。当 EPDC 驱动注册帧缓冲设备时，FB Console 驱动随后将调用 FBIOPUT_VSCREENINFO ioctl。这反过来会初始化 EPDC 屏。

6.4.5.10 灰度帧缓冲选择

EPDC 帧缓冲驱动支持对帧缓冲使用 8 位灰度（Y8）和 8 位反转灰度（Y8 倒）像素格式（除了更常见的 RGB565 像素格式）。为了将帧缓冲格式配置为 8 位灰度，应用将调用 FBIOPUT_VSCREENINFO framebuffer ioctl。这个 ioctl 将 fb_var_screeninfo 指针作为参数。此参数指定帧缓冲的属性，并允许应用请求更改帧缓冲格式。为了请求更改为 8 位灰度格式，必须设置 fb_var_screeninfo 参数的两个关键成员：bits_per_pixel 和 grayscale。必须将 bits_per_pixel 设置为 8，且将 grayscale 设置为 2 个有效的灰度格式值之一：GRAYSCALE_8BIT 或 GRAYSCALE_8BIT_INVERTED。

以下代码片段演示了将帧缓冲更改为使用 Y8 像素格式的请求：

```
fb_screen_info screen_info;
screen_info.bits_per_pixel = 8;
screen_info.grayscale = GRAYSCALE_8BIT;
retval = ioctl(fd_fb0, FBIOPUT_VSCREENINFO, &screen_info);
```

6.4.5.11 软件操作

可以从用户空间和内核空间访问 EPDC 帧缓冲区。一组函数描述了 EPDC 帧缓冲驱动扩展。使用 IOCTL 接口的用户空间和直接使用函数的内核空间可通过两种模式访问这些功能。下面将介绍每个 IOCTL 和函数组合。

MXCFB_SET_WAVEFORM_MODES / mxc_epdc_fb_set_waveform_modes()

说明：

定义常见波形模式的映射。

参数：

mxcfb_waveform_modes **modes*

指向包含常用波形模式的波形模式值的结构的指针。必须配置这些值，才能使自动波形模式选择功能正常运行。

MXCFB_SET_TEMPERATURE / mxc_epdc_fb_set_temperature

说明：

设置 EPDC 驱动在后续屏更新中使用的温度。

参数：

int*32_t temperature*

温度值，以摄氏度为单位。请注意，在使用 MXCFB_SEND_UPDATE ioctl 时，可通过将温度值参数设置为 TEMP_USE_AMBIENT 之外的任何值来覆盖此温度设置。

MXCFB_SET_AUTO_UPDATE_MODE / mxc_epdc_fb_set_auto_update

说明：

在自动和区域更新模式之间进行选择。

参数：

__u32 mode

在区域更新模式下，必须通过 MXCFB_SEND_UPDATE IOCTL 提交更新。

在自动模式下，驱动通过检测帧缓冲存储器区域中已修改的页面自动生成更新。

MXCFB_SET_UPDATE_SCHEME / mxc_epdc_fb_set_upd_scheme

说明：

选择一个方案指示驱动内的更新流程。

参数：

__u32 scheme

选择以下更新方案：

UPDATE_SCHEME_SNAPSHOT——在 Snapshot 更新方案中，帧缓冲区的内容被立即处理并存储在驱动内部的存储器缓冲区中。当对 MXCFB_SEND_UPDATE 的调用完成时，帧缓冲区域是空闲的，可以在不影响上次更新完整性的情况下进行修改。如果由于其他未决更新而导致更新帧提交延迟，则在最终将更新提交到 EPDC 硬件时显示原始缓冲区内容。如果更新导致冲突，则在冲突清除后重新提交原始更新内容。

UPDATE_SCHEME_QUEUE——Queue 更新方案使用工作队列异步处理所有更新的处理和提交。当通过 MXCFB_SEND_UPDATE 提交更新时，更新会添加到队列中，然后在 EPDC 硬件资源可用时按顺序进行处理。因此，处理和更新的帧缓冲区内容不能保证反映在将更新发送到驱动时帧缓冲区中存在的内容。

UPDATE_SCHEME_QUEUE_AND_MERGE——Queue and Merge 方案使用 Queue 方案的排队概念，但增加了一个合并步骤。这意味着，在工作队列中处理更新之前，首先将其与其他挂起的更新进行比较。如果任何更新与当前更新的模式和标志匹配，并且还与当前更新的更新区域重叠，则该更新将与当前的更新合并。尝试合并所有挂起的更新后，将处理并提交最终合并的更新。

MXCFB_SEND_UPDATE / mxc_epdc_fb_send_update

说明：

请求将帧缓冲区的一个区域更新到显示器。

参数：

mxcfb_update_data **upd_data*

指向定义当前更新的帧缓冲区区域、波形模式和冲突模式的结构的指针。此结构还包括一个标志字段，可从以下更新选项中进行选择：

EPDC_FLAG_ENABLE_INVERSION——启用更新区域中所有像素的反转。

EPDC_FLAG_FORCE_MONOCHROME——启用更新区域中所有像素的全黑/白分色。

EPDC_FLAG_USE_ALT_BUFFER——启用从备用（非帧缓冲区）存储器缓冲区进行更新。

如果启用，最后的 *upd_data* 参数包括备用存储器缓冲区的详细配置信息。

MXCFB_WAIT_FOR_UPDATE_COMPLETE / mxc_epdc_fb_wait_update_complete

说明：

阻止并等待上一个更新请求完成。

参数：

mxfb_update_marker_data *marker_data*

应在此处重新使用标识特定更新（在 MXCFB_SEND_UPDATE IOCTL 调用中作为参数传递）的 update_marker 值，以等待更新完成。如果更新是碰撞测试更新，则 collision_test 变量将返回指示是否发生碰撞的结果。

MXCFB_SET_PWRDOWN_DELAY / mxc_epdc_fb_set_pwrdown_delay

说明：

设置驱动中所有更新完成与驱动关闭 EPDC 和 E Ink 显示器电源之间的延迟。

参数：

int32_t *delay*

以毫秒为单位的输入延迟值。要完全禁用 EPDC 断电，请使用 FB_POWERDOWN_DISABLE（定义如下）。

MXCFB_GET_PWRDOWN_DELAY / mxc_epdc_fb_get_pwrdown_delay

说明：

检索驱动当前的断电延迟值。

参数：

int32_*t delay*

以毫秒为单位的输出延迟值。

6.4.5.12 结构和定义

```
#define GRAYSCALE_8BIT                                                  0x1
#define GRAYSCALE_8BIT_INVERTED                                         0x2
```

```
#define AUTO_UPDATE_MODE_REGION_MODE                                    0
#define AUTO_UPDATE_MODE_AUTOMATIC_MODE                                 1
#define UPDATE_SCHEME_SNAPSHOT                                          0
#define UPDATE_SCHEME_QUEUE                                             1
#define UPDATE_SCHEME_QUEUE_AND_MERGE                                   2
#define UPDATE_MODE_PARTIAL                                             0x0
#define UPDATE_MODE_FULL                                                0x1
#define WAVEFORM_MODE_AUTO                                              257
#define TEMP_USE_AMBIENT                                                0x1000
#define EPDC_FLAG_ENABLE_INVERSION                                      0x01
#define EPDC_FLAG_FORCE_MONOCHROME                                      0x02
#define EPDC_FLAG_USE_ALT_BUFFER                                        0x100
#define EPDC_FLAG_TEST_COLLISION                                        0x200
#define FB_POWERDOWN_DISABLE                                            -1
struct mxcfb_rect {
_ u32 left; /* Starting X coordinate for update region */
_ u32 top; /* Starting Y coordinate for update region */
_ u32 width; /* Width of update region */
_ u32 height; /* Height of update region */
};
struct mxcfb_waveform_modes {
int mode_init; /* INIT waveform mode */
int mode_du; /* DU waveform mode */
int mode_gc4; /* GC4 waveform mode */
int mode_gc8; /* GC8 waveform mode */
int mode_gc16; /* GC16 waveform mode */
int mode_gc32; /* GC32 waveform mode */
};
struct mxcfb_alt_buffer_data {
_ u32 phys_addr; /* physical address of alternate image buffer */
_ u32 width; /* width of entire buffer */
_ u32 height; /* height of entire buffer */
struct mxcfb_rect alt_update_region; /* region within buffer to update */
};
struct mxcfb_update_data {
struct mxcfb_rect update_region; /* Rectangular update region bounds */
_ u32 waveform_mode; /* Waveform mode for update */
_ u32 update_mode; /* Update mode selection (partial/full) */
_ u32 update_marker; /* Marker used when waiting for completion */
int temp; /* Temperature in Celsius */
uint flags; /* Select options for the current update */
struct mxcfb_alt_buffer_data alt_buffer_data; /* Alternate buffer data */
};
struct mxcfb_update_marker_data {   u32 update_marker;   u32 collision_test; };
```

6.4.5.13 源代码结构

下表列出了与 EPDC 驱动关联的源文件和用于编址访问的头文件。

表 67. EPDC 源文件

文件	说明
drivers/video/fbev/mxc/mxc_epdc_v2_fb.c	用于 i.MX 7Dual 的 EPDC 第二代 V2 帧缓冲驱动
drivers/video/fbdev/mxc/epdc_v2_regs.h	EPDC 第二代寄存器定义
drivers/video/fbdev/mxc/mxc_epdc_fb.c	用于 i.MX 6Sololite、6SLL 和 6DualLite 的第一代 EPDC 帧缓冲驱动

下页继续.....

表 67. EPDC 源文件（续)

文件	说明
drivers/video/fbdev/mxc/epdc_regs.h	EPDC 第一代寄存器定义
drivers/video/fbdev/mxc/epdc_v2_regs.h	第二代 EPDC v2 寄存器定义
include/linux/uapi/mxcfb.h	EPDC IOCTL 和帧缓冲驱动的头文件
include/linux/mxcfb_epdc.h	用于直接内核访问 EPDC API 扩展的头文件

6.4.5.14 菜单配置选项

为 EPDC 模块提供了以下 Linux 内核配置选项：

- CONFIG_FB_MXC_EINK_PANEL——支持电泳显示控制器。在 menuconfig 中，选择“Device Drivers > Graphics Support > E-Ink Panel Framebuffer”
- CONFIG_FB_MXC_EINK_V2_PANEL——支持 v2 电泳显示控制器。在 menuconfig 中，此选项位于“Device Drivers > Graphics support > E-Ink Panel Framebuffer based on EPDC V2”
- CONFIG_FB——包括帧缓冲区支持，默认启用。在 menuconfig 中选择“Device Drivers > Graphics support > Support for frame buffer devices”
- CONFIG_MXC_PXP_V2——支持 PxP 并且 EPDC 驱动需要它才能处理（颜色空间转换、旋转、自动波形选择）帧缓冲区更新区域。在 menuconfig 中选择“Device Drivers > DMA Engine support > MXC PxP support”
- CONFIG_MXC_PXP_V3——支持下一级 PxP，第二代 EPDC 驱动需要它才能处理帧缓冲区更新区域。在 menuconfig 中选择“Device Drivers > DMA Engine support > MXC PxP V3 support”

6.4.6 高清多媒体接口（HDMI）和显示端口（DP）概述

6.4.6.1 介绍

高清多媒体接口（HDMI）和显示端口（DP）提供高清视频。一些 i.MX 芯片支持 HDMI 模块，无论是片上解决方案还是外部解决方案。显示端口（DP）提供嵌入式显示端口（eDP）发射器，包括 HDMI 传输（TX）控制器和 PHY。

以下是合规版本。

- HDMI 1.4 和 2.0
- DVI 1.0
- DP 1.3
- eDP 1.4
- HDCP 1.4/2.2

每个 SoC HDMI 解决方案在单独的章节中介绍。i.MX 上的显示端口使用相同的 IP 块，但规格不同。下表列出了支持 HDMI 和显示端口的 SOC 及其支持的版本。

表 68. HDMI 支持

SoC	特性
i.MX 6QuadPlus/Quad/Dual	片上 HDMI 1.4
i.MX 7ULP	外部芯片 HDMI 1.4

下页继续.....

表 68. HDMI 支持（续）

SoC	特性
i.MX 8M Quad	片上 HDMI 2.0/显示端口 1.3
i.MX 8QuadMax	片上 HDMI 2.0/显示端口 1.3

HDMI 音频数据源来自 S/PDIF TX。

6.4.6.2 软件操作

HDMI 驱动根据其两个主要用途（向 HDMI 接收设备提供视频和音频）分为多个子组件。

视频显示驱动组件和音频驱动组件需要一个额外的核心驱动组件来管理常见的 HDMI 资源，包括 HDMI 寄存器、时钟和 IRQ。

6.4.6.3 核心

片上 HDMI i.MX 解决方案支持核心驱动，用于管理必须在 HDMI 音频和视频驱动之间共享的资源。HDMI 音频和视频驱动取决于 HDMI 核心驱动，HDMI 核心驱动应始终在音频和视频之前加载和初始化。核心驱动具有以下功能：

- 映射 HDMI 寄存器区域并提供用于读取和写入 HDMI 寄存器的 API。
- 对关键 HDMI 寄存器执行一次性初始化。
- 初始化 HDMI IRQ，并提供用于启用和禁用 IRQ 的共享 API。
- 提供了一种在音频和视频驱动之间共享信息的方法（例如，HDMI 像素时钟）。
- 提供了在发生隐藏/恢复、插入/拔出事件时实现 HDMI 视频和 HDMI 音频之间同步的方法。当 HDMI 线处于拔出状态或 HDMI 处于隐藏状态时，HDMI 音频无法开始工作。每次 HDMI 音频开始播放时，HDMI 音频驱动都应将其 PCM 注册到核心驱动中，并在播放完成后注销 PCM。一旦发生 HDMI 视频隐藏或电缆拔出事件，如果其 PCM 已注册，核心驱动将暂停 HDMI 音频 DMA 控制器。当 HDMI 恢复或发生电缆插入事件时，核心驱动首先会检查电缆是否处于插入状态，视频状态是否为恢复，以及 PCM 是否已注册。如果上面列出的项目均为“是”，则核心驱动将重新启动 HDMI 音频 DMA。

6.4.6.4 显示设备注册和初始化

在操作系统启动流程中，发生以下软件活动，通过 MXC 显示驱动系统将 HDMI 显示设备连接到 i.MX 帧缓冲驱动：

1. 在 HDMI 视频驱动初始化过程中，调用 `mxc_dispdrv_register()`将 HDMI 模块注册为显示设备，并将 `mxc_hdmi_disp_init()`函数设置为显示设备 init 回调。
2. i.MX 帧缓冲驱动初始化时，调用 `mxc_dispdrv_init()`。这将导致对所有已注册的显示设备执行 init 调用。
3. 执行 `mxc_hdmi_disp_init()`回调。HDMI 驱动从 i.MX 帧缓冲驱动接收包含帧缓冲区信息（fbi）的结构。HDMI 驱动对自己进行注册，以接收 FB 驱动事件的通知。最后，HDMI 驱动通过配置 HDMI 接收热插拔中断来完成初始化。

注意

所有显示设备驱动必须在 i.MX 帧缓冲驱动之前进行初始化，以便所有显示设备都注册为 MXC Display Driver 设备。

6.4.6.5 热插拔处理和视频模式更改

一旦通过 MXC 显示驱动接口在 i.MX 帧缓冲驱动和 HDMI 之间建立了连接，HDMI 视频驱动将等待热插拔中断，该中断指示有效的 HDMI 接收设备已连接并准备好接收 HDMI 视频数据。HDMI 和 i.MX 帧缓冲驱动之间的后续通信通过 Linux 帧缓冲 API 进行。下面列出了识别 HDMI 接收设备并配置 ELCDIF FB 驱动以驱动视频输出的软件流程：

1. HDMI 视频驱动接收热插拔中断并从 HDMI 接收设备读取 EDID，根据检索到的 EDID 信息构建视频模式列表。使用 Linux 内核命令行中的视频模式字符串（用于初始连接）或最近的视频模式（用于稍后的 HDMI 电缆连接），HDMI 驱动从模式列表中选择最匹配的视频模式。
2. HDMI 视频驱动调用 `fb_set_var()` 来更改 i.MX 帧缓冲驱动中的视频模式。i.MX 帧缓冲驱动完成对新模式的重新配置。
3. 调用 fb_set_var()后，将帧缓冲区通知发送回 HDMI 驱动，指示已发生 FB_EVENT_MODE_CHANGE。HDMI 驱动为新视频模式配置 HDMI 硬件。
4. 最后，启用 HDMI 模块以生成到 HDMI 接收设备的输出。

i.MX 帧缓冲驱动将与每个 SoC 特定的显示接口对齐，如每个 SoC HDMI 章节所述。

6.4.6.6 音频

由于 HDMI Tx 音频驱动使用 ALSA SoC 框架，它被分解为几个文件，如每个 HDMI 章节的“源代码结构”部分所示。大部分代码都位于平台 DMA 驱动（sound/soc/imx/imx-hdmi-dma.c）和 CODEC 驱动（sound/soc/codecs/mxc_hdmi.c）中。设备驱动（sound/soc/imx/imx-hdmi.c）分配 SoC 音频设备，并将所有 SoC 组件链接在一起。DAI 驱动（sound/soc/imx/imx-hdmi-dai.c）符合 SoC 要求。它主要用于获取平台数据。

HDMI CODEC 驱动完成 HDMI 音频采样器的大部分初始化工作。请注意，HDMI Tx 模块仅实现 AHB DMA 音频，而不实现其他音频接口（SSI、S/PDIF 等）。HDMI CODEC 驱动的另一个主要功能是设置需要进入音频流的 IEC 头信息的结构。由于该结构连接到 ALSA 层，可以使用“iecset”实用程序在用户空间访问 IEC 设置。

平台 DMA 驱动处理 HDMI Tx 块 DMA 引擎。请注意，HDMI 音频使用 HDMI 块 DMA 以及 SDMA。SDMA 用于实施多缓冲机制。由于 HDMI Tx 块不会自动将 IEC 音频头信息合并到音频流中，平台 DMA 驱动在发送缓冲区之前，使用 hdmi_dma_copy()（不使用存储器映射）或 hdmi_dma_mmap_copy()（使用存储器映射模式）函数进行合并。请注意，由于 IEC 音频头添加操作，用户空间应用可能无法获得足够的 CPU 周期，无法及时将数据馈送到 HDMI 音频驱动，特别是在系统负载较高的情况下。在这种情况下，会听到一些火花噪音。在不同的音频框架（ALSA LIB 或 PULSE AUDIO）中，可能会打印关于该噪声的不同日志。例如，在 ALSA LIB 中，会打印“underrung!!! at least * ms is lost”之类的日志。

HDMI 音频播放取决于 HDMI 像素时钟。因此，在 HDMI 隐藏和电缆拔出的状态下，HDMI 音频要么停止，要么无法播放。详情请参见 software_operation_core。

请注意，因为 HDMI 音频驱动需要添加 IEC 头，驱动需要知道已经写入 HDMI 音频驱动的数据量。如果应用无法解码写入的数据量，例如 ALSA LIB 中的 DMIX 插件，HDMI 音频驱动将无法正常工作。将听不到任何声音。

HDMI 音频支持以下功能：

- 播放采样率
  - 32k、44.1k、48k、88.2k、96k、176.4k、192k
  - HDMI 接收功能
- 播放通道：
  - 2、4、6、8
  - HDMI 接收功能

- 播放音频格式：
  - SNDRV_PCM_FMTBIT_S16_LE

6.4.6.7 i.MX 8 显示端口

6.4.6.7.1 介绍

高清多媒体接口（HDMI）驱动支持 i.MX 8QuadMax 和 iMX 8MQuad 的片上 Cadence HDTX IP 模块，能够使用一根电缆传输未压缩视频、音频和数据。HDMI 驱动分为 3 个子组件：与 DPU/DCSS DRM 驱动集成的视频显示设备驱动、与 ALSA/SoC 子系统集成的音频驱动，以及管理 HDMI 驱动的共享软件和硬件资源的核心 API 驱动。

HDTX IP 支持以下功能：

- 符合 HDMI 2.0 规范。
- 支持 600 Mhz 的像素时钟。
- 支持所有视频格式，包括双视频、立体声和所有色度选项（RGB、YCbCr444/422 和 YCbCr420）。
- 支持以下音频格式：PCM、HBR、DST、一位音频、多流和 3D 音频。
- 支持所有信息帧。
- APB 接口用于控制和读取状态信息。
- 嵌入式 CPU 执行简化 SoC 集成的所有协议特定任务：
  - HDCP 1.4/2.2
  - 音频回传通道（ARC）

HD Display TX 控制器支持一种或多种协议，如 HDMI、DisplayPort 或 eDP。每个协议都需要不同的固件二进制文件。下图对此进行了描述。

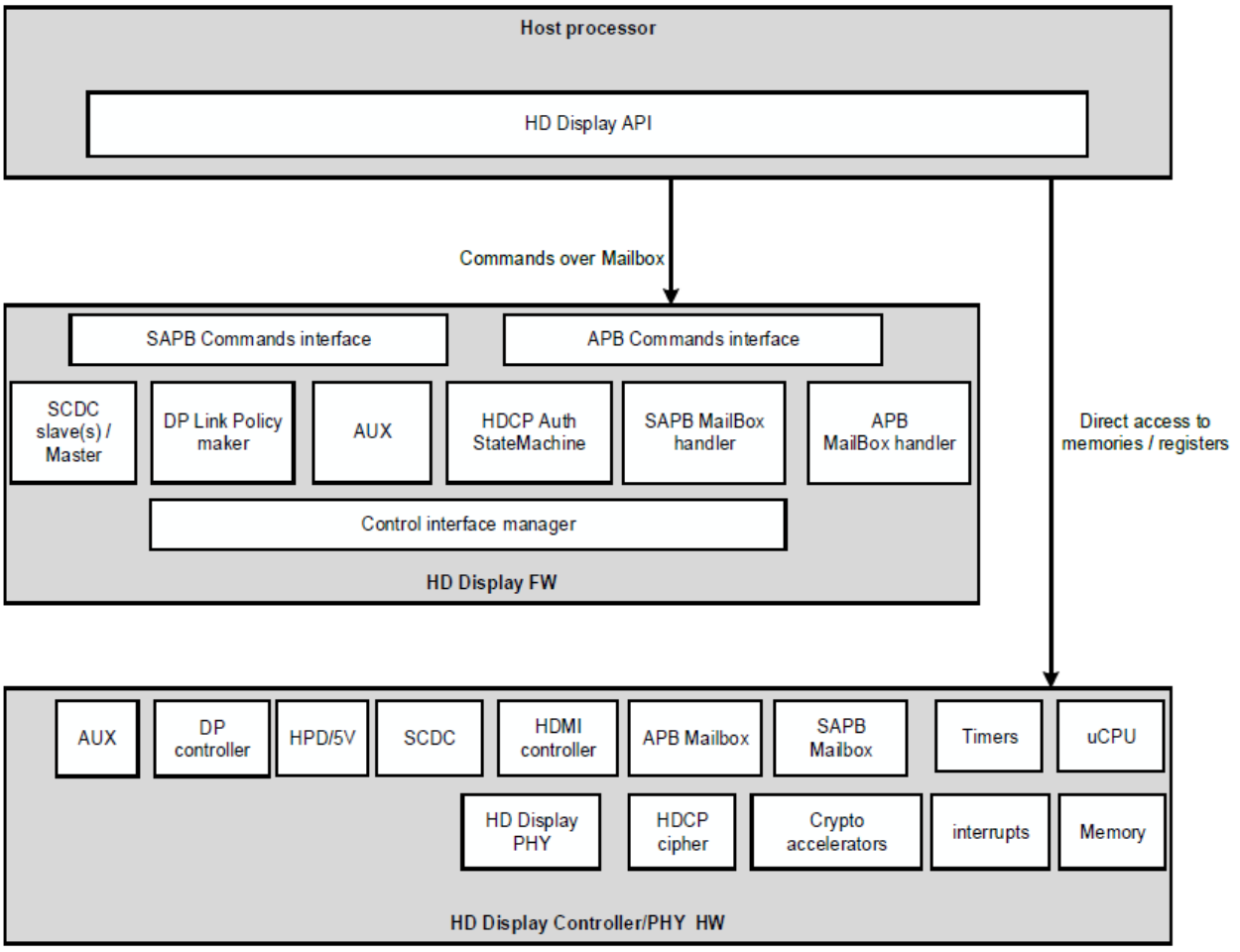


图 21. HDMI 硬件集成

HD Display 控制器集成了一个运行嵌入式固件（FW）的 CPP（uCPU）。固件管理 HD Display 链接，并提供边带通道通信。固件不涉及数据通路（视频、音频或信息帧）。

主处理器通过 APB 接口与 HD Display 控制器对接。主处理器通过以下一种或多种方式管理 HD Display 控制器：

- 直接访问硬件寄存器以进行调试。
- 直接访问 I-MEM 和 D-MEM（在启动期间）以进行固件下载。
- 在操作模式下直接访问指定硬件模块的硬件寄存器（不受固件控制的模块）。
- 在操作模式下，通过命令接口与固件进行通信（使用 GENERAL_WRITE_REGISTER 和 GENERAL_READ_REGISTER 命令），间接访问指定硬件模块的硬件寄存器。
- 使用命令接口通过邮箱与不同的固件模块进行通信。

6.4.6.7.2 软件操作

HDMI 驱动根据其两个主要用途分为 2 个子组件：HDP DRM 驱动和核心 API 驱动。

HDP DRM 驱动需要核心 API 驱动组件来配置 HDMI 固件。

6.4.6.7.3 源代码结构

HDMI 驱动有 3 个软件组件：MHDP DRM 桥和核心 API 驱动、MHDP i.MX 8 平台驱动和 HDMI 音频驱动。

表 69. HDP 核心 API 驱动文件列表

文件	说明
drivers/gpu/drm/bridge/cadence	MHDP DRM 桥和核心 API 驱动
drivers/gpu/drm/imx/mhdp	MHDP i.MX 8 平台驱动
• sound/soc/fsl/fsl_hdmi.c • sound/soc/fsl/imx-hdmi.c • sound/soc/fsl/hdmi_pcm.S • sound/soc/fsl/imx-hdmi-dma.c • sound/soc/fsl/imx-cdnhdmi.c	HDMI 音频驱动

6.4.6.7.4 菜单配置选项

有 3 个主要的 Linux 内核配置选项，用于在 Linux 操作系统映像中选择和包含 HDMI 驱动功能。CONFIG_DRM_CDNS_MHDP 选项支持 MHDP DRM 桥和核心 API 驱动，并且可以在 menuconfig 中的以下菜单位置进行选择：

Device Drivers > Graphics support >Display Interface Bridges > Cadence MHDP COMMON API driver

CONFIG_DRM_IMX_CDNS_MHDP 选项支持 i.MX 8 HDMI/DP 视频驱动，可以在 menuconfig 中的以下菜单位置进行选择：

Device Drivers > Graphics support > NXP i.MX MX8 DRM HDMI/DP

CONFIG_SND_SOC_IMX_CDNHDMI 选项通过 ALSA/SoC 子系统支持 HDMI 音频，可在 menuconfig 中的以下位置找到：

Device Drivers > Sound card support > Advanced Linux Sound Architecture > ALSA for SoC audio support > SoC Audio support for CDN - HDMI

6.4.6.8 i.MX 6 片上高清多媒体接口（HDMI）

6.4.6.8.1 介绍

高清多媒体接口（HDMI）驱动支持 i.MX 6QuadPlus、6Quad 和 6Dual SoC 上的片上 DesignWare HDMI 硬件模块，该驱动可使用一根电缆传输未压缩的视频、音频和数据。

HDMI 驱动分为 4 个子组件：

- 与 Linux Frame Buffer API 集成的视频显示设备驱动
- 与 ALSA/SoC 子系统集成的音频驱动
- CEC 驱动
- 多用设备（MFD）驱动，用于管理 HDMI 驱动的共享软件和硬件资源。

HDMI 驱动支持以下功能：

- 与 MXC Display Device 框架集成（用于管理显示设备与 IPU 的连接）
- 分辨率为 1080p60 的 HDMI 视频输出
- 支持从 HDMI 接收设备读取 EDID 信息

- 热插拔检测
- 支持 CEC
- 自动时钟管理，以最大限度地降低功耗
- 支持系统挂起/恢复
- HDMI 音频播放（2、4、6 或 8 通道，16 位，采样率为 32-KHz 至 192-KHz）
- 使用“iecset”实用程序通过 ALSA 公开的 IEC 音频头信息

HDMI 模块从图像处理单元（IPU）接收视频数据，从外部存储器接口接收音频数据，从 CPU 接收控制数据，如下图所示。输出数据通过 3 个最小转换差分信号（TMDS）通道传输到 SoC 外部的 HDMI 接收设备。HDMI 还承载了 VESA 数据显示通道（DDC）。DDC 是一个 I2C 接口，允许 HDMI 发送器向 HDMI 接收器查询扩展显示标识数据（EDID）。CEC 通道提供可选的对发送设备和接收设备的高级控制功能。

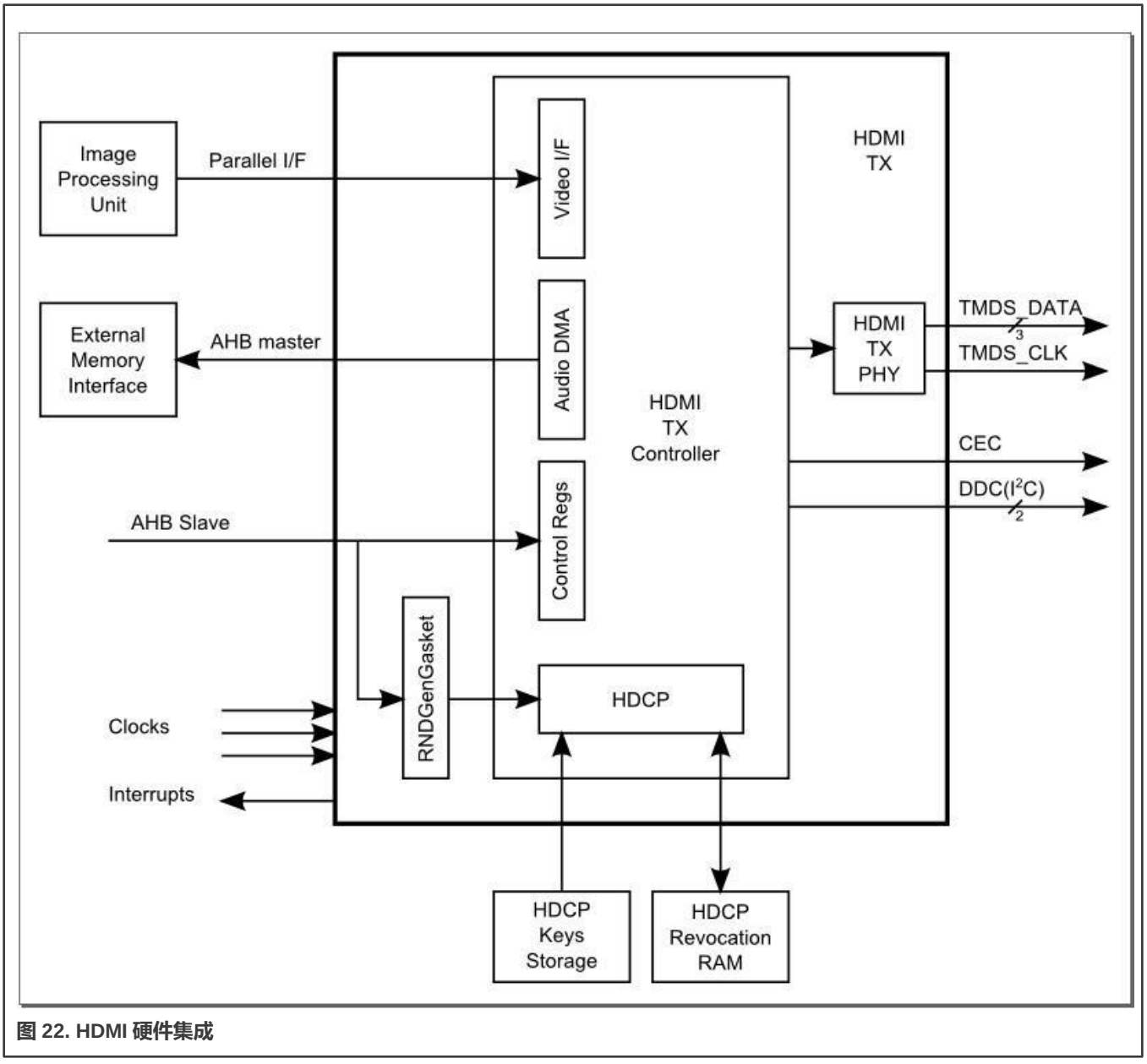


图 22. HDMI 硬件集成

HDMI 的视频输入可配置，可以来自 i.MX 6 系列芯片的两个 IPU 模块中的任意一个，也可以来自 IPU 的两个 Display Interface（DI）端口 DI0 或 DI1 中的任意一个。此配置通过 IOMUX 模块使用 HDMI_MUX_CTRL register 字段进行控制。如需了解此互连的说明，请参见下图。

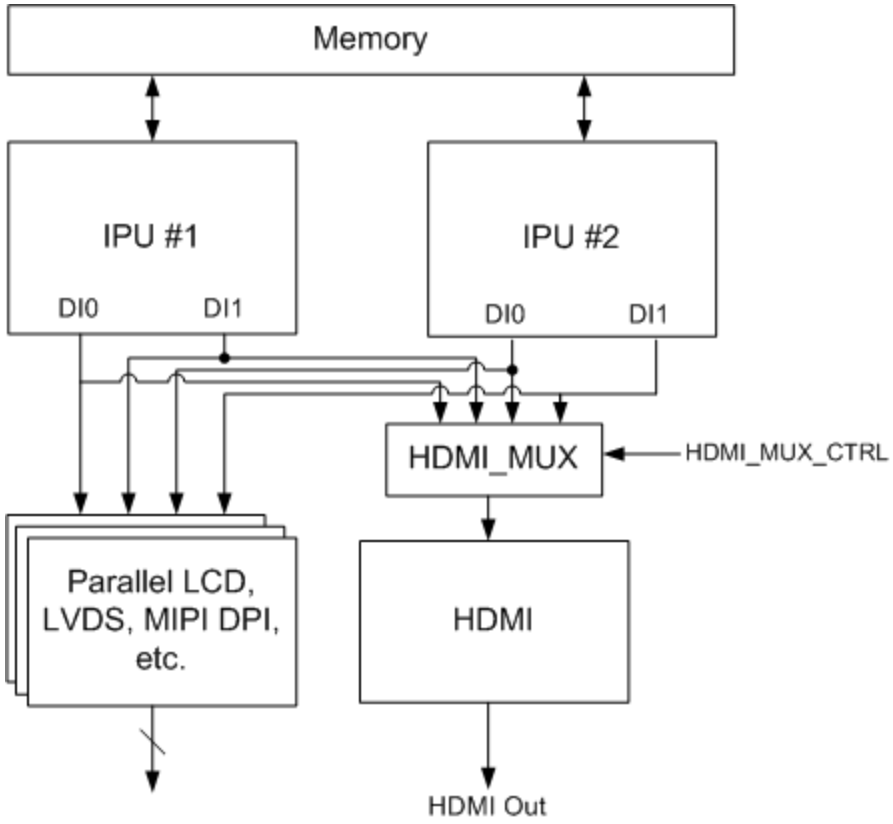


图 23. IPU-HDMI 硬件互连

6.4.6.8.2 软件操作

HDMI 驱动根据其两个主要用途分为两个子组件：向 HDMI 接收设备提供视频和音频。

视频显示驱动组件和音频驱动组件需要一个额外的核心驱动组件来管理常见的 HDMI 资源，包括 HDMI 寄存器、时钟和 IRQ。下图说明了各种 HDMI 子驱动之间的互连情况，以及 HDMI 视频驱动和 Linux 帧缓冲子系统之间的互连。

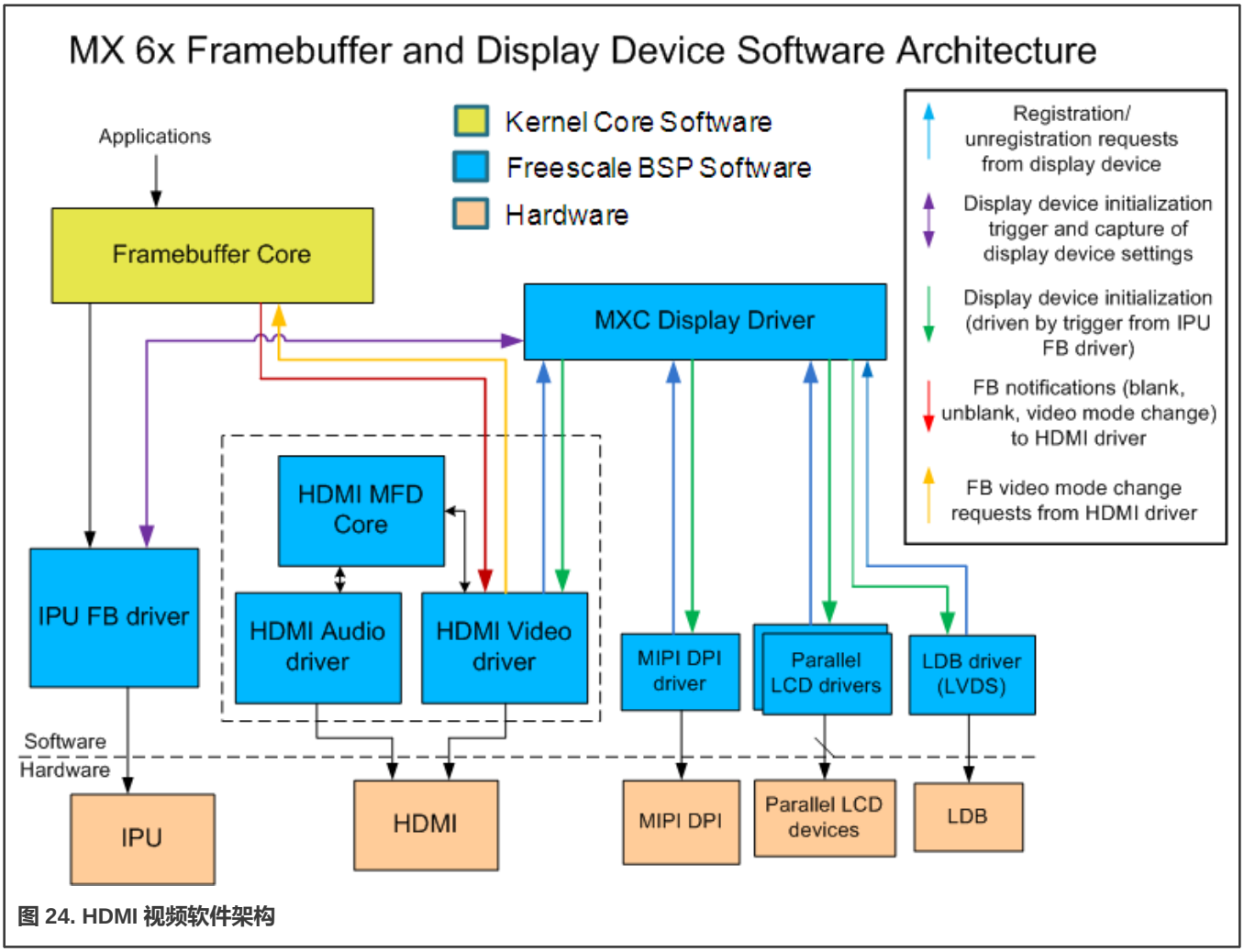


图 24. HDMI 视频软件架构

i.MX 6Dual/6Quad/6QuadPlus/6Solo/6DualLite 支持连接到 IPU 模块并由其驱动的多种显示输出设备（例如 LVDS、LCD、HDMI 和 MIPI 显示器）。MXC 显示驱动 API 提供了一个系统，用于注册显示设备并配置如何将其连接到每个 IPU DI。HDMI 驱动使用此 API 将自己注册为显示设备，以便从 IPU 接收正确的视频输入。

6.4.6.8.3 CEC

HDMI CEC 是一种协议，提供对用户环境中的各种视听产品的高级控制。HDMI CEC 驱动实施 HDMI CEC 底层协议的软件部分。其中包括获取逻辑地址、CEC 消息发送和接收、错误处理、消息重传等。

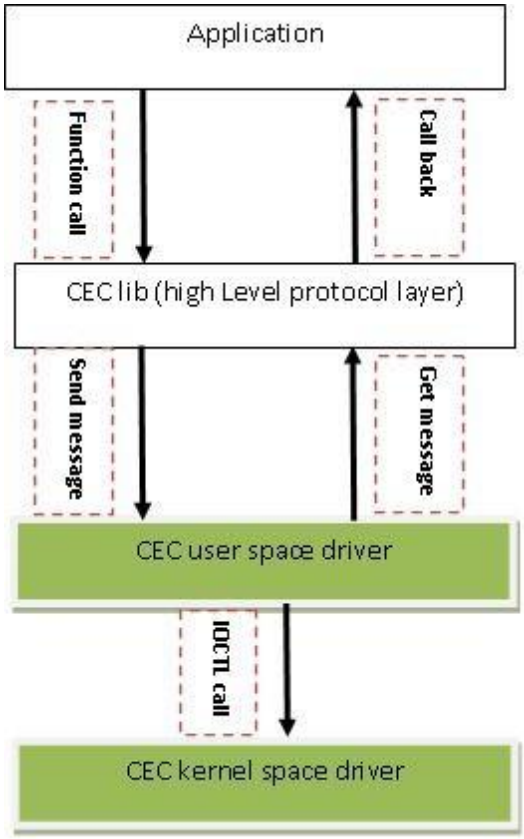


图 25 HDMI CEC 软件架构

6.4.6.8.4 源代码结构

HDMI 源代码在 HDMI 核心驱动、HDMI 显示驱动和 HDMI 音频驱动中提供。

表 70. HDMI 源文件

文件	说明
drivers/mfd/mxc-hdmi-core.c	HDMI 核心驱动实施
include/linux/mfd/mxc-hdmi-core.h	HDMI 核心驱动头文件
drivers/video/fbdev/mxc/mxc_hdmi.c	HDMI 显示驱动实施
sound/soc/fsl/fsl_hdmi.c	HDMI Audio SoC DAI 驱动实施
sound/soc/fsl/imx-hdmi-dma.c	HDMI Audio SoC 平台 DMA 驱动实施
drivers/mxc/hdmi-cec/hdmi-cec.c	HDMI CEC 驱动实施。 HDMI CEC 库文件在 codeaurraforum 上的 imx-lib 存储库中提供。
drivers/mxc/hdmi-cec/hdmi-cec.c	HDMI CEC 驱动实施。 HDMI CEC 库文件在 code aurora forum 上的 imx-lib 中提供。

6.4.6.8.5 菜单配置选项

有 3 个主要的 Linux 内核配置选项用于在 Linux 操作系统映像中选择和包含 HDMI 驱动功能。

HDMI 视频支持取决于对同步屏帧缓冲区的支持，以及对 IPUv3 的支持。CONFIG_FB_MXC_HDMI 支持 HDMI 视频驱动，可以通过“Device Drivers > Graphics support> Support for frame buffer devices > MXC HDMI driver support”来选择

CONFIG_SND_SOC_IMX_HDMI 通过 ALSA/SoC 子系统支持 HDMI 音频，可以通过“Device Drivers > Sound card support > Advanced Linux Sound Architecture > ALSA for SoC audio support > SoC Audio support for IMX – HDMI”进行选择

选择前两个配置选项中的任何一个都会导致选择 MXC HDMI Core 配置选项 CONFIG_MFD_MXC_HDMI。可以通过“Device Drivers > Multifunction device drivers > MXC HDMI Core”选择此选项

CONFIG_MXC_HDMI_CEC 选项支持 HDMI CEC 驱动，可以通过“Device Drivers > MXC support drivers > MXC HDMI CEC (Consumer Electronic Control) support”进行选择

6.4.6.9 外接 HDMI

6.4.6.9.1 介绍

高清多媒体接口（HDMI）驱动支持外部 SiI9022 HDMI 硬件模块，能够使用一根电缆传输未压缩视频、音频和数据。

HDMI 驱动分为两个子组件：与 Linux Frame Buffer API 集成的视频显示设备驱动和将 S/PDIF 音频数据传输到 SiI9022 HDMI 硬件模块的 S/PDIF 音频驱动。

HDMI 驱动仅用于演示应用，支持以下功能：

- HDMI 视频输出支持 1080p60 和 720p60 分辨率。
- 支持从 HDMI 接收设备读取视频 EDID 信息。
- 热插拔检测
- HDMI 音频播放（2 个通道，16/24 位，采样率为 44.1 KHz）

i.MX 6 7ULP SoC 支持外接 HDMI。

输出数据通过 3 个最小转换差分信号（TMDS）通道传输到 SoC 外部的 HDMI 接收设备。此外，HDMI 还承载了 VESA 数据显示通道（DDC）。DDC 是一个 I2C 接口，允许 HDMI 发送器向 HDMI 接收器查询扩展显示标识数据（EDID）。CEC 通道提供可选的对发送设备和接收设备的高级控制功能。

6.4.6.9.2 软件操作

HDMI 驱动根据其两个主要用途分为两个子组件：向 HDMI 接收设备提供视频和音频。

音频输出取决于视频显示。

6.4.6.9.3 源代码结构

HDMI 驱动的源代码分为 HDMI 显示驱动和 HDMI 音频驱动。

HDMI 显示驱动源文件位于 drivers/video/fbdev/mxc 中。HDMI 音频驱动源文件位于 sound/soc/fsl 中。

表 71. HDMI 源文件

文件	说明
drivers/video/fbdev/mxc/mxsfb_sii902x.c	HDMI 显示驱动实施
sound/soc/fsl/imx-spdif.c	S/PDIF Audio SoC Machine 驱动实施。
sound/soc/fsl/fsl_spdif.c	S/PDIF Audio SoC DAI 驱动实施。

6.4.6.9.4 菜单配置选项

有两个主要的 Linux 内核配置选项用于在 Linux 操作系统映像中选择并包含 HDMI 驱动功能。

启用 HDMI 支持需要以下配置选项。

The CONFIG_FB_MXS_SII902X 选项支持 Sii902x HDMI 视频驱动，可以通过“Device Drivers > Support for frame buffer devices > Si Image SII9022 DVI/HDMI Interface Chip”进行选择。

i.MX 6Sololite 上的 HDMI 视频依赖于 MXC ELCDIF 帧缓冲区。

CONFIG_SND_SOC_IMX_SII902X 选项支持 HDMI 音频驱动，并且可以通过“Device Drivers > Sound card support > ALSA for SoC audio support > ommon SoC Audio options for Freescale CPUs: > SoC Audio support for i.MX boards with sii902x”进行选择。

6.5 Video for Linux 2（V4L2）

6.5.1 介绍

Video for Linux 2（V4L2）驱动是 V4L2 框架的插件，支持摄像头采集和显示。

一些 i.MX SoC 基于相关的图像处理单元和采集硬件支持 V4L2。

如需了解有关 V4L2 的更多信息，请访问 Linux Media Subsystem Documentation 中的 Linux Video for Linux 2 API 规范。

V4L2 API 支持摄像头和显示器控制，但 i.MX 8 仅支持 V4L2 采集，不支持显示，使用 DPU 进行显示控制。i.MX 6 和 i.MX 7 可采集和显示 V4L2。

6.5.1.1 i.MX 8 DPU V4L2

i.MX 8 上的 Video for Linux 2（V4L2）驱动是 V4L2 框架的插件，仅支持使用显示处理单元（DPU）进行摄像头采集。

V4L2 DPU 摄像头驱动仅支持基本采集。V4l2 采集设备从摄像头或电视解码器获取传入的视频图像，并将图像采集到存储器中。V4L2 驱动支持的功能如下所示：

- 用于采集接口的 RGB 24 位和 YUV 4:2:2 交错格式
- 不同传感器驱动的插件
- 流式（排队）输入缓冲区
- 可编程输入和输出像素格式和大小
- RGB 16、24 和 32 位、YUV 4:2:0 和 4:2:2 交错输入格式

在使用 V4L2 摄像头功能之前，必须先运行命令 modprobe mxc_v4l2_capture。

6.5.1.2 PxP V4L2

PxP 的 Video for Linux Two（V4L2）驱动仅用于显示输出。

6.5.1.3 带 IPU V4L2 的 i.MX 6

Video for Linux Two（V4L2）驱动是 V4L2 框架的插件，支持摄像头和预处理功能，以及视频和后处理功能。V4L2 摄像头驱动支持所有摄像头相关功能。V4l2 采集设备从摄像头或视频流中获取传入的视频图像，并对其进行处理。输出设备获取视频并对其进行处理，然后将其发送到显示器或类似设备。

IPU V4L2 驱动支持的功能如下所示：

- 直接预览和输出到 SDC 前景覆盖平面（与 LCD 刷新同步）
- 直接预览到图形帧缓冲区（不与 LCD 刷新同步）
- 帧缓冲区和覆盖平面的颜色键控或 alpha 混合
- 从 IPU 编码通道进行流式传输（排队）采集
- 直接（原始拜耳）静态采集（取决于传感器）
- 用于预览和采集的可编程像素格式、大小、帧速率
- 使用自定义 API 进行可编程旋转和翻转
- RGB 16 位、24 位和 32 位预览格式
- 原始拜耳（仅静态，取决于传感器）、RGB 16、24 和 32 位、YUV 4:2:0 和 4:2:2 平面、YUV 4:2:2 交错和 JPEG 格式用于采集
- 控制传感器属性，包括公开、白平衡、亮度、对比度等
- 不同传感器驱动的插件
- 链路后处理调整大小和 CSC、旋转和显示 IPU 通道
- 流式（排队）输入缓冲区
- 覆盖和中间（旋转）缓冲区的双重缓冲
- 输入缓冲区的可配置 3+ 缓冲
- 可编程输入和输出像素格式和大小
- 可编程缩放和帧速率
- RGB 16、24 和 32 位、YUV 4:2:0 和 4:2:2 平面以及 YUV 4:2:2 交错输入格式
- 电视输出

命令 modprobe mxc_v4l2_capture 必须在 V4L2 函数之前运行。

6.5.1.4 IPU V4L2 采集设备

V4L2 采集设备有两个接口：

- 采集接口——使用 IPU 预处理 ENC 通道录制 YCrCb 视频流
- 覆盖接口——使用 IPU 设备驱动将预览视频显示到 SDC 前景和背景屏。

可以在内核配置期间选择 V4L2 采集支持。该驱动有两个层。上层是通用的 Video for Linux 驱动，它包含链式缓冲区管理、流 API 和其他 ioctl 接口。此设备的文件位于

```
drivers/media/platform/mxc/capture/
```

V4L2 采集设备驱动位于 mxc_v4l2_capture.c 文件中。下层覆盖驱动位于 ipu_fg_overlay_sdc.c、ipu_bg_overlay_sdc.c 中

此代码（ipu_prp_enc.c）与 IPU ENC 硬件对接，而 ipu_still.c 与 IPU CSI 硬件对接。传感器帧速率控制由 VIDIOC_S_PARM ioctl 处理。在设置帧速率之前，传感器打开 AE 并打开 AWB。帧速率可能会根据光传感器样本而变化。

特定摄像头的驱动位于

```
drivers/media/platform/mxc/capture/
```

6.5.2 V4L2 采集设备

V4L2 采集设备有两个接口：

- 采集接口——使用 i.MX 处理引擎录制 YCrCb 视频流
- 覆盖接口——使用 i.MX 处理引擎将预览视频显示到 SDC 前景和背景屏。

该驱动有两个层。上层是通用的 Video for Linux 驱动，它包含链式缓冲区管理、流 API 和其他 ioctl 接口。下层是 i.MX SoC 实施，用于与每个 V4l2 SoC 章节中详细介绍的 SoC 相关联的显示引擎。

6.5.2.1 V4L2 Capture IOCTL

目前，支持存储器映射流 API。支持的 V4L2 IOCTL 如下所示：

- VIDIOC_QUERYCAP
- VIDIOC_G_FMT
- VIDIOC_S_FMT
- VIDIOC_REQBUFS
- VIDIOC_QUERYBUF
- VIDIOC_QBUF
- VIDIOC_DQBUF
- VIDIOC_STREAMON
- VIDIOC_STREAMOFF
- VIDIOC_OVERLAY
- VIDIOC_G_FBUF
- VIDIOC_S_FBUF
- VIDIOC_G_CTRL
- VIDIOC_S_CTRL
- VIDIOC_CROPCAP
- VIDIOC_G_CROP
- VIDIOC_S_CROP
- VIDIOC_S_PARM
- VIDIOC_G_PARM
- VIDIOC_ENUMSTD
- VIDIOC_G_STD
- VIDIOC_S_STD
- VIDIOC_ENUMOUTPUT

- VIDIOC_G_OUTPUT
- VIDIOC_S_OUTPUT

V4L2 控制代码已扩展，支持旋转。ID 为 V4L2_CID_PRIVATE_BASE。支持的值如下所示：

- 0：正常运行
- 1：垂直翻转
- 2：水平翻转
- 3：180° 旋转
- 4：90° 顺时针旋转
- 5：90° 顺时针旋转和垂直翻转
- 6：90° 顺时针旋转和水平翻转
- 7：90° 逆时针旋转

下图是 V4L2 Capture API 交互的框图。

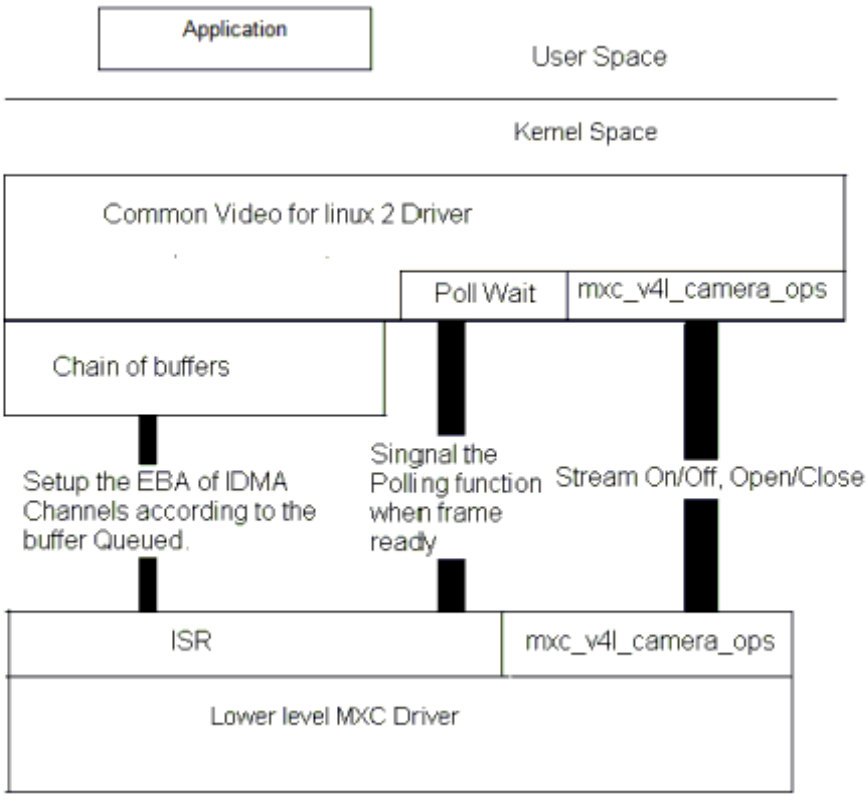


图 26. Video4Linux2 Capture API 交互

6.5.2.2 V4L2 Capture API 的使用

本节介绍了一个 V4L2 采集示例流程。应用执行以下步骤：

1. 通过 IOCTL VIDIOC_S_FMT 设置采集像素格式和大小。
2. 通过 IOCTL VIDIOC_S_CTRL 设置旋转使用的控制信息。

3. 使用 IOCTL VIDIOC_REQBUFS 请求缓冲区。常见的 V4L2 驱动会创建一个缓冲链（目前最大帧数为 3）。
4. 存储器将缓冲区映射到其用户空间。
5. 使用 IOCTL 命令 VIDIOC_QBUF 对缓冲区进行排队。
6. 使用 IOCTL VIDIOC_STREAMON 启动流。此 IOCTL 启用 i.MX Processing Enginee 任务和 IDMA 通道。当一个帧的处理完成后，驱动切换到排队等待下一帧的缓冲区。驱动还向信号量发出信号，指示缓冲区已准备就绪。
7. 使用 IOCTL VIDIOC_DQBUF 从队列中获取缓冲区。此 IOCTL 阻塞，直到 ISR 驱动向其发出信号。
8. 将缓冲区存储到 YCrCb 文件。
9. 再次执行 VIDIOC_QBUF，替换 V4L2 驱动队列中的缓冲区。

对于 V4L2 静态图像采集流程，应用执行以下步骤：

1. 通过执行 IOCTL VIDIOC_S_FMT 设置采集像素格式和大小。
2. 使用 YUV422 读取一帧静态图像。

对于 V4L2 覆盖支持用例，应用执行以下步骤：

1. 通过 IOCTL VIDIOC_S_FMT 设置覆盖窗口。
2. 通过 IOCTL VIDIOC_OVERLAY 打开覆盖任务。
3. 通过 IOCTL VIDIOC_OVERLAY 关闭覆盖任务。

6.5.3 V4L2 输出设备

该驱动为输出设备实施标准的 V4L2 API。可以在内核配置期间选择 V4L2 输出设备支持。该驱动位于

```
drivers/media/platform/mxc/output/mxc_vout.c
```

6.5.3.1 V4L2 输出 IOCTL

目前，支持存储器映射流 API。支持的 V4L2 IOCTL 如下所示：

- VIDIOC_QUERYCAP
- VIDIOC_REQBUFS
- VIDIOC_G_FMT
- VIDIOC_S_FMT
- VIDIOC_QUERYBUF
- VIDIOC_QBUF
- VIDIOC_DQBUF
- VIDIOC_STREAMON
- VIDIOC_STREAMOFF
- VIDIOC_G_CTRL
- VIDIOC_S_CTRL
- VIDIOC_CROPCAP
- VIDIOC_G_CROP

- VIDIOC_S_CROP
- VIDIOC_ENUM_FMT

V4L2 控制代码已被扩展，以支持反交错运动。对于此用途，ID 为 V4L2_CID_MXC_MOTION。支持的值如下所示：

- 0：中速运动
- 1：低速运动
- 2：高速运动

6.5.3.2 V4L2 输出 API 的使用

本节介绍使用 V4L2 输出 API 的 V4L2 输出示例流程。应用执行以下步骤：

1. 使用 IOCTL VIDIOC_S_FMT 设置输入像素格式和大小。
2. 使用 IOCTL VIDIOC_S_CTRL 设置控制信息，用于旋转、反交错运动（如果需要）。
3. 使用 IOCTL VIDIOC_S_CROP 设置输出信息。
4. 使用 IOCTL VIDIOC_REQBUFS 请求缓冲区。通用 V4L2 驱动创建一个缓冲链（尚未分配）。
5. 存储器将缓冲区映射到其用户空间。
6. 执行 IOCTL VIDIOC_QUERYBUF 来查询缓冲区。
7. 将需要后处理的数据传递到缓冲区。
8. 使用 IOCTL 命令 VIDIOC_QBUF 对缓冲区进行排队。
9. 执行 IOCTL VIDIOC_DQBUF 将缓冲区出列。
10. 通过执行 IOCTL VIDIOC_STREAMON 启动流。
11. 通过执行 IOCTL VIDIOC_STREAMOFF 停止流。

6.5.4 软件操作

6.5.4.1 源代码结构

下表列出了与 V4L2 驱动关联的源文件和头文件。

这些文件位于 drivers/media/platform/mxc

表 72. V4L2 驱动文件

文件	说明
drivers/media/platform/mxc/output/mxc_vout.c	MX6 和 MX7 V4L2 输出设备驱动
drivers/media/platform/mxc/output/mxc_pxp_v4l2.c	V4L2 PxP 输出设备驱动
drivers/media/platform/mxc/output/mxc_pxp_v4l2.h	V4L2 PxP 输出设备驱动头文件
drivers/media/platform/mxc/capture/mxc_v4l2_capture.c	V4L2 采集设备驱动
drivers/media/platform/mxc/capture/mxc_v4l2_capture.h	V4L2 采集设备驱动的头文件
drivers/media/platform/mxc/capture/ipu_bg_overlay_sdc.c	IPU 同步背景驱动
drivers/media/platform/mxc/capture/ipu_fg_overlay_sdc.c	IPU 同步前景驱动

下页继续.....

表 72. V4L2 驱动文件（续）

文件	说明
drivers/media/platform/mxc/capture/ipu_prp_sw.h	IPU 预处理头文件
drivers/media/platform/mxc/capture/ipu_still.c	IPU 预处理静态图像采集驱动
drivers/media/platform/mxc/capture/ipu_prp_vf_sdc_bg.c	IPU 预处理取景器（同步背景）
drivers/media/platform/mxc/capture/ipu_prp_enc.c	IPU 预处理编码器驱动
drivers/media/platform/mxc/capture/ipu_csi_enc.c	IPU CSI 接口驱动

V4L2 摄像头的驱动位于 divers/media/platform/mxc/capture。

V4L2 输出的驱动位于 drivers/media/platform/mxc/output。

6.5.4.2 菜单配置选项

下面提供了内核配置选项。

Device Drivers > V4L platform devices > MXC Video For Linux Video Capture

Device Drivers > V4L platform devices > MXC Video For Linux Video Output

6.6 视频模数转换器（VADC）

6.6.1 介绍

视频模数转换器（VADC）由模拟视频前端（AFE）和数字视频解码器组成。AFE 接受来自模拟摄像头等设备的 NTSC 或 PAL 输入。

这两个组件在 VADC 驱动中进行配置。视频解码器输出 YUV444 格式的数据。

视频 ADC 具有以下功能：

- 内部电压和电流参考发生器
- 10 位分辨率（9.5 位 ENOB，66.5 Msps）
- 4 个模拟输入，所有输入都可用于 CVBS
- 可编程抗混叠滤波器、增益和钳位

视频解码器具有以下功能：

- NTSC/PAL 解码器
- 直接数据通路（无需复杂的重采样）
- 自动标准检测
- 2D 自适应梳状滤波器
- 数据通路/时钟架构包含 VCR 信号的时基校正器
- Luma 通带平坦至 > 6 MHz

6.6.2 软件操作

VADC 驱动位于 Linux V4L2 架构下，它实施了 V4L2 采集接口。应用无法直接使用该摄像头驱动，而是使用 V4L2 采集驱动打开和关闭摄像头，以进行图像采集。

V4L2 采集支持以下操作：

- 采集流模式

支持以下图片格式：

- YUV444

支持以下图片尺寸：

- PAL
- NTSC

6.6.3 源代码结构

下表列出了 drivers/media/platform/mxc/capture 中提供的 VADC 驱动源文件。

表 73. VADC 驱动文件

文件	说明
drivers/media/platform/mxc/capture/mxc_vadc.c	VADC 驱动源代码
drivers/media/platform/mxc/capture/mxc_vadc.h	VADC 驱动头文件

6.6.4 菜单配置选项

在菜单配置中启用以下模块：

Device Drivers > Multimedia devices > Video capture adapters > MXC Video For Linux Camera > MXC VADC support

6.6.5 DTS 配置

VADC 模拟输入可选择[0-3]。CSI1 或 CSI2 可用于采集 VADC 数据，可以在 DTS 文件中对其进行配置。

例如：

```
vadc_in = <0>;  /* VADC input select */
csi_id = <1>; /* CSI select */
```

选择 vin1 和 CSI2 的 VADC 输入用于采集 VADC 数据。

6.7 视频处理单元（VPU）

6.7.1 介绍

VPU 硬件执行所有编解码器计算和大部分比特流解析/打包任务。因此，该软件只需要较少的控制和工作便可实施复杂而高效的多媒体编解码系统。

i.MX 6 和 i.MX 8 SoC 支持不同的 VPU。下表列出了各种 VPU。

表 74. VPU

SoC	VPU 供应商	库
i.MX 6	Chips and Media	imx-vpu.so
i.MX 8M Quad、8M Mini 和 8M Plus	Hantro	imx-hantro.so
i.MX 8QuadMax、i.MX 8QuadXPlus	Amphion	无库

注意

Malone 是解码器而 Windsor 是编码器。两者都由 Amphion 提供。

Hantro 代表以下提供程序：

- hantro/（8mq/8mp 解码器）
- hantro_845/（8mini 解码器）
- hantro_845_h1/（8mini 编码器）
- hantro_vc8000e（8mp 编码器）

6.7.2 软件操作

VPU 软件可以分为两部分：内核驱动和用户空间库，以及用户空间中的应用。内核驱动负责系统控制和预留资源（存储器/IRQ）。它为用户空间中的应用层提供了一个 IOCTL 接口，作为访问系统资源的路径。用户空间中的应用调用相关的 IOCTL 和编解码器库函数来实施复杂的编解码器系统。

VPU 内核驱动提供以下功能：

- 模块初始化，使用设备特定的结构对模块进行初始化
- 设备初始化，对 VPU 时钟和硬件进行初始化并请求 IRQ
- 中断服务例程，支持一帧已完成事件
- 文件操作例程，为用户空间提供以下接口：
- 文件打开
- 文件释放
- 文件 IOCTL，为存储器分配和释放提供接口
- 存储器映射，用于用户空间中寄存器和存储器访问

VPU 用户空间驱动提供以下功能：

- 编解码器库
- 初始化编解码器系统
- 设置编解码器系统配置
- 通过命令控制编解码器系统
- 报告编解码器状态和结果
- 系统 I/O 操作
- 请求和释放存储器
- 将存储器/寄存器映射到用户空间和取消映射
- 设备管理

用于简单验证的用户空间应用：

- 读取视频原始数据
- YUV 文件转储
- 配置编解码器行为的常规选项

下图展示了 H.264 示例中所示的简单工作流程。

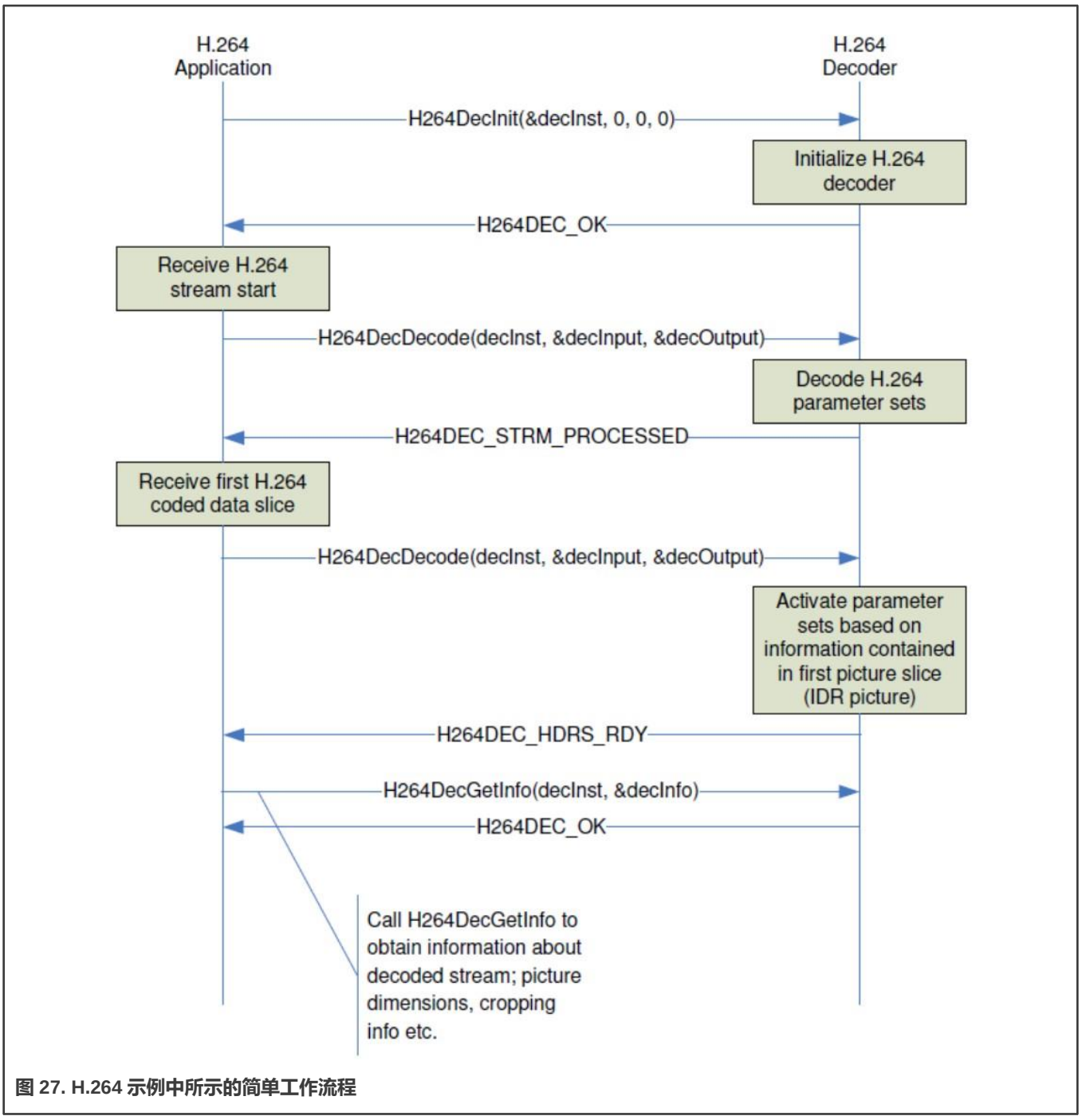
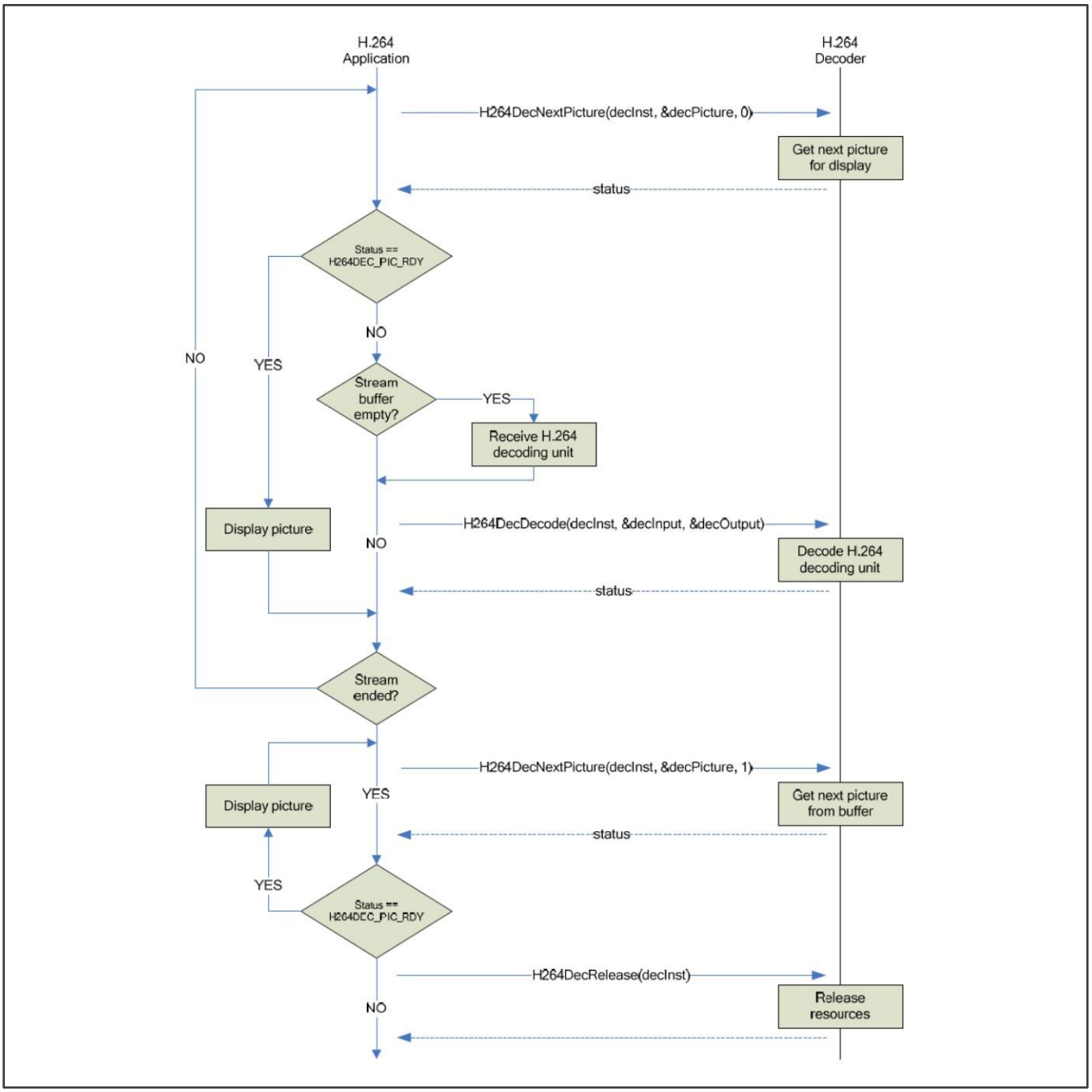


图 27. H.264 示例中所示的简单工作流程



VPU 模块只有一个用户空间编程接口。应用层的用户不能直接访问内核驱动接口。VPU 库为用户访问内核驱动接口。有一个统一的接口可包装所有视频格式。以下是相关的 API：

```
CODEC_STATE decoder_decode_xxx(CODEC_PROTOTYPE * arg,STREAM_BUFFER * buf, OMX_U32 * consumed,FRAME *
frame);
CODEC_STATE decoder_getinfo_xxx(CODEC_PROTOTYPE * arg,STREAM_INFO * pkg);
CODEC_STATE decoder_setppargs_xxx(CODEC_PROTOTYPE * codec,PP_ARGS * args);
CODEC_STATE decoder_setframebuffer_xxx(CODEC_PROTOTYPE * arg, BUFFER *buff,OMX_U32 available_buffers);
CODEC_STATE decoder_pictureconsumed_xxx(CODEC_PROTOTYPE * arg, BUFFER *buff);
CODEC_STATE decoder_getframe_mpeg4(CODEC_PROTOTYPE * arg, FRAME * frame,OMX_BOOL eos);
```

```
FRAME_BUFFER_INFO decoder_getframebufferinfo_xxx(CODEC_PROTOTYPE * arg);
CODEC_STATE decoder_endofstream_xxx(CODEC_PROTOTYPE * arg)
OMX_S32 decoder_scanframe_xxx(CODEC_PROTOTYPE * arg, STREAM_BUFFER * buf,OMX_U32 * first, OMX_U32 *
last);
CODEC_STATE decoder_abort_xxx(CODEC_PROTOTYPE * arg);
CODEC_STATE decoder_abortafter_xxx(CODEC_PROTOTYPE * arg);
CODEC_STATE decoder_setnoreorder_xxx(CODEC_PROTOTYPE * arg, OMX_BOOL no_reorder);
static void decoder_destroy_xxx(CODEC_PROTOTYPE * arg)
```

6.7.3 源代码结构

下表列出了 drivers/mxc/vpu 中的内核空间源文件。

表 75. VPU 驱动文件

文件	说明
drivers/mxc/vpu/mxc_vpu.c	Chips and Media VPU 驱动
include/linux/mxc_vpu.h	Chips and Media VPU 头文件
include/linux/mxc_vpu-malone.h	Malone VPU 头文件
drivers/mxc/vpu-malone/mxc_vpu-malone.c	Malone VPU 驱动
drivers/mxc/vpu_windsor/vpu_rpc.c	Malone VPU B0 Decoder RPC
drivers/mxc/vpu_windsor/vpu_rpc.h	Malone VPU B0 RPC Decoder 头文件
drivers/mxc/vpu_windsor/vpu_b0.h	Malone VPU B0 Decoder 头文件
drivers/mxc/vpu_windsor/vpu_b0.c	Malone VPU B0 Deocder 驱动
drivers/mxc/vpu_windsor/vpu_encoder_rpc.h	Malone VPU B0 Encoder RPC 头文件
drivers/mxc/vpu_windsor/vpu_encoder_rpc.c	Malone VPU B0 Encoder RPC 驱动
drivers/mxc/vpu_windsor/vpu_encoder_b0.h	Malone VPU Encoder B0 头文件
drivers/mxc/vpu_windsor/vpu_encoder_b0.c	Malone VPU Encoder B0 驱动
drivers/mxc/vpu_windsor/hantrodec.c	Hantro 8MQuad VPU Decoder 驱动
include/linux/hantrodec.h	Hantro Decoder 内核头文件
include/uapi/linux/hantrodev.h	Hantro Decoder 用户空间头文件
drivers/mxc/hantro_845/hantrodec_845s.c	Hantro 8MMini VPU Deocder 驱动
drivers/mxc/hantro_845_h1/hx280enc.c	Hantro 8MMini HL Encodeer 驱动
drivers/mxc/hantro_845_h1/hx280enc.h	Hantro 8MMini HL Encodeer 头文件

下表列出了 i.MX 6 中的用户空间库源文件

```
imx-vpu-(version)/vpu
```

目录：

表 76. MX6 VPU 库文件

文件	说明
vpu_io.c	与内核驱动对接的接口，用于打开 VPU 设备并分配存储器
vpu_io.h	IOCTL 的头文件
vpu_lib.c	用户空间中的核心编解码器实施
vpu_lib.h	编解码器的头文件
vpu_reg.h	VPU 的寄存器定义
vpu_util.c	实施通用实用程序的文件
vpu_util.h	头文件

下表列出了以下目录中提供的固件文件：

```
firmware-imx-(version)/lib/firmware/vpu/ directory
```

表 77. VPU 固件文件

文件	说明
vpu_fw_imx6xxx.bin	i.MX 6 VPU 固件
vpu_fw_imx8xxx.bin	i.MX 8 VPU 固件

6.7.4 菜单配置选项

在菜单配置中，为 VPU 驱动启用以下模块：

对于带 VPU 的 i.MX 6，选择“Device Drivers > MXC support drivers > Support for MXC VPU (Video Processing Unit)”

对于 i.MX 8M，选择“Device Drivers > MXC support drivers > MXC HANTRO (Video Processing Unit) support”

对于 i.MX 8QuadMax 和 i.MX 8QuadXPlus，选择“Device Drivers > MXC support drivers > Support for MXC VPU (Video Processing Unit) DECODER MXC, and VPU (Video Processing Unit) WINDSOR ENCODER support”

6.8 JPEG 编码器和解码器

6.8.1 介绍

JPEG 编码器由一个 JPEG-E-X 核心和一个 JPEG 编码器包装器（JPGENCWRP）组成。同样，JPEG 解码器由 JPEG 解码器核心（JPEG-D-X）及其相应的包装器组成。

JPEG 核心符合行业标准基准和扩展 ISO/IEC 10918-1 JPEG，《*i.MX 8DualXPlus 应用处理器参考手册*》（IMX8DQXPRM）中记录了一些限制。

JPEG 编码器包装器（JPGENCWRP）与 Cast JPEG 编码器核心配合使用。它提供配置模式和编码模式。

- 在配置模式下，它可以从系统存储器中获取配置比特流并将其馈送到编码器。
- 在编码模式下，它可以通过 AXI 总线接口获取图像像素数据并馈送到 Encoder Core （编码器核心）进行编码。

同样，JPEG 解码器包装器为 Cast JPEG 解码器核心提供了接口。

JPEG 包装器对描述符进行编码，通过上下文切换支持多种图像编码。有 4 个比特流槽。其中每个都可以通过链式描述符独立启用。

JPEG 编码器和解码器支持 8K（0x2000）像素的最大水平分辨率。水平分辨率需要为 8 的整数倍。垂直分辨率也是如此。对于 YUV422 和 YUV420，分辨率必须为 16 的倍数。图像尺寸可能高达 64K x 64K。

6.8.2 JPEG 编码器和解码器驱动概述

该驱动依赖于 V4L2 框架。

JPEG 编码器和解码器驱动实施了 V4L2 框架公开的 IOCTL 的子集，即以下 v4l2_ioctl_ops：

- VIDIOC_QUERYCAP
- VIDIOC_ENUM_FMT_VID_CAP
- VIDIOC_ENUM_FMT_VID_OUT
- VIDIOC_TRY_FMT_VID_CAP
- VIDIOC_TRY_FMT_VID_OUT
- VIDIOC_S_FMT_VID_CAP
- VIDIOC_S_FMT_VID_OUT
- VIDIOC_G_FMT_VID_CAP
- VIDIOC_G_FMT_VID_OUT
- VIDIOC_QBUF
- VIDIOC_DQBUF
- VIDIOC_CREATE_BUFS
- VIDIOC_PREPARE_BUF
- VIDIOC_REQBUFS
- VIDIOC_QUERYBUF
- VIDIOC_STREAMON
- VIDIOC_STREAMOFF

用户应用可以通过支持的 V4L2 IOCTL 与驱动进行交互。

JPEG 驱动支持通过存储器映射进行流式 I/O。当使用 VIDIOC_QUERYCAP 时，此功能通过 V4L2_CAP_STREAMING 标志公开。流式传输是一种 I/O 方法，其中在应用和驱动之间只交换指向缓冲区的指针，而不复制数据。存储器映射主要用于将设备存储器中的缓冲区映射到应用的地址空间。

JPEG 驱动支持通过单平面 API 进行缓冲区存储器映射。

如需了解有关流式 I/O 的更多信息，请参见 “流式 I/O”（存储器映射）章节。

6.8.3 JPEG 编码器/解码器驱动的限制

硬件（即 JPEG 包装器）通过上下文切换支持多种图像编码。该驱动不使用上下文切换，只使用 4 个可用插槽中的一个。硬件支持比特流缓冲区半/满，并返回比特流缓冲区管理功能，但驱动不使用这些功能。

硬件支持以下格式：YUV444、YUV420、YUV422、RGB、ARGB 和 Gray。

驱动支持以下格式：YUV444、YUV420（与 NV12 相同）、YUV422（与 YUYV 或 YUY2 相同）、RGB（有一些限制）和 Gray。不支持 ARGB 格式。

该驱动支持通过 gstreamer 对 JPEG 图像进行编码和解码，但尚不支持 MJPEG 视频。

硬件的限制是解码后的图像分辨率应大于 64x64。

硬件的限制是，解码后的图像应该至少有一个默认的霍夫曼表（jpeg 输入流中应存在 DHT 标记部分）。

第 7 章
音频

7.1 高级 Linux 声音架构片内系统（ALSA System on Chip, ASoC）声卡

7.1.1 ALSA 声卡驱动程序简介

高级 Linux 声音架构（ALSA）是目前 Linux 系统中最流行的架构，为 Linux 操作系统提供了音频和 MIDI 功能。

ALSA 具有以下显著特征：

- 高效支持所有类型的音频接口，从消费者级声卡到专业多声道音频接口。
- 完全模块化的声卡驱动程序
- SMP 和线程安全设计。
- 用户空间库（alsa-lib），可简化应用程序编程并提供更高级别的功能。
- 支持较旧的 Open Sound System(OSS)API，为大多数 OSS 程序提供二进制兼容性。

ALSA 片内系统（ASoC）层是为 SoC 音频设计的。ASoC 层的总体项目目标是为嵌入式片内系统处理器和便携式音频编解码器提供更好的 ALSA 支持。

ASoC 层还提供以下功能：

- 编解码器独立性，可在其他平台和机器上重复使用编解码器驱动程序。
- 在编解码器和 SoC 之间轻松设置 I2S/PCM 音频接口。每个 SoC 接口和编解码器都向内核注册其音频接口功能。
- 动态音频电源管理（DAPM）。这是一种 ASoC 技术，不管哪种音频用例处于活动状态，都可最大限度地降低音频子系统的功耗。DAPM 始终保证最低的音频功率状态，对用户空间音频组件完全透明。DAPM 非常适合移动设备或具有复杂音频要求的设备。
- 减少弹出和点击。按正确的顺序打开/关闭编解码器（包括使用数字静音），可以减少弹出和点击。ASoC 向编解码器发送何时更改电源状态的信号。
- 特定于机器的控制。允许机器向声卡添加控制，例如扬声器放大器的音量控制。

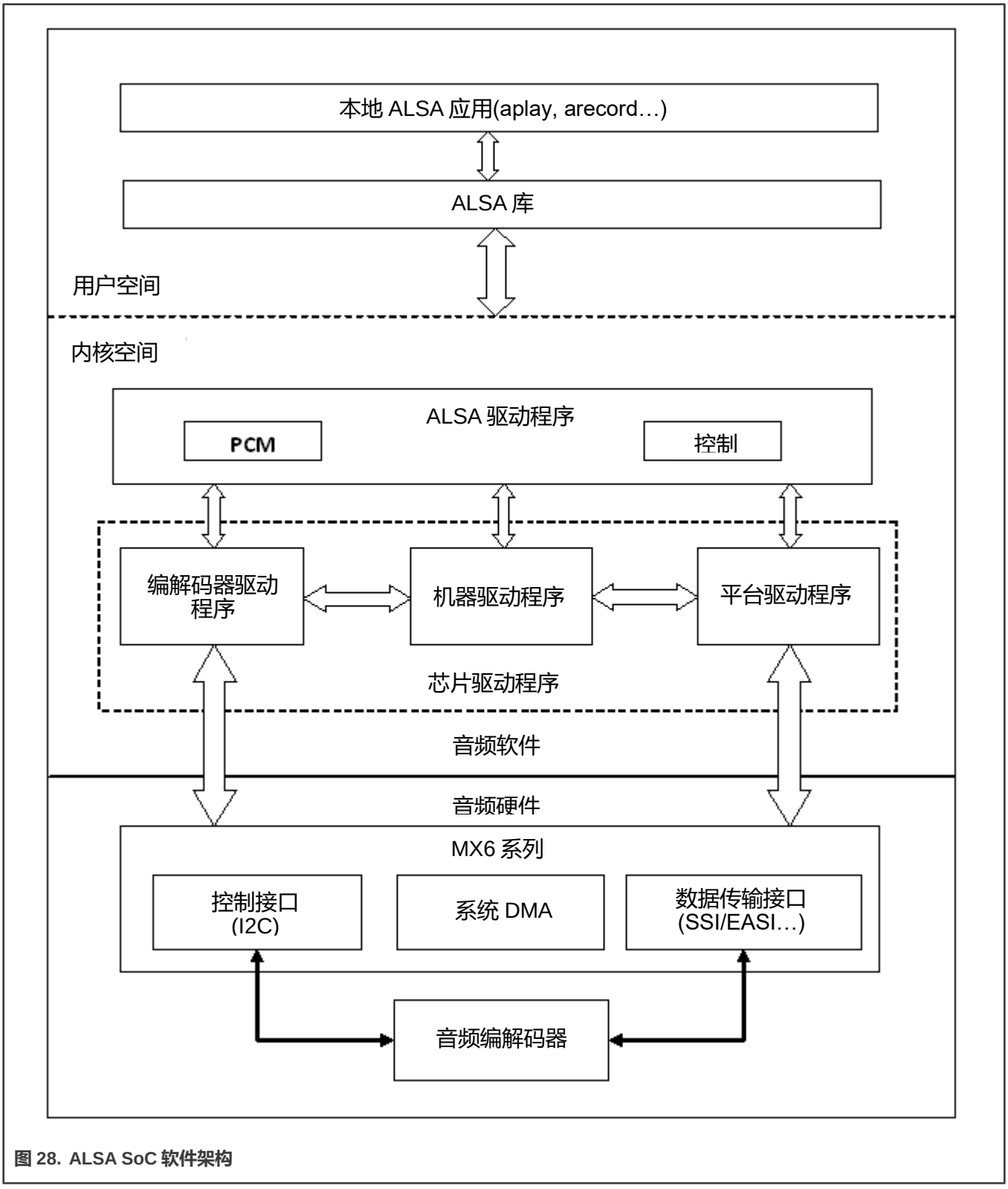


图 28. ALSA SoC 软件架构

ASoC 基本上将嵌入式音频系统分为三个组件：

- 机器驱动程序，可处理任何机器特定的控制和音频事件，例如在播放开始时打开外部放大器。

- 平台驱动程序，包含该平台的音频 DMA 引擎和音频接口驱动程序（例如，I2S、AC97、PCM）。
- 编解码器驱动程序，独立于平台，包含音频控制、音频接口功能、编解码器 DAPM 定义和编解码器 I/O 功能。

如需了解 ASoC 的更多详细信息，可以在 Linux OS 源代码树中的 Linux 内核文档中找到，该源代码树位于 linux/Documentation/ound/alsa/soc 和 www.alsa-project t.org/main/index.php/asoc。

7.1.2 SoC 声卡

目前，立体声编解码器（WM8958、WM8960、WM8962 和 WM8524）、7.1 编解码器（cs42888）和 AM/FM 编解码器（si4763）驱动程序使用 ASoC 架构来实现。

这些声卡驱动程序是独立内置的。立体声声卡支持立体声播放和采集。7.1 声卡支持多达八个声道的音频播放。启用 ASRC 时，7.1 声卡仅支持 2 或 6 声道音频播放。AM/FM 声卡支持无线 PCM 采集。

注意

仅在 i.MX 6Quad 和 i.MX 6Solo Sabre Auto 平台上支持 7.1 编解码器。
仅在 i.MX 6Quad 和 i.MX 6Solo Sabre Auto 平台上支持 AM/FM 编解码器。

7.1.2.1 立体声编解码器功能

立体声编解码器支持以下功能：

- 播放和采集的采样率分别为 8 KHz、32 KHz、44.1 KHz、48 KHz 和 96 KHz
- 声道：
  - 播放：支持两个声道
  - 采集：支持两个声道
- 音频格式：
  - 播放格式：
    - SNDRV_PCM_FMTBIT_S16_LE
    - SNDRV_PCM_FMTBIT_S20_3LE
    - SNDRV_PCM_FMTBIT_S24_LE
  - 采集格式：
    - SNDRV_PCM_FMTBIT_S16_LE
    - SNDRV_PCM_FMTBIT_S20_3LE
    - SNDRV_PCM_FMTBIT_S24_LE

7.1.2.2 7.1 音频编解码器功能

- 播放和录制的采样率：
  - 48 KHz、96 KHz、192 KHz
  - 播放：5.512 k、8 k、11.025 k、16 k、22 k、32 k、44.1 k、48 k、64 k、88.2 k、96 k、176.4 k、192 k（已启用 ASRC）
- 通道：
  - 播放通道：2、4、6、8 个通道
  - 播放通道（已启用 ASRC）：2、6 个通道
  - 采集通道：2、4 个通道

- 音频格式：
  - 播放格式
    - SNDRV_PCM_FMTBIT_S16_LE
    - SNDRV_PCM_FMTBIT_S20_3LE
    - SNDRV_PCM_FMTBIT_S24_LE
  - 播放格式（已启用 ASRC）：
    - SNDRV_PCM_FMTBIT_S16_LE
    - SNDRV_PCM_FMTBIT_S24_LE
  - 采集格式：
    - SNDRV_PCM_FMTBIT_S16_LE
    - SNDRV_PCM_FMTBIT_S20_3LE
    - SNDRV_PCM_FMTBIT_S24_LE

7.1.2.3 AM/FM 编解码器功能

- 支持的采集采样率：48 KHz
- 支持的声道：
  - 采集：支持两个声道
- 支持的音频格式：
  - 采集格式：SNDRV_PCM_FMTBIT_S16_LE

7.1.2.4 声卡信息

使用命令 aplay-l 和 arecord-l，列出以下已注册的声卡信息。例如，立体声声卡注册为 card 0。

```
root@ /$ aplay -l
**** List of PLAYBACK Hardware Devices ****
card 0: wm8962audio [wm8962-audio], device 0: HiFi wm8962-0 []
Subdevices: 1/1
Subdevice #0: subdevice #0
```

7.1.3 硬件操作

以下章节介绍 ASoC 驱动程序的硬件操作。

7.1.3.1 立体声音频编解码

立体声音频编解码器由 I^2C 接口控制。音频数据通过 DMA 通道从用户数据缓冲区传输到 SSI FIFO，或从 SSI FIFO 传回。根据音频采样位数选择 DMA 通道。AUDMUX 可建立 SSI 端口和与编解码器连接的输出端口之间的路径。编解码器在主模式下工作，并提供 BCLK 和 LRCLK。BCLK 和 LRCLK 可根据音频采样率进行配置。

WM8958、WM8960 和 WM8962 ASoC 编解码器驱动程序根据 ASoC 架构输出录音/播放/混音器 API。

编解码器驱动程序是通用且独立于硬件的代码，可配置编解码器来提供音频采集和播放功能。它不包含特定于目标平台或机器的代码。编解码器驱动程序可处理：

- 编解码器 DAI 和 PCM 配置
-

- 编解码器控制 I/O-使用 I^2C
- 混音器和音频控制
- 编解码器音频操作
- DAC 数字静音控制

模块初始化时，WM8958、WM8960 和 WM8962 编解码器注册为 I2C 客户端。API 通过 snd_soc_dai_ops 结构导出到上层。

耳机的插入/拔出可通过 GPIO 中断信号来检测。

默认情况下，启用 SSI dual FIFO 功能。

7.1.3.2 7.1 音频编解码器

7.1 音频编解码器包括 8 通道 DAC 和 4 通道 ADC，由 I2C 接口控制。音频数据通过 DMA 通道从用户数据缓冲区传输到 ESAI FIFO。根据音频采样比特选择 DMA 通道。编解码器在从机模式下工作，因为 ESAI 提供 BCLK 和 LRCLK。BCLK 和 LRCLK 可以根据音频采样率进行配置。ESAI 最多支持八个音频输出端口。在启用 ASRC 的同时，7.1 音频编解码器支持通过 ASRC 的 2 或 6 声道播放。在 i.MX 6 Sabre ARD 板上，使用带有 4 个音频输入端口的 CS42888 编解码器，每个端口接收两通道 I2S 格式的数据（网络模式），提供 8 通道播放功能。该编解码器还具有 2 个音频输出端口，连接 ESAI，提供 4 通道录音功能。

编解码器驱动程序是配置编解码器以提供音频采集和播放的通用且独立于硬件的代码。它不包含特定于目标平台或机器的代码。编解码器驱动程序处理：

- 编解码器 DAI 和 PCM 配置
- 编解码器控制 I/O-使用 I2C
- 混音器和音频控制
- 编解码器音频操作
- DAI 数字静音控制

CS42888 编解码器在模块初始化时注册为 I2C 客户端。API 通过 snd_soc_dai_ops 结构导出到上层。

7.1.3.3 AM/FM 编解码器

AM/FM 编解码器是一个虚拟编解码器，它只有一个 PCM 接口可连接到调谐器设备。音频数据通过 DMA 通道从用户数据缓冲区传输到 SSI FIFO 或从 SSI FIFO 传输。根据音频采样位数选择 DMA 通道。AUDMUX 可设置 SSI 端口和与编解码器连接的输出端口之间的路径。编解码器在主模式下工作，因为它提供 BCLK 和 LRCLK。BCLK 和 LRCLK 可根据音频采样率进行配置。

7.1.4 软件操作

以下各章节介绍 ASOC 驱动程序的软件操作。

7.1.4.1 ASoC 驱动程序源架构

imx-pcm-dma.c 文件由立体声 ALSA SoC 驱动程序、7.1 ALSA SoC 驱动程序和其他编解码器驱动程序共享。此文件负责预分配 DMA 缓冲区和管理 DMA 通道。

立体声编解码器通过 SSI 接口连接到 CPU。fsl_ssi.c 为立体声 ALSA SoC 注册 CPU DAI 驱动程序并配置片内 SSI 接口。wm8962.c 注册立体声编解码器和高保真 DAI 驱动程序。立体声编解码器上的直接硬件操作位于 wm8994.c、wm8960.c 和 wm8962.c 中。imx-wm8958.c、imx-wm8960.c 和 imx-wm8962.c 是机器层代码，它们创建驱动设备和注册立体声声卡。

多通道编解码器通过 ESAI 接口连接到 CPU。fsl_esai.c 为立体声 ALSA SoC 注册 CPU DAI 驱动器并配置片内 ESAI 接口。Cs42888.c 注册多通道编解码器和 hifi DAI 驱动程序。多声道编解码器的直接硬件操作在 cs42888.c 中。imx-cs42888.c 是创建驱动程序设备和注册立体声声卡的机器层代码。

AM/FM 编解码器通过 SSI 接口连接到 CPU。fsl_ssi.c 为立体声 ALSA SoC 注册 CPU DAI 驱动器并配置片内 SSI 接口。si476x.c 注册调谐器编解码器和调谐器 DAI 驱动程序。编解码器上的直接硬件操作位于 si476x.c 中。imx-si476x.c 是创建驱动程序设备和注册声卡的机器层代码。

7.1.4.2 声卡注册

编解码器具有相同的注册序列：

1. 编解码器驱动程序注册编解码器驱动程序、DAI 驱动程序及其操作函数。
2. 此平台驱动程序注册 PCM 驱动程序、CPU DAI 驱动程序及其操作函数，为 PCM 组件预先分配缓冲区，并根据需要设置播放和采集操作。
3. 机器层在编解码器之间创建 DAI 链路，并且 CPU 注册声卡和 PCM 设备。

7.1.4.3 设备开放

ALSA 驱动程序执行以下功能：

- 为要执行的操作分配一个空闲子流。
- 打开底层硬件设备。
- 将硬件能力分配给 ALSA 运行时信息（运行时结构包含已打开子流的所有硬件、DMA 和软件能力）。
- 配置用于操作的 DMA 读取或写入通道。
- 配置 CPU DAI 和编解码器 DAI 接口。
- 配置编解码器硬件。
- 触发转移。

第一次触发后，后续的 DMA 读写操作由 DMA 回调配置。

7.1.4.4 设备树绑定

请参阅以下文档：

- Documentation/devicetree/bindings/sound/fsl,ssi.txt
- Documentation/devicetree/bindings/sound/fsl-sai.txt
- Documentation/devicetree/bindings/sound/fsl,esai.txt
- Documentation/devicetree/bindings/sound/fsl,asrc.txt
- Documentation/devicetree/bindings/sound/wm8962.txt
- Documentation/devicetree/bindings/sound/wm8960.txt
- Documentation/devicetree/bindings/sound/wm8994.txt
- Documentation/devicetree/bindings/sound/cs42xx8.txt
- Documentation/devicetree/bindings/sound/imx-audmux.txt
- Documentation/devicetree/bindings/sound/imx-audio-wm8962.txt
- Documentation/devicetree/bindings/sound/imx-audio-cs42888.txt
- Documentation/devicetree/bindings/sound/imx-audio-si476x.txt

7.1.4.5 源代码结构

下表显示了 sound/soc/fsl 中的立体声编解码器 SoC 驱动程序源代码。

表 78. 立体声编解码器 SoC 驱动程序文件

文件	说明
sound/soc/fsl/imx-wm8958.c sound/soc/fsl/imx-wm8960.c sound/soc/fsl/imx-wm8962.c	立体声编解码器 ALSA SoC 的机器层（编解码器为 I2S 主设备）
sound/soc/fsl/imx-pcm-dma.c	立体声编解码器 ALSA SoC 的平台层
sound/soc/fsl/imx-pcm.h	PCM 驱动程序和 AUDMUX 寄存器定义用的标头文件。
sound/soc/fsl/fsl_ssi.c	立体声编解码器 ALSA SoC 的 SSI CPU DAI 驱动程序
sound/soc/fsl/fsl_ssi.h	SSI CPU DAI 驱动程序和 SSI 寄存器定义用的标头文件
sound/soc/fsl/fsl_sai.c	立体声编解码器 ALSA SoC 的 SAI CPU DAI 驱动程序
sound/soc/fsl/fsll_sai.h	SAI CPU DAI 驱动程序和 SAI 寄存器定义用的标头文件
codecs/wm8994.c codecs/wm8960.c codecs/wm8962.c	立体声编解码器 ALSA SoC 的编解码器层
codecs/wm8994.h codecs/wm8960.h codecs/wm8962.h	立体声编解码器驱动程序的标头文件

下表列出了 AM/FM 编解码器 SoC 驱动程序源文件。这些文件位于 `sound/soc`。

表 79. AM/FM 编解码器 SoC 驱动程序源文件

文件	说明
sound/soc/fsl/imx-si476x.c	立体声编解码器 ALSA SoC 的机器层（编解码器作为 I2S Slave）
sound/soc/fsl/imx-pcm-dma.c	立体声编解码器 ALSA SoC 的平台层
sound/soc/fsl/imx-pcm.h	PCM 驱动程序和 AUDMUX 寄存器定义用的标头文件
sound/soc/fsl/fsl_ssi.c	立体声编解码器 ALSA SoC 的 SSI CPU DAI 驱动程序
sound/soc/fsl/fsl_ssi.h	SSI CPU DAI 驱动程序和 SSI 寄存器定义用的标头文件
sound/soc/codecs/si476x.c	立体声编解码器 ALSA SoC 的编解码层

下表显示了多通道 ADC SoC 驱动程序源文件。

表 80. CS42888 ASoC 驱动程序源文件

文件	说明
sound/soc/fsl/imx-cs42888.c	多通道编解码器 ALSA SoC 的机器层
sound/soc/fsl/imx-pcm-dma.c	多通道编解码器 ALSA SoC 的平台层
sound/soc/fsl/imx-pcm.h	PCM 驱动程序的标头文件
sound/soc/fsl/fsl_esai.c	多通道编解码器 ALSA SoC 的 ESAI CPU DAI 驱动程序
sound/soc/fsl/fsl_esai.h	ESAI CPU DAI 驱动程序的标头文件
sound/soc/codecs/cs42xx8.c	多通道编解码器 ALSA SoC 的编解码器层
sound/soc/>codecs/cs42xx8.h	多通道编解码器驱动程序的标头文件
sound/soc/fsl/fsl_asrc.c	ASRC P2P 的 CPU DAI 驱动程序
sound/soc/fsl/fsl_asrc.h	ASRC P2P 的 CPU DAI 驱动程序头文件
sound/soc/fsl/fsl_asrc_pcm.c	ASRC P2P 的平台层

7.1.4.6 菜单配置选项

此模块提供了以下 Linux 内核配置选项。

- WM8958、WM8960 和 WM8962 编解码器的 SoC Audio 支持。在 menuconfig 中，此选项可用：

```
-> Device Drivers
   -> Sound card support
      -> Advanced Linux Sound Architecture
         -> ALSA for SoC audio support
            -> SoC Audio for Freescale CPUs
               -> SoC Audio support for i.MX boards with wm8962 (or wm8958, wm8960)
```

- i.MX cs42888 的 SoC Audio 支持。在 menuconfig 中，此选项可用：

```
-> Device Drivers
   -> Sound card support
      -> Advanced Linux Sound Architecture
         -> ALSA for SoC audio support
            -> SoC Audio for Freescale CPUs
               -> SoC Audio support for i.MX boards with cs42888
```

- AM/FM 的 SoC Audio 支持。在 menuconfig 中，此选项可用：

```
-> Device Drivers
   -> Sound card support
      -> Advanced Linux Sound Architecture
         -> ALSA for SoC audio support
            -> SoC Audio for Freescale CPUs
               -> SoC Audio support for i.MX boards with si476x
```

7.2 异步采样率转换器（ASRC）

7.2.1 简介

异步采样率转换器（ASRC）将信号的采样率转换为不同采样率的信号。ASRC 支持多达 10 个通道的并行采样率转换。每个通道的采样率转换与一对输入和输出采样率相关联。ASRC 同时支持多达三个采样率对。

7.2.1.1 硬件操作

ASRC 包括以下功能：

- 支持 1/24 至 8 之间的比率（Fsin/Fsout）范围。
- 可用于 44.1 kHz、32 kHz、48 kHz 和 96 kHz 之间的速率转换。
- 还支持 8 kHz 至 100 kHz 范围内的其他输入采样率，但性能较差（有关详细信息，请参阅 IC 规范）。
- 还支持 30 kHz 至 100 kHz 范围内的其他输出采样率，但性能较差。
- 自动调节，以减缓输入和输出采样率的变化。
- 能够容忍采样时钟抖动。
- 主要用于实时流音频使用。当输入采样时钟不可用时，可用于非实时流音频使用。
- 在任何使用情况下，必须激活输出采样时钟。
- 实时流音频的情况下，输入和输出时钟都需要可用且激活。
- 非实时流音频的情况下，可以在 ASRC 接口寄存器中设置理想的比率值来避免输入采样率时钟。

ASRC 支持轮询、中断和 DMA 模式，但为了获得更好的性能，此平台中只使用 DMA 模式。ASRC 支持以下 DMA 信道：

- 外设到外设，例如：ASRC 到 ESAI
- 存储器到外设，例如：存储器到 ASRC
- 外设到存储器，例如：ASRC 到存储器

如需了解更多信息，请参阅与 SoC 相关的应用处理器文档中的 ASRC 章节。

7.2.2 软件操作

作为音频系统中的辅助组件，ASRC 驱动程序的实现取决于平台中的用例。目前，ASRC 用于两种场景。

- Memory > ASRC > Memory，ASRC 由用户应用程序或 ALSA 插件控制。
- Memory > ASRC > peripheral，ASRC 由其他 ALSA 驱动程序直接控制。

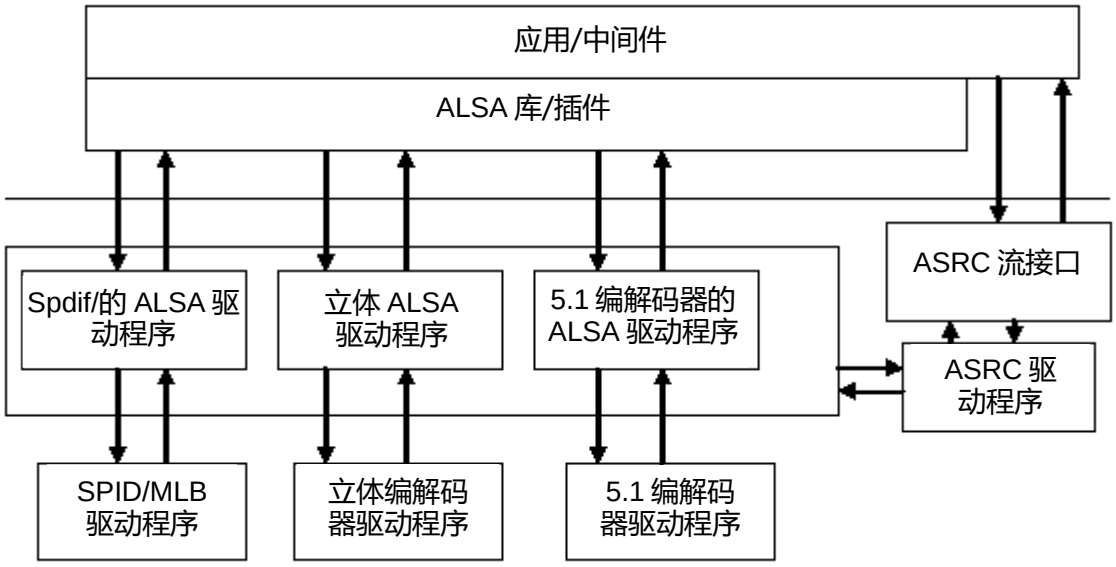


图 29. 音频驱动程序交互

如上图所示，ASRC 流接口为用户空间提供接口。ASRC 注册在/dev/mxc_asrc，并在插入模块时创建 proc 文件/proc/drive/asrc。proc 用于跟踪每对的通道号。如果没有使用所有通道对，用户可以通过 proc 文件调整频道号。通道总数应为 10 个，否则无法正确保存调整后的值。

7.2.2.1 存储器到 ASRC 再到存储器的顺序

- 打开/dev/mxc_asrc 设备
- 请求 ASRC pair - ASRC_REQ_PAIR
- 配置 ASRC pair - ASRC_CONIFG_PAIR
- 启动 ASRC - ASRC_START_CONV
- 将原始音频数据（要转换的）写入用户维护的输入缓冲区。用输入/输出缓冲区的长度和地址填充 asrc_convert_buffer struct。驱动程序将根据输出缓冲区的大小将输出数据复制到用户维护的输出缓冲区地址。重复此步骤，直到所有数据都转换完毕。-ASRC_CONVERT
- 停止 ASRC 转换 - ASRC_STOP_CONV
- 释放 ASRC 对 - ASRC_RELEASE_PAIR
- 关闭/dev/mxc_asrc 设备

7.2.2.2 存储器到 ASRC 再到外设的顺序

7.1 音频编解码器驱动程序中涉及存储器到 ASRC 到外设的音频路径。在 7.1 声卡中，指定了一个名为“cs42888audio [cs42888-audio], device 1: HiFi-ASRC-FE (*)”的新设备，用于 ASRC 的播放和采集。下面的步骤显示了调用 ASRC 到存储器再到外设的流程：

- 声音设备（PCM）已注册，并开始启用 ALSA 驱动程序中的 DMA 通道
- 请求 ASRC 对 - fsl_asrc_request_pair
- 配置 ASRC 对 - fsl_asrc_config_pair
- 启用从存储器到 ASRC 以及从 ASRC 到存储器的 DMA 通道

- 启动 DMA 通道并启动 ASRC 转换 - fsl_asrc_start_pair
- 音频数据播放完成后，停止 DMA 通道和 ASRC - fsl_asrc_stop_pair
- 释放 ASRC 对 - fsl_asrc_release_pair

7.2.2.3 源代码结构

下表列出了 sound /soc/fsl 中可用的源文件。

表 81. ASRC 源文件列表

文件	说明
sound/soc/fsl/fsl_asrc_m2m.c	ASRC M2M 驱动程序实现代码
sound/soc/fsl/imx-cs42888.c	7.1 音频编解码机驱动程序中实施存储器到 ASRC 再到 ESAI TX 的顺序
sound/soc/fsl/imx-pcm-dma.c	7.1 音频编解码器平台驱动程序中实施存储器到 ASRC 再到 ESAI TX 的顺序
sound/soc/fsl/fsl_esai.c	7.1 音频编解码器 CPU 驱动程序中实施存储器到 ASRC 再到 ESAI TX 的顺序
sound/soc/fsl/cs42xx8	7.1 音频编解码器的编解码器驱动程序中实施存储器到 ASRC 再到 ESAI TX 的顺序
sound/soc/fsl/fsl_asrc.c	ASRC P2P 的 ALSA CPU DAI 驱动程序
sound/soc/fsl/fsl_asrc.h	ASRC P2P 的 ALSA CPU DAI 驱动程序的标头文件
sound/soc/fsl/fsl_asrc_dma.c	ASRC P2P 的 ALSA 平台层
sound/soc/fsl/sound/soc/fsl/fsl_asrc_dma.c	ASRC M2M 的 ALSA 平台层

7.2.2.4 菜单配置选项

菜单配置选项如下所示：

```
-> Device Drivers
  -> Sound card support
    -> Advanced Linux Sound Architecture
      -> ALSA for SoC audio support
        -> SoC Audio for Freescale i.MX CPUs
          -> Asynchronous Sample Rate Converter (ASRC) module support
```

ASRC 驱动程序只能使用内置模块进行配置。

7.2.2.5 设备树绑定

ASRC M2M 的设备树绑定功能如下：

- compatible：兼容列表，必须包含“fsl,imx6q-asrc”。
- reg：设备寄存器设置的偏移量和长度。
- interrupts：包含 ASRC 中断。
- clocks：时钟名称中的每个条目都包含一个条目。
- clock-names：必须包含“mem”、“ipg”、“asrck”和“dma”。（通常，“dma”用于 SPBA 时钟。）。
- dmas：Documentation/devicetree/bindings/dma/dma.txt 中描述的通用 DMA 设备树绑定。
- dma-names：必须定义六个 DMA：“txa”、“rxa”、“txb”、“rxb”、“txc”、“rxc”。

- fsl,clk-map-version：不同 SOC 的映射关系不同。该版本号可用于指示时钟映射信息。
- fsl,clk-channel-bits：表示通道位数信息。

ASRC P2P 的设备树绑定功能如下：

- compatible：兼容列表，必须包含“fsl，imx6q-asrc-p2p”。
- fsl,p2p-rate：用于 Back-End (I2S)播放和录制的有效采样率。
- fsl,p2p-width：用于 Back-End (I2S)播放和录制的有效采样宽度。
- fsl,asrc-dma-rx-events：包含 ASRC Rx 的三个 SDMA 事件编号。
- fsl,asrc-dma-tx-events：包含 ASRC Tx 的三个 SDMA 事件编号。

7.2.2.6 编程接口（导出的 API 和 IOCTL）

ASRC Exported API 允许 ALSA 驱动程序使用 ASRC 服务。

以下 ASRC IOCTL 用于用户空间应用：

ASRC_REQ_PAIR：

应用 ASRC 驱动程序中的一对。分配完一对后，ASRC 内核时钟就会启用。

ASRC_CONFIG_PAIR：

配置已分配的 ASRC 对。用户负责提供 ASRC_CONFIG 结构中定义的参数。asrc_config 中的项目如下所示：

- pair：IOCTL 分配的 ASRC 对（ASRC_REQ_PAIR）。
- channel_num：通道编号。
- buffer_num：使用输入输出缓冲区所需的缓冲区编号。输入输出缓冲区在 ASRC 驱动程序内部分配。用户负责将其重新映射到用户空间。
- dma_buffer_size：输入和输出缓冲区的缓冲区大小。缓冲区大小应以页的大小为单位。通常使用 4k 字节。
- input_sample_rate：输入采样率。输入采样率应为 5.512k、8k、11.025k、16k、22k、32k、44.1k、48k、64k、88.2k 96k、176.4k、192k。
- output_sample_rate：输出采样率。输出采样率应为 32k、44.1k、48k、64k、88.2k、96k、176.4k 和 192k。
- input_word_width：输入音频数据的字宽。输入数据字宽度可以是 16 位或 24 位。
- output_word_width：输出音频数据的字宽。输出数据字宽度可以是 16 位或 24 位。
- inclk：输入时钟源可以是 ESAI RX clock、SSI1 RX clock、SSI2 RX clock、SPDIF RX clock、MLB_clock、ESAI TX clock、SSI1 TX clock、SSI2 TX clock、SPDIF TX clock、ASRCLK1 clock 和 NONE。如果使用“NONE”以外的时钟，用户应确保那个时钟可用。
- outclk：输出时钟源与输入时钟源相同。

ASRC_CONVERT：

根据 ASRC_CONFIG_PAIR 设置的参数将输入数据转换为输出数据。驱动程序将从 input_buffer_vaddr 复制 input_buffer_length 字节数据进行转换。转换后，驱动根据 ASRC 生成的数据编号填充 output_buffer_length，并将 output_buffer_length 复制到 output_buffer_vaddr。但是，在调用 ASRC_CONVERT 之前，用户负责根据输入采样率和输出采样率的比例填充 output_buffer_length。如果生成的缓冲区大小大于用户填充的 output_buffer_size，则驱动程序只会将用户填充的 output_buffer_size 复制到 output_buffer_vaddr。如果生成的缓冲区大小，小于用户填充的 output_buffer_size（差值应小于 64 字节），则调用 ASRC_CONVERT 将失败。

- input_buffer_vaddr：输入缓冲区的虚拟地址。

- output_buffer_vaddr：输出缓冲区的虚拟地址。
- input_buffer_length：输入缓冲区的长度（字节）。
- output_buffer_length：输出缓冲区的长度（字节）。

ASRC_START_CONV：

启动 ASRC 对转换。

ASRC_STOP_CONV：

停止 ASRC 对转换。

ASRC_STATUS：

查询 ASRC 对状态。

7.3 HDMI 音频

7.3.1 简介

视频章节中的“HDMI 概述”介绍了 HDMI 音频。详见 HDMI 音频。

7.4 索尼/飞利浦数字接口（S/PDIF）

7.4.1 简介

索尼/飞利浦数字接口（S/PDIF）音频模块是一个立体声收发器，支持处理器接收和传输数字音频。S/PDIF 收发器可处理 S/PDIF 通道状态（CS）和用户（U）数据。此频率测量块允许 S/PDIF RX 部分从输入的 S/PDIF 流分析出接收时钟。

7.4.1.1 S/PDIF 概述

下图是 S/PDIF 接口的框图。

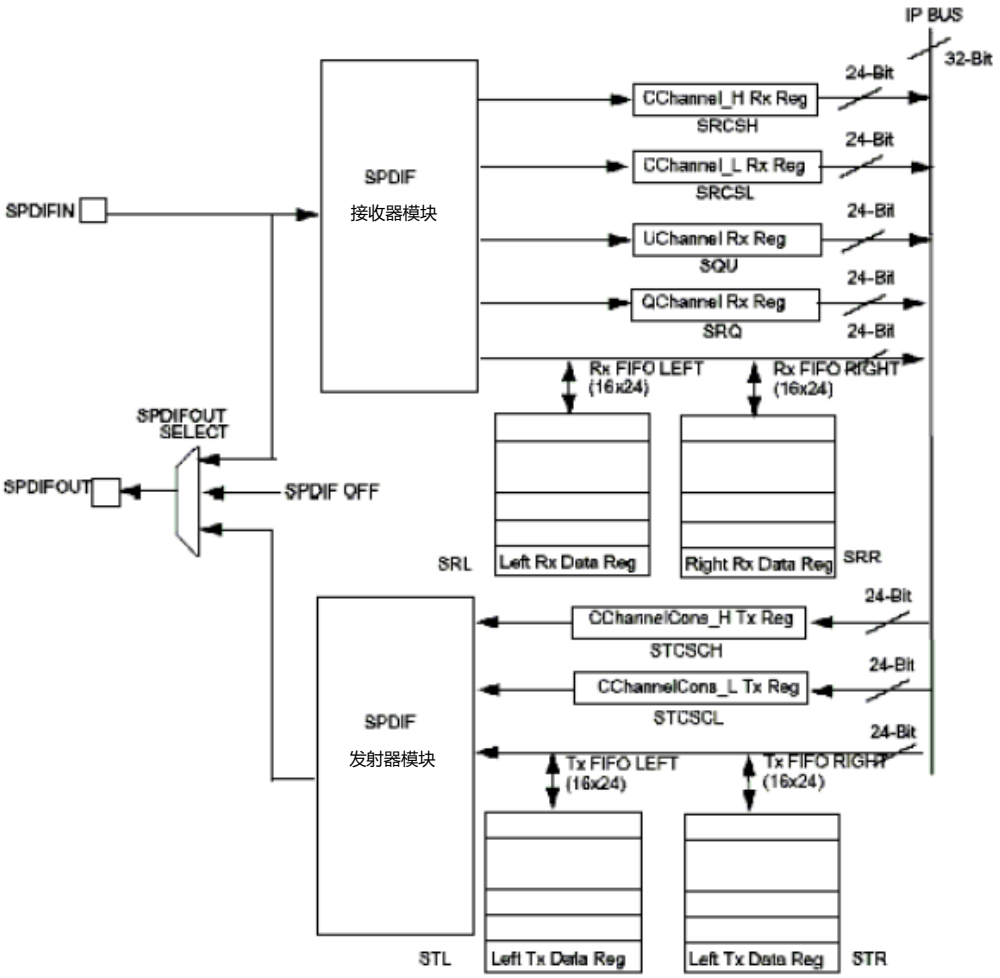


图 30. S/PDIF 收发器数据接口的框图

7.4.1.2 硬件概述

S/PDIF 由两部分组成:

- S/PDIF 接收器从每个 S/PDIF 帧中提取音频数据，并将数据放入 S/PDIF Rx 左侧 FIFO 和右侧 FIFO。通道状态和用户位也会从每一帧中提取并放入相应的寄存器中。S/PDIF 接收器提供旁路选项，可将 S/PDIF 输入信号直接传输到 S/PDIF 发射器。
- 对于 S/PDIF 发射器，处理器通过 SPDIFTxLeft 和 SPDIFTxRight 寄存器提供音频数据。通道状态位通过相应的寄存器提供。S/PDIF 发射器生成双相标记格式（IEC958）的 S/PDIF 输出比特流，它由音频数据、通道状态和用户比特组成。

在 S/PDIF 发射器中，IEC958 二相比特流在 S/PDIF 发送时钟的两侧产生。S/PDIF 发送时钟由 S/PDIF 内部时钟分频器产生，源来自 S/PDIF 块之外。S/PDIF 接收器可以从 S/PDIF 流中恢复 S/PDIF Rx 时钟。图 30 显示了 S/PDIF 收发器的时钟结构。

7.4.1.3 软件概述

S/PDIF 驱动程序在 ALSA 片内系统（ASOC）层下设计。S/PDIF 的 ASOC 驱动程序为 Tx 提供一个播放设备，为 Rx 提供一个采集设备。播放输出音频格式可以是线性 PCM 数据或具有 16 位、20 位和 24 位音频的压缩数据。支持的采样比特率为 44.1、48 或 32 kHz。采集输入音频格式可以是线性 PCM 数据或压缩的 24 位数据，支持的采样比特率为 16 到 96 kHz。该驱动程序为 PCM 和压缩数据传输提供了相同的接口。

7.4.1.4 AsoC 层

ASoC 层将嵌入式平台的音频驱动程序划分为可重用的独立层。ASoC 将音频驱动程序分为编解码器驱动程序、机器层、DAI（数字音频接口）层和平台层。Linux 内核文档在 linux/Documentation/sound/alsa/soc 中简要地介绍了这些层。对于 S/PDIF 驱动程序，我们能够重用 ssi 立体声编解码器驱动程序使用的平台层（imx-pcm-dma.c），以及在没有真正编解码器的情况下创建 DAI 链接的通用 dummy 编解码器驱动程序。

7.4.2 S/PDIF Tx 驱动程序

S/PDIF Tx 驱动程序支持以下功能。

- 32、44.1 和 48 KHz 采样率。
- 签名的 16 位和 24 位小端（Little Endian）样本格式。借助 S/PDIF SDMA 功能，24 位输出样本文件在每个通道中每帧必须有 32 位。只有 24 个最低有效位（LSB）有效。
- 在 ALSA 子系统中，受支持的格式可定义为 S16_LE 和 S24_LE。
- 立体声播放。
- 通过 iecset 或 amixer 进行信息查询。
- 可以通过使用“aplay-l”实用程序列出播放音频设备来确定设备 ID。

  例如：

```
root@ ~$ aplay -l
```

```
**** List of PLAYBACK Hardware Devices ****
```

```
card 0: imxspdif [imx-spdif], device 0: S/PDIF PCM snd-soc-dummy-dai-0 []
```

```
Subdevices: 1/1
```

```
Subdevice #0: subdevice #0
```

注意

IMX_SPDIF 行开头的数字是卡 ID。方括号中的字符串是卡名。

- ALSA 实用程序为用户空间提供了操作和使用 ALSA 驱动程序的通用方法

```
#aplay -Dplughw:0,0 audio.wav
```

注意

aplay 的-D 参数表示 PCM 设备的卡 ID 和 PCM 设备 ID，例如 hw:[card id]，[pcm device id]

“iecset”实用程序提供了设置或转储 IEC958 状态位的通用方法。

```
#iecset -c 0
```

7.4.2.1 驱动程序设计

在 S/PDIF 播放之前，需要初始化配置、中断、时钟和通道寄存器。在 S/PDIF 播放期间，通道状态位是固定的。DMA 和中断已启用。S/PDIF 在左通道和右通道上分别有 16 个 TX 采样 FIFO。当两个 FIFO 均为空时，如果已启用空中断，则会生成空中断。如果 20.8μs(1/48000)中没有重新填充数据，则会产生欠载中断。如果每次只为每个通道填充 16 个采样 FIFO，则可避免过载。如果启用自动重新同步，则硬件检查左右 FIFO 是否同步，如果不同步，则将右 FIFO 的填充指针设置为等于左 FIFO 的填充指针，并产生中断。

7.4.2.2 提供的用户界面

S/PDIF 发射器驱动程序除了提供通用的 PCM 操作接口外，还为用户提供了一个 ALSA 混音器声音控制接口。它为用户提供了将 S/PDIF 通道状态码写入驱动器的接口，以便它们可以在 S/PDIF 流中发送。该接口的输入参数为下图所示的 IEC958 数字音频结构，仅使用状态成员：

```
struct snd_aes_iec958 {
unsigned char status[24];       /* AES/IEC958 channel status bits */
unsigned char subcode[147];     /* AES/IEC958 subcode bits */
unsigned char pad;              /* nothing */
unsigned char dig_subframe[4];  /* AES/IEC958 subframe bits */
};
```

7.4.3 S/PDIF Rx 驱动程序

S/PDIF Rx 驱动程序支持以下功能：

- 16、32、44.1、48、64 和 96 kHz 接收采样率。
- 签名的 24 位小端样本格式。由于 S/PDIF SDMA 功能，PCM 记录帧中的每个通道位长度为 32 位，只有 24 个最低有效位（LSB）有效。

  在 ALSA 子系统中，所支持的格式可定义为 S24_LE。
- 立体声唱片。
- 可以通过使用“arecord-l”列出记录设备来确定设备 ID。

  例如：

```
root@ ~$ arecord -l
```

```
**** List of CAPTURE Hardware Devices ****
```

```
card 0: cs42888audio [cs42888-audio], device 0: HiFi CS42888-0 []
```

```
Subdevices: 1/1
```

```
Subdevice #0: subdevice #0
```

```
card 1: imxspdif [imx-spdif], device 0: S/PDIF PCM snd-soc-dummy-dai-0 []
```

```
Subdevices: 1/1
```

```
Subdevice #0: subdevice #0
```

- ALSA 实用程序为用户空间操作和使用 ALSA 驱动程序提供了一种通用方法。

```
#arecord -Dplughw:1,0" -c 2 -r 44100 -f S24_LE record.wav
```

注意

arecord 的-D 参数表示 PCM 设备的卡 ID 和 PCM 设备 ID，例如 hw:[card id],[pcm device id]

“iecset”实用程序提供了设置或转储 IEC958 状态位的通用方法。

```
#iecset -c 1
```

7.4.3.1 驱动程序设计

此驱动程序必须等待内部 DPLL 锁定，才能从 S/PDIF 接收器 FIFO 读取数据帧。使用高速系统时钟，内部 DPLL 可以从输入位流中提取位时钟（高级脉冲）。当该内部 DPLL 被锁定时，PhaseConfig 寄存器的 LOCK 位被设置，并且此驱动程序配置中断、时钟和 SDMA 通道。之后，此驱动程序可以同时接收音频数据、通道状态、用户位和有效位。

对于通道状态接收，在两个寄存器中总共接收 48 个通道状态位。当用户应用发出请求时，驱动程序会读出它们。

用户位的用户通道接收有两种模式：CD 和非 CD 模式。这些模式由 USyncMode（CDText_Control 寄存器的第 1 位）来确定。用户可以调用声音控制接口来设置模式（参见表 82），但无论是什么模式，此驱动程序都以相同的方式来处理用户位。对于 S/PDIF Rx，硬件模块将 Q 位从用户位复制到 QChannel 寄存器，并将用户位放入 UChannel 寄存器。此驱动程序为 U 位和 Q 位分配两个队列缓冲区。U 位队列缓冲区大小为 96x2 字节，Q 位队列缓冲区大小为 12x2 字节，队列缓冲区填充在 U/Q Full、Err 和 Sync 中断处理程序中。这意味着当 S/PDIF 驱动程序读取新的 U/Q 位时，用户可以获得先前就绪的 U/Q 位。

对于有效位接收，S/PDIF Rx 硬件块在接收时触发中断并设置中断状态。为用户提供声控界面，以获取该有效位的状态。

7.4.3.2 提供的用户界面

S/PDIF Rx 驱动程序为用户应用程序提供接口，如下表所示。

表 82. S/PDIF Rx 驱动程序接口

接口	类型	模式 [1]	参数	注释
Common PCM	PCM	-	-	PCM 打开/关闭 准备/触发 hw_params/sw_params
Rx Sample Rate	声音控制 [2]	r	整数 范围：[16000, 96000]	获取采样率。由于 DPLL 测频模块的原因，采样率精度不高。因此，用户应用必须对获取值进行校正。
USyncMode	声音控制	rw	Boolean 值：0 或 1	将 CD 模式设置为 1 将非 CD 模式设置为 0

下页继续.....

表 82. S/PDIF Rx 驱动程序接口（续）

接口	类型	模式 [1]	参数	注释
Channel Status	声音控制	r	struct snd_aes_iec958 仅使用状态[6]数组成员	-
User bit	声音控制	r	字节数组 U 位为 96 字节 Q 位为 12 字节	-
No good V bit	声音控制	r	Boolean 值：0 或 1	中断与有效标志相关联。（中断 16-SPDIFValNoGood）。每次在 SPDIF 接口上看到有效位设置为无效的帧时，就会设置该中断。

1. 模式列显示接口属性：r（读取）或 w（写入）。
2. 接口的声音控制类型由 snd_ctl_xxx()alsa-lib 函数调用

用户应用程序可以按照图 31 中的程序流程来使用 S/PDIF Rx 驱动程序。首先，应用程序打开 S/PDIF Rx PCM 设备，等待 DPLL 锁定输入比特流，并获得输入采样率。如果需要设置 USyncMode，请在读取 U/Q 位之前设置它。接下来，设置硬件参数，包括从驱动程序获取的通道数、格式和采集采样率。然后，调用准备和触发来启动 S/PDIF Rx 流读取。最后，调用读取函数来获取数据。在读取过程中，应用程序可以从驱动程序读取 U/Q 位和通道状态，并使无效位有效。

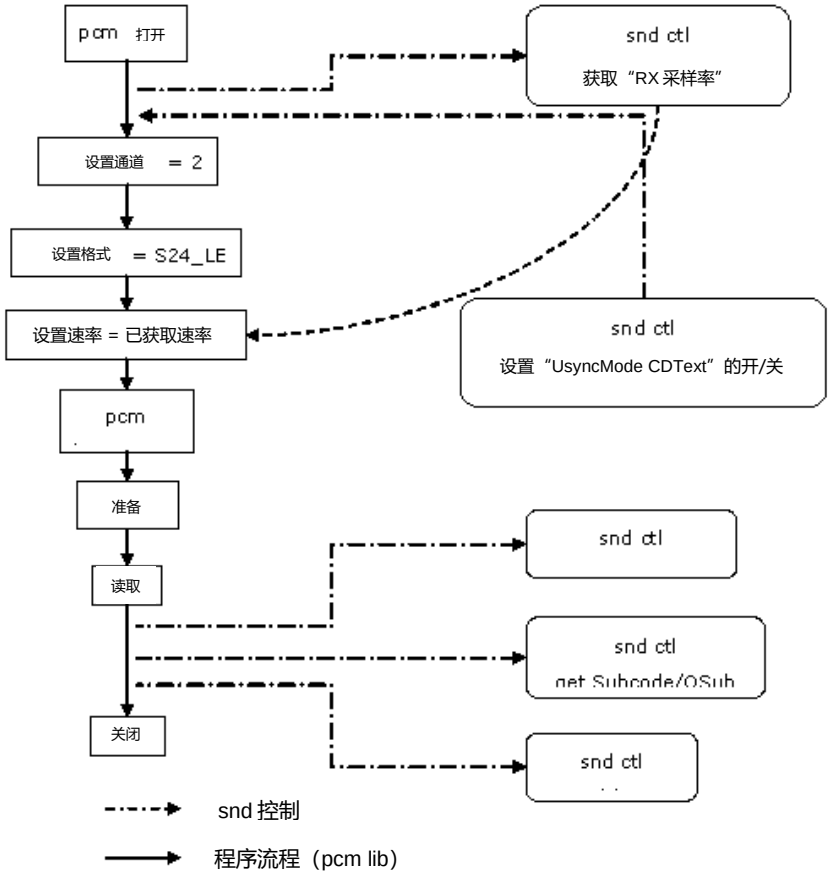


图 31. S/PDIF Rx 应用程序流程

7.4.4 源代码结构

下表列出了此驱动程序的源文件。

表 83. S/PDIF 驱动程序文件

文件	说明
sound/soc/soc-utils.c	Dummy ALSA SOC 编解码器驱动程序
sound/soc/fsl/imx-spdif.c	S/PDIF ALSA SOC 机器层
sound/soc/fsl/fsl_spdif.c	S/PDIF ALSA SOC DAI 层
sound/soc/fsl/imx-pcm-dma.c	ALSA SOC 平台层
sound/soc/fsl/imx-pcm.h	ALSA SOC 平台层标头文件

7.4.4.1 菜单配置选项

此模块提供了以下 Linux 内核配置：

在菜单配置中启用以下模块：

- CONFIG_SND_IMX_SPDIF - Configuration option for the S/PDIF driver:
- Device Drivers -> Sound card support -> Advanced Linux Sound Architecture -> ALSA for SoC audio support -> SoC Audio for Freescale i.MX CPUs -> SoC Audio support for i.MX boards with S/PDIF

7.4.4.2 设备树绑定

请参阅以下文档：

- Documentation/devicetree/bindings/sound/fsl,spdif.txt
- Documentation/devicetree/bindings/sound/imx-audio-spdif.txt

7.4.4.3 中断和异常

S/PDIF Tx/Rx 硬件块有许多中断来指示成功、异常和事件。

驱动程序处理以下中断：

- DPLL Lock and Loss Lock（DPLL 锁定和丢失锁定） - 保存 DPLL 锁定状态；在获取 Rx 采样率时使用。
- U/Q Channel Full and overrun/underrun（U/Q 通道已满和过载/欠载）- 将 U/Q 通道寄存器数据放入队列缓冲区，并更新队列缓冲区写入指针。
- U/Q Channel Sync（U/Q 通道同步） - 保存其 U/Q 数据已准备好读取的缓冲区的 ID。
- U/Q Channel Error（U/Q 通道错误）- 重置 U/Q 队列缓冲区

7.4.5 单元测试准备

为了准备运行单元测试，请执行以下操作：

- 在 PC 上安装 M-Audio Transfer 驱动程序来设置 M-Audio Transfer USB 声卡。
- 在 PC 上安装 WaveLab 工具。

7.4.5.1 Tx 测试步骤

- 将光纤插入 M-Audio 传输的[line|optical]端口。

注意

插入光纤后，确保 M-Audio 传输的[optical out]端口没有输出（红灯熄灭）。

- 启动 WaveLab，按工具栏上的录制按钮，设置录制文件名、采样率和通道号，然后录制。
- 同时，在板上，使用以下命令播放一个 Wave 文件：

```
#aplay -D hw:[card id],[pcm id] audioXXkYYS.wav
```

- aplay 结束后，停止在 WaveLab 中录制。
- 在 wavelab 中播放录制的 WAV 文件进行查看。

7.4.5.2 Rx 测试步骤

- 将光纤插入 M-Audio 传输的[optical port]
- 启动 Wavelab，打开测试 WAV 文件 audioXXkYYS.wav 循环播放

- 同时，在板上，使用以下命令录制一个 WAV 文件。录制完成后，您可以在其他电路板或 PC 上的声卡上播放录制的 WAV 文件。

```
#arecord -D hw:[card id],[pcm id] -c 2 -d 20 -r [sample rate in Hz] -f S24_LE record.wav
```

注意

arecord 命令中的采样率参数必须与 WaveLab 上播放的 WAV 文件一致。

7.5 混音器（AUDMIX）

7.5.1 简介

许多应用程序需要混合两个或多个音频才能获得不同的效果。可以使用混音器（Audio Mixer）将两个音频流混合成一个流。混音器有两个输入串行音频接口。它们由两个同步音频接口（SAI）模块来驱动。每个输入串口以 TDM（时分复用）方式在其帧中承载 8 个音频通道。混音器将来自两个接口的相应通道的音频样本混合到单个样本中。在混音之前，可以根据配置对两个输入的音频样本进行衰减。混音器的输出也是一个串行音频接口。与输入接口一样，它也具有相同的 TDM 帧格式。此输出用于驱动音频编解码器的串行 DAC TDM 接口，并随正常音频 SAI 模块的接收路径发送到外部引脚，供 CPU 回读。

混音器的输出可以从以下三个流中选择：

- 串行音频输入 1
- 串行音频输入 2
- 混合音频

混合操作与音频采样率无关，但两个音频输入流必须具有相同的音频采样率，且 TDM 帧中的通道数相同，才可以进行混合。

7.5.2 框图

下图以概览的形式显示了混音器模块。

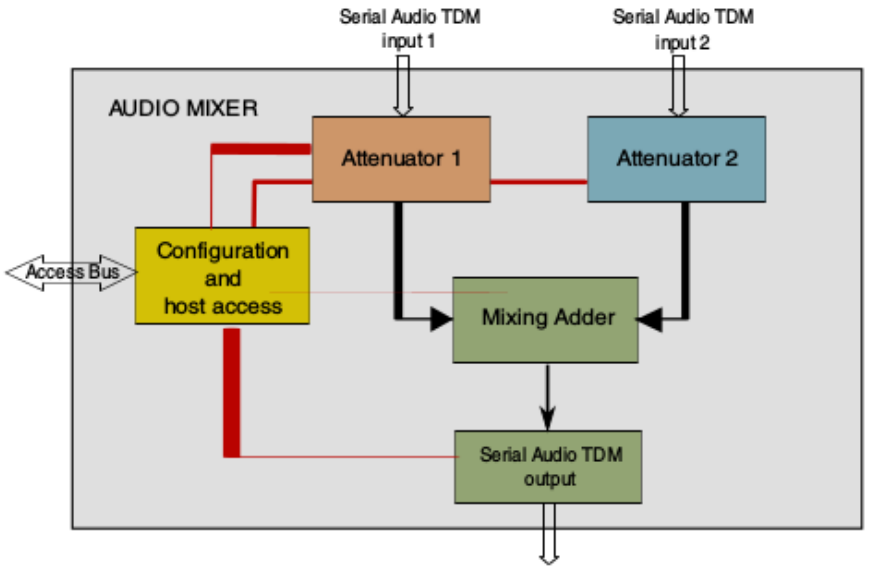


图 32. 混音器框图

7.5.3 硬件概述

混音器模块有两个串行音频输入接口，用于两个音频流。其中一个接口用于正常音频，另一个用于安全提示音。串行音频 TDM 帧可以包含 8 个样本，每个样本 32 位。前六个样本用于三个立体声 DAC。每个 DAC 分别对左声道和右声道进行两次采样。最后两个采样是额外采样，留作将来使用。在混音应用中，两路音频输入流必须具有相同的声道数和帧率。帧格式如下图所示。

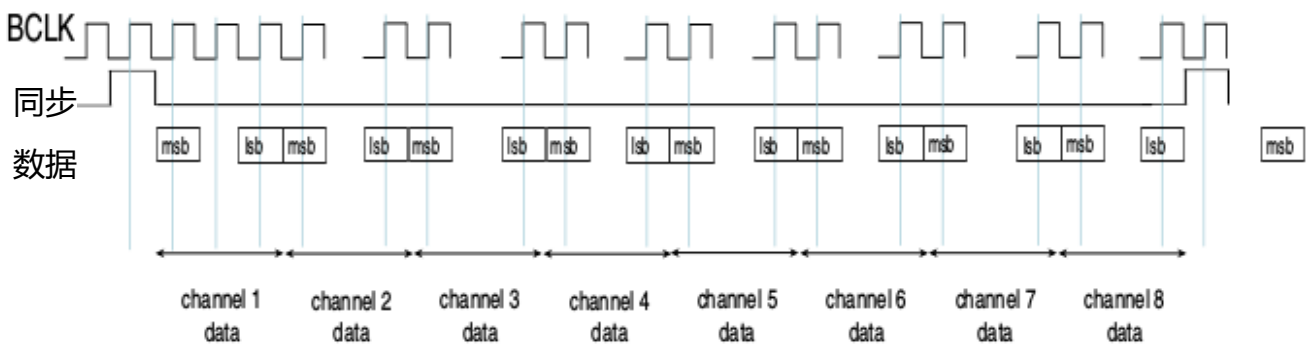


图 33. 音频 TDM 串行接口帧

输入 TDM 帧从其自身接口位时钟中的帧脉冲开始，反序列化（de-serialize）为 32 位采样。每个采样都通过衰减器。衰减器降低了音频信号的电平。这一过程称为衰减。信号的衰减是通过将音频样本乘以衰减值来实现的。衰减值定义衰减器输出端的音频信号电平。衰减可以启用或禁用。如果禁用，则音频采样无需调整即可通过。如果启用，则根据定义了不同时间衰减值的配置（称为衰减配置文件）进行衰减。

两个音频流有两个独立的衰减器。两个衰减器的输出可用来混音。混音是通过相加来自两个衰减器相应通道的采样来完成的。结果给出了混合样本值。然后对此值进行量化，获得所需的音频样本宽度。量化后的样本四舍五入后形成输出样本。量化样本的 LSB 也进行四舍五入。最终样本被序列化，并以与所选位时钟的输入接口相同的帧格式传输。

7.5.4 软件概述

混音器驱动程序是在 ALSA 片内系统（ASoC）层下设计的。混音器的 ASoC 驱动程序提供了两个播放设备进行混音器输入和一个采集设备来采集混音器输出。播放音频格式为线性 PCM 音频，其宽度为 16 位、24 位或 32 位。采集的音频格式为线性 PCM 音频数据，其宽度为 16 位、18 位、20 位、24 位或 32 位。

7.5.4.1 用户接口

混音器界面可使用 `amixer -c <audio mixer card>`工具从用户空间访问。以下混音器控件显示在用户空间中。

表 84. 混音器控件

ID	名称	类型	接入方式	值	默认情况
1	混合时钟源	enum	r/w	#0 'TDM1', #1 'TDM2'	#0 ‘TDM1’
2	输出源	enum	r/w	#0 'Disabled', #1 'TDM1', #2 'TDM2', #3 'Mixed'	#0 'Disabled'

下页继续.....

表 84. 混音器控件（续表）

ID	名称	类型	接入方式	值	默认情况
3	输出宽度	enum	r/w	#0 '16b', #1 '18b', #2 '20b', #3 '24b', #4 '32b'	#4 '32b'
4	输出 时钟极性	enum	r/w	#0 'Positive edge', #1 'Negative edge'	#1 'Negative edge'
5	帧速率差异误差	enum	r/w	#0 'Unmask', #1 'Mask'	#0 'Unmask'
6	时钟频率差异误差	enum	r/w	#0 'Unmask', #1 'Mask'	#0 'Unmask'
7	同步模式配置	enum	r/w	#0 'Disabled', #1 'Enabled'	#0 'Disabled'
8	同步模式时钟源	enum	r/w	#0 'TDM1', #1 'TDM2'	#0 'TDM1'
9	TDM1 衰减	enum	r/w	#0 'Disabled', #1 'Enabled'	#0 'Disabled'
10	TDM1 衰减方向	enum	r/w	#0 'Downward', #1 'Upward'	#0 'Downward'
11	TDM1 衰减分频器	int	r/w	min=0, max=4095	0
12	TDM1 衰减初始值	int	r/w	min=0, max=262143	262143
13	TDM1 衰减步进因子	int	r/w	min=0, max=262143	174762
14	TDM1 衰减步降因子	int	r/w	min=0, max=262143	196608
15	TDM1 衰减目标	int	r/w	min=0, max=262143	16
16	TDM2 衰减	enum	r/w	#0 'Disabled', #1 'Enabled'	#0 'Disabled'
17	TDM2 衰减方向	enum	r/w	#0 'Downward', #1 'Upward'	#0 'Downward'
18	TDM2 衰减分频器	int	r/w	min=0, max=4095	0
19	TDM2 衰减初始值	int	r/w	min=0, max=262143	262143
20	TDM2 衰减步进因子	int	r/w	min=0, max=262143	174762
21	TDM2 衰减步降因子	int	r/w	min=0, max=262143	196608

下页继续.....

表 84. 混音器控件（续表）

ID	名称	类型	接入方式	值	默认情况
22	TDM2 衰减阶跃目标	int	r/w	min=0, max=262143	16

7.5.4.2 源代码结构

下表列出了驱动程序的源文件。

表 85. 混音器驱动程序文件

文件	说明
sound/soc/fsl/fsl_amix.h	包括具有共用定义的文件
sound/soc/fsl/fsl_amix.c	混音器 DAI 驱动程序
sound/soc/fsl/imx-amix.c	混音器机器驱动程序
Documentation/devicetree/bindings/sound/fsl,amix.txt	混音器设备树绑定文档

7.5.4.3 菜单配置选项

本模块提供了以下 Linux 内核配置：

- CONFIG_SND_IMX_AMIX - Configuration option for the Audio Mixer Driver
- Device Drivers -> Sound card support -> Advanced Linux Sound Architecture -> ALSA for SoC audio support -> SoC Audio for Freescale i.MX CPUs -> SoC Audio support for i.MX boards with AMIX

7.6 PDM 麦克风接口（MICFIL）

7.6.1 简介

在移动电话等多个应用中，PDM 是将音频从麦克风传输到处理器的一种流行方式。然而，当前的数字音频系统使用多位音频信号（也称为多位 PCM）来表示信号。为此，通常在 DSP 或软件上实施一组 FIR、CIC 或/和半带滤波器。该模块实现所需的数字接口，以可配置的输出采样率提供 PDM 麦克风数位流的 16 位音频信号。

7.6.2 框图

下图从高层级显示了 PDM 麦克风接口模块。

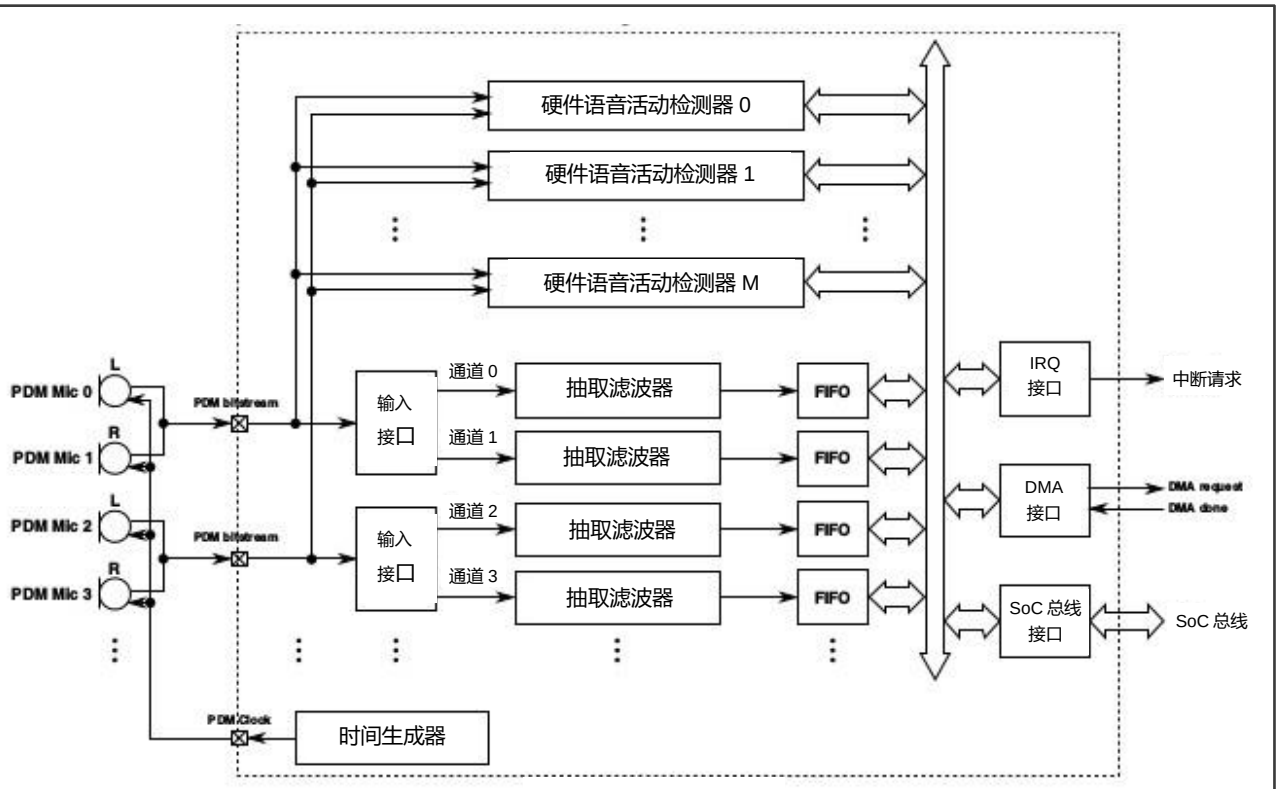


图 34. PDM 麦克风接口块

7.6.3 硬件概述

该模块的实现是基于数字信号处理技术在硬件上的应用。PDM 麦克风接口架构旨在节省门限和最小化功耗。它实现了一系列滤波器，将 1 位的 PDM 位流转换为音频频带中的 16 位 PCM 信号。

为了避免在通带内混叠频率，整个滤波器的阻带衰减为 80 dB，通带纹波小于 0.2 dB。整个模块实现多通道工作模式。所有通道都具有相同的配置，但每个输入通道都可以单独打开/关闭。

PDM 麦克风接口模块由以下部分组成：

- 每对 PDM 麦克风均有输入接口
- 按通道设置的抽取过滤器
- 每个通道设有 FIFO
- 时间生成单元
- DMA、中断和 SoC 的共享接口
- 一个或多个硬件语音活动检测器（HWVAD）

抽取滤波器实现了音频频段的低通滤波（20 Hz-22.5 KHz，默认情况下输出采样率为 48 kHz），其抽取率可配置。它使用 CIC、半带、FIR 和 DC remover 滤波器来实现。

时间生成器单元为麦克风生成 PDM 时钟。此时钟对所有 PDM 麦克风都是相同的，并且对所有麦克风都是激活的，也就是说，不可能只为一个麦克风关闭该时钟。它还为所有滤波器块生成定时信号和控制。滤波器中的抽取也由该模块控制。它激活每个块和通道，并将启动信号提供给 FIR FSM 和半带 FSM。

最后，每个抽取滤波器的输出存储在 FIFO 缓冲区中。每个 FIFO 都映射到 DATACHn 寄存器中。在所有启用通道的每个 FIFO 中，存储的数据数量超过配置的限度时，可以生成中断或 DMA 请求。

7.6.4 软件概述

PDM 麦克风驱动程序是在 ALSA 片内系统（ASOC）层下设计的。PDM 麦克风的 ASoC 驱动程序提供了一个采集设备来采集 PDM 麦克风输出。采集的音频格式为 8 声道 32 位宽的线性 PCM 音频数据，其频率为 48 kHz 或 44.1 kHz。

7.6.4.1 用户界面

使用 `amixer -c <pdm mic card>`工具，可从用户空间访问 PDM 麦克风界面。下表列出了这些控件。

表 86. PDM 麦克风控件

ID	名称	类型	接入方式	值	默认情况
1	CH0 增益	int	r/w	min=0, max=15	15
2	CH1 增益	int	r/w	min=0, max=15	15
3	CH2 增益	int	r/w	min=0, max=15	15
4	CH3 增益	int	r/w	min=0, max=15	15
5	CH4 增益	int	r/w	min=0, max=15	15
6	CH5 增益	int	r/w	min=0, max=15	15
7	CH6 增益	int	r/w	min=0, max=15	15
8	CH7 增益	int	r/w	min=0, max=15	15
9	MICFIL 质量选择	enum	r/w	#0 'Medium', #1 'High', #2 'N/A', #3 'N/A', #4 'VLow2', #5 'VLow1', #6 'VLow0', #7 'Low'	#0 'Medium'
10	HWVAD 初始化模式	enum	r/w	#0 'Envelope mode', #1 'Energy mode'	#0 'Envelope mode'
11	HWVAD 高通滤波器	enum	r/w	#0 'Filter bypass', #1 'Cut-off @1750Hz', #2 'Cut-off @215Hz', #3 'Cut-off @102Hz'	#0 'Filter bypass'
12	HWVAD 过零检测器使能	enum	r/w	#0 'OFF', #1 'ON'	#0 'OFF'
13	HWVAD 过零检测器自动阈值	enum	r/w	#0 'OFF', #1 'ON'	#0 'OFF'
14	HWVAD 噪声或使能	enum	r/w	#0 'Disabled', #1 'Enabled'	#0 'Disabled'

下页继续.....

表 86. PDM 麦克风控件（续）

ID	名称	类型	接入方式	值	默认情况
15	HWVAD 采样率	enum	r/w	#0 '48KHz', #1 '44.1KHz'	#0 '48KHz'
16	时钟源	enum	r/w	#0 'Auto', #1 'AudioPLL1', #2 'AudioPLL2', #3 'ExtClk3'	#0 'Auto'
17	HWVAD 输入增益	int	r/w	min=0, max=15	0
18	HWVAD 声音增益	int	r/w	min=0, max=15	0
19	HWVAD 噪声增益	int	r/w	min=0, max=15	0
20	HWVAD 检测器帧时间	int	r/w	min=1, max=64	1
21	HWVAD 检测器初始化时间	int	r/w	min=1, max=32	1
22	HWVAD 噪声滤波器调整	int	r/w	min=1, max=32	1
23	HWVAD 过零检测器阈值	int	r/w	min=1, max=1024	1
24	HWVAD 过零检测器调整	int	r/w	min=1, max=16	1

7.6.4.2 源代码结构

下表列出了驱动程序的源文件。

表 87. 混音器驱动程序文件

文件	说明
sound/soc/fsl/fsl_micfil.h	包括具有共用定义的文件
sound/soc/fsl/fsl_micfil.c	PDM 麦克风 DAI 驱动程序
sound/soc/fsl/imx-micfil.c	PDM 麦克风机器驱动程序
Documentation/devicetree/bindings/sound/fsl,micfil.txt	PDM 麦克风设备树绑定文档

7.6.4.3 菜单配置选项

本模块提供了以下 Linux 内核配置：

- CONFIG_SND_IMX_MICFIL - Configuration option for PDM Microphone Driver
- Device Drivers -> Sound card support -> Advanced Linux Sound Architecture > ALSA for SoC audio support -> SoC Audio for Freescale i.MX CPUs -> SoC Audio support for i.MX boards with micfil

第 8 章
安全性

8.1 加密加速和保证模块（CAAM）

8.1.1 CAAM 设备驱动程序概述

本节讨论如何在 Linux 内核中实施支持 CAAM（加密加速和保证模块）的内核驱动程序组件。

CAAM 基本驱动程序包可分为两个不同的级别：

- 配置和作业执行级别
- API 接口级别

配置和作业执行级别包括：

- 控制和配置模块，可映射主寄存器页并写入全局或系统所需的配置信息。
- 模块通过作业环反馈作业并报告状态。

API 接口级别包括：

- Sctterlist Crypto API 的接口，支持异步单通身份验证-加密操作和常见的块密码 - `caamalg`。
- Scatterlist Crypto API 的接口，支持异步散列 - `caamhash`。
- hwrng API 的接口，支持使用随机数生成器 - `caamrng`。

8.1.2 配置和作业执行级别

本节分为两个部分：

- 控制/配置驱动程序
- 作业环驱动程序

8.1.3 控制/配置驱动程序

控制和配置驱动程序负责初始化和设置主寄存器页、初始化早期特征初始化、提供有限的调试和监控能力，以及通常确保所有其他从属驱动程序子系统可以连接到正确配置的设备。

在启动时，它会逐步执行以下操作：

- 为此级别分配专用存储块。
- 将虚拟地址映射到整个 CAAM 寄存器页面。
- 为 SNVS 寄存器页面映射虚拟地址。
- 为安全存储器映射虚拟（高速缓存一致性）地址。
- 登记安全违规中断。
- 为平台选择正确的 DMA 地址大小，并设置匹配的 DMA 地址掩码。
- 识别其他相关的中断连接。
- 初始化所有作业环实例。

- 如果系统配置包括 DPAA 队列接口，则该接口启用了帧弹出功能。

注意

i.MX 6 配置不包含此逻辑。

- 如果实例包含 TRNG，则设置其振荡器/熵配置再启动。
- 将配置信息发送到系统控制台，会显示驱动程序处于活动状态，以及它采用的配置。
- 如果在内核配置中选择 CONFIG_DEBUG_FS，则添加了条目以在性能监视器中启用有用寄存器的 debugfs 视图。通过 debugfs 根目录的 caam/ctl 目录可以访问寄存器视图。

8.1.4 作业环驱动程序

作业环驱动程序负责为更高级别的驱动程序提供作业执行服务。它负责输入和输出环的整体管理以及驱动输出环的中断服务。

驱动程序调用可用于更高层，将作业排队到环中执行：

```
int caam_jr_enqueue(struct device *dev, u32 *desc, void (*cbk)(struct device
*dev, u32 *desc, u32 status, void *areq), void *areq);
```

参数：

`dev` 指针：指向与要使用的作业环关联的结构体。在当前配置中，控制器的专用数据块存在一个或多个结构体条目，每个作业环对应一个条目。

`desc` 指针：指向要执行的 CAAM 作业描述符。驱动程序将在执行之前映射描述符，并在完成后取消映射。然而，由于驱动程序无法合理地了解描述符引用的数据，因此调用者有责任在提交之前映射/刷新这些数据，并在请求完成后取消映射/使数据失效。

`cbk` 指针：指向作业处理完成后将调用的回调函数。

`areq` 指针：指向与此请求相关联的元数据或上下文数据。通常，它可以包含引用的数据映射信息，请求后处理（通过回调）可以在完成后使用这些信息来清理或释放资源。

回调函数参数：

`dev` 指针：指向与要使用的作业环关联的结构体。

`desc` 指针：指向提交执行的原始描述符。

`status`：指从执行请求的 CAAM DECO 收到的完成状态。仅当发生错误时才为非零。error.c 中列举了描述每个错误的字符串。

`areq`：元数据/上下文指针，传递给原始请求。

返回：

- 如果成功提交作业，则为零
- 如果输入环已满，则为-EBUSY
- 如果驱动程序无法映射作业描述符，则为-EIO

8.1.5 API 接口级别

CAAM 模块通过 Scatterlist Crypto API 为常见对称分组密码和单通道身份验证加密服务提供连接。下表按通用名称、驱动程序名称和用途列出了所有已安装的身份验证加密算法。请注意，某些平台（如 i.MX 6）包含的低功耗 MDHA 加速器不支持 SHA384 或 SHA512。

名称	驱动程序名称	用途
authenc(hmac(md5),cbc(aes))	authenc-hmac-md5-cbc-aes-caam	使用 MD5 和 AES-CBC 进行单通道身份验证/加密
authenc(hmac(sha1),cbc(aes))	authenc-hmac-sha1-cbc-aes-caam	使用 SHA1 和 AES-CBC 进行单通道身份验证/加密
authenc(hmac(sha224),cbc(aes))	authenc-hmac-sha224-cbc-aes-caam	使用 SHA224 和 AES-CBC 进行单通道身份验证/加密
authenc(hmac(sha256),cbc(aes))	authenc-hmac-sha256-cbc-aes-caam	使用 SHA256 和 AES-CBC 进行单通道身份验证/加密
authenc(hmac(sha384),cbc(aes))	authenc-hmac-sha384-cbc-aes-caam	使用 SHA384 和 AES-CBC 进行单通道身份验证/加密
authenc(hmac(sha512),cbc(aes))	authenc-hmac-sha512-cbc-aes-caam	使用 SHA512 和 AES-CBC 进行单通道身份验证/加密
authenc(hmac(md5),cbc(des3_ede))	authenc-hmac-md5-cbcdes3_ede-caam	使用 MD5 和 Triple-DES-CBC 进行单通道身份验证/加密
authenc(hmac(sha1),cbc(des3_ede))	authenc-hmac-sha1-cbc-des3_ede-caam	使用 SHA1 和 Triple-DES-CBC 进行单通道身份验证/加密
authenc(hmac(sha224),cbc(des3_ede))	authenc-hmac-sha224-cbc-des3_ede-caam	使用 SHA224 和 Triple-DES-CBC 进行单通道身份验证/加密
authenc(hmac(sha256),cbc(des3_ede))	authenc-hmac-sha256-cbc-des3_ede-caam	使用 SHA256 和 Triple-DES-CBC 进行单通道身份验证/加密
authenc(hmac(sha384),cbc(des3_ede))	authenc-hmac-sha384-cbc-des3_ede-caam	使用 SHA384 和 Triple-DES-CBC 进行单通道身份验证/加密
authenc(hmac(sha512),cbc(des3_ede))	authenc-hmac-sha512-cbc-des3_ede-caam	使用 SHA512 和 Triple-DES-CBC 进行单通道身份验证/加密
authenc(hmac(md5),cbc(des))	authenc-hmac-md5-cbc-des-caam	使用 MD5 和 Single-DES-CBC 进行单通道身份验证/加密
authenc(hmac(sha1),cbc(des))	authenc-hmac-sha1-cbc-des-caam	使用 SHA1 和 Single-DES-CBC 进行单通道身份验证/加密
authenc(hmac(sha224),cbc(des))	authenc-hmac-sha224-cbc-des-caam	使用 SHA224 和 Single-DES-CBC 进行单通道身份验证/加密
authenc(hmac(sha256),cbc(des))	authenc-hmac-sha256-cbc-des-caam	使用 SHA256 和 Single-DES-CBC 进行单通道身份验证/加密
authenc(hmac(sha384),cbc(des))	authenc-hmac-sha384-cbc-des-caam	使用 SHA384 和 Single-DES-CBC 进行单通道身份验证/加密
authenc(hmac(sha512),cbc(des))	authenc-hmac-sha512-cbc-des-caam	使用 SHA512 和 Single-DES-CBC 进行单通道身份验证/加密

此表按通用名称、驱动程序名称和用途列出了所有安装的对称密钥分组密码算法。

名称	驱动程序名称	用途
cbc(aes)	cbc-aes-caam	带有 CBC 模式包装器的 AES
cbc(des3_ede)	cbc-3des-caam	带有 CBC 模式包装器的三重 DES
cbc(des)	cbc-des-caam	带有 CBC 模式包装器的单个 DES
ecb(aes)	ecb-aes-caam	带有 ECB 模式包装器的 AES
ecb(des3_ede)	ecb-3des-caam	带有 ECB 模式包装器的三重 DES
ecb(des)	ecb-des-caam	带有 ECB 模式包装器的单个 DES
ecb(arc4)	ecb-arc4-caam	带有 ECB 模式包装器的 ARC4
ctr(aes)	ctr-aes-caam	带有 CTR 模式包装器的 AES

有关如何通过 API 使用这些服务，请参见内核加密子系统中的通用一致性/性能测试模块示例，该加密子系统为 tcrypt，在 crypto/tcrypt.c 的内核源代码树中可见。

`caamhashmodule` 通过 Scatterlist Crypto API 为常见的异步散列提供连接。

此表按通用名称、驱动程序名称和用途列出了所有安装的异步散列函数。请注意，某些平台（如 i.MX 6）包含的低功耗 MDHA 加速器不支持 SHA384 或 SHA512。

名称	驱动程序名称	用途
sha1	sha1-caam	SHA1-160 散列计算
sha224	sha224-caam	SHA224 散列计算
sha256	sha256-caam	SHA256 散列计算
sha384	sha384-caam	SHA384 散列计算
sha512	sha512-caam	SHA512 散列计算
md5	md5-caam	MD5 散列计算
hmac(sha1)	hmac-sha1-caam	SHA1-160 散列消息认证码
hmac(sha224)	hmac-sha224-caam	SHA224 散列消息认证码
hmac(sha256)	hmac-sha256-caam	SHA256 散列消息认证码
hmac(sha384)	hmac-sha384-caam	SHA384 散列消息认证码
hmac(sha512)	hmac-sha512-caam	SHA512 散列消息认证码
hmac(md5)	hmac-md5-caam	MD5 散列消息认证码

有关如何通过 API 使用这些服务，请参见内核加密子系统中的通用一致性/性能测试模块示例，该加密子系统为 tcrypt，在 crypto/tcrypt.c 的内核源代码树中可见。

`caamrng` 模块安装了一种机制，即使用 CAAM 的随机数生成器将随机数据馈送到一对缓冲器中，这些缓冲器可以通过 `/dev/random` 访问。

/dev/random通常用于提供内核自己的熵池，可在内部用作其他随机数据“设备”的熵源。

有关支持此服务的更多信息，请参见 sourceforge.net/projects/gkernel/files/rng- tools 中提供的 rng-tools。

8.1.6 驱动程序配置

驱动程序的配置由以下内核配置参数控制（位于 Cryptographic API -> Hardware Crypto Devices 下）：

```
CRYPTO_DEV_FSL_CAAM
```

启用构建基本控制器驱动程序和作业环后端。

```
CRYPTO_DEV_FSL_CAAM_RINGSIZE
```

选择作业环的大小（例如，最大作业条目数）。在 2-9 的范围内可以选择 2 的幂，允许选择作业环深度（4 到 512 个作业条目）。默认选择为 9，因此作业环深度为 512 个作业条目。

```
CRYPTO_DEV_FSL_CAAM_INTC
```

启用硬件的中断合并功能，这可以减少系统在高利用率期间产生的中断开销。禁用此选项会导致每次作业完成时需强制执行一次中断，从而简化操作，但会增加开销。

```
CRYPTO_DEV_FSL_CAAM_INTC_COUNT_THLD
```

如果启用了中断合并功能，则选择引发中断之前允许排队的作业数，范围在 1 到 255 之间。选择 1 实际上会使合并功能失效。任何大于作业环大小的选项都会先导致中断超时，然后才会引发中断。

默认选择是 255。

```
CRYPTO_DEV_FSL_CAAM_INTC_TIME_THLD
```

如果启用了中断合并功能，则在中断合并超时之前选择总线时钟计数（除以 64），如果未达到计数阈值，则在时段结束时引发中断，选择 1 至 65535 的整数。

默认选择是 2048。

```
CRYPTO_DEV_FSL_CAAM_CRYPTO_API
```

启用 Scatterlist Crypto API，通过 API 支持异步分组密码和单通道身份验证加密操作，使用 CAAM 硬件加速。

```
CRYPTO_DEV_FSL_CAAM_AHASH_API
```

启用 Scatterlist Crypto API 通过 API 支持异步散列，使用 CAAM 硬件加速。

```
CRYPTO_DEV_FSL_CAAM_RNG_API
```

通过 hwrng API 启用 CAAM 随机数生成器，可用于生成随机数据，为内核伪随机数生成器提供熵池。

```
CRYPTO_DEV_FSL_CAAM_RNG_TEST
```

启用强制测试，确保 CAAM RNG 驱动程序正在运行并缓冲随机数据。

8.1.7 局限性

- 驱动程序的组件目前不作为模块构建和运行。这可能会在未来的版本中得到纠正。
- 控制器和作业环后端之间存在相互依赖关系，因此它们都必须在相同的系统分区中运行。未来版本的驱动程序可能会将作业环后端分离为独立模块，让其可以在自己的系统分区中独立运行（并支持独立的 API 和 SM 实例）。
- 整个 CAAM 寄存器页面通过控制器驱动程序映射，指向选定子系统的派生指针被计算并传递给更高层的驱动程序组件。独立于分区的配置必须映射自己的子系统指针。
- 此驱动程序的上游变体仅支持 Power 架构。这种 Arm 架构特定的端口此时不会上行，尽管部分端口可能会在某个时间点向上传输。
- 未来的版本可能需要将 TRNG kickstart 迁移到启动加载程序中，以便更早地使用 RNG。
- Job Ring 驱动程序具有当前不需要的注册和注消功能（在未来版本中可能会被重写），可向更高层提供关闭通知。
- 完整的 CAAM 功能与 DSM 中的 Mega/Fast mix off（快速混合开关）功能不能同时开启。如果启用了 CAAM，则需要禁用 Mega/Fast mix off 功能，用户应在内核启动后进行“echo enabled>/sys/bus/platform/devices/2100000.aips-bus/2100000.CAAM/2101000.jr0/power/wakeup”操作，然后 Mega/Fast mix 将在 DSM 中保持上电。

8.1.8 现有实施局限性的概述

本章介绍了密钥库管理接口的原型，旨在提供对 CAAM 安全存储器的访问。

安全存储器提供了受控的访问保护区域，可以在该区域运行的系统中存储和处理关键的系统安全参数，而不会暴露总线级的明文密码。密码可以导入到安全存储器中，也可以从安全存储器导出，但绝不能以明文形式从安全存储器导出。但密码可能会以一种隐性形式从安全存储器导出，即使用外部永远看不到的密钥。

该驱动程序通过其内核级 API 开放了一个基本接口，支持内核级服务访问安全存储器功能。它分为两部分：

- 密钥库初始化和维护接口
- 密钥库访问接口

初始化和维护服务用来初始化和定义密钥库接口的实例。同样，访问接口支持内核级服务使用 API 来管理安全参数。

8.1.9 初始化密钥库管理接口

安装一组函数指针，实施连接密钥库子系统的底层物理接口。

在当前版本中，一套默认的（和隐藏的）函数实施这个接口。未来实施此 API 时可能会安装替代接口。如果出现这种情况，可以提供此调用的替代方案。

```
void sm_init_keystore(struct device *dev);
```

参数：

dev：指向为管理安全存储器子系统的资源而建立的结构体。

8.1.10 检测可用的安全存储器存储单元

返回此驱动程序的本地实例可以访问的可用单元（“页面”）的数量，可用作资源探测器。

```
u32 sm_detect_keystore_units(struct device *dev);
```

参数：

dev：指向为管理安全存储器子系统的资源而建立的结构体。

返回：检测到的可供使用的单位数量（从 0 到 n-1），可用于对所有其他 API 函数的后续调用。

8.1.11 在检测到的单元中建立密钥库

在检测到的单元中设置分配表，可用于存储密钥（或其他密码）。该单元将被分成一系列固定大小的插槽，每个插槽都在分配表中标记为可用。每个插槽的大小都是构建时可选择的参数。

在调用 sm_establish_keystore()之前，不能调用密钥库访问接口。

sm_establish_keystore()应该在调用 sm_detect_keystore_units()之后执行。

```
int sm_establish_keystore(struct device *dev, u32 unit);
```

参数：

dev：指向为管理安全存储器子系统的资源而建立的结构体。

unit：通过调用 sm_detect_keystore_units()检测到的一个单元。

返回：

- 如果成功返回，则为零
- 如果密钥库子系统未初始化，则为-EINVAL
- 如果分配表和相关的上下文数据没有存储器可用，则为-ENOSPC。

8.1.12 释放密钥库

释放此密钥库单元使用的所有资源。无法进一步调用密钥库访问接口。

```
void sm_release_keystore(struct device *dev, u32 unit);
```

参数：

dev：指向为管理安全存储器子系统的资源而建立的结构体。

Unit：通过调用 sm_detect_keystore_units()检测到的一个单元。

8.1.13 从密钥库中分配插槽

从密钥库中分配插槽，通过密钥库访问接口在所有其他后续操作中使用。

```
int sm_keystore_slot_alloc(struct device *dev, u32 unit, u32 size, u32*slot);
```

参数：

dev：指向为管理安全存储器子系统的资源而建立的结构体。

unit：通过调用 sm_detect_keystore_units()检测到的一个单元。

size：要存储在分配的插槽中的所需数据大小。

slot：指向变量的指针，用于接收已知的分配插槽号。

返回：

- 如果成功完成，则为零。
- 如果请求的大小超过选定的插槽大小，则执行-EKEYREJECTED。

8.1.14 将数据加载到密钥库插槽

将数据加载到分配的密钥库槽中，以便在其上执行其他操作（如封装）。

```
int sm_keystore_slot_load(struct device *dev, u32 unit, u32 slot, constu8 *key_data, u32 key_length);
```

参数：

dev：指向为管理安全存储器子系统的资源而建立的结构体。

unit：通过调用 sm_detect_keystore_units()检测到的一个单元。

key_length：要写入插槽的信息的长度（字节）。

key_data 指针：指向要加载数据的缓冲区，而且必须是连续缓冲区。

返回：

- 如果成功完成，则为零。
- 如果请求的大小超过插槽可以容纳的大小，则执行-EFBIG。

8.1.15 更新演示图像

将写入密钥库插槽的数据封装为安全存储器 Blob。

```
int sm_keystore_slot_encapsulate(struct device *dev, u32 unit, u32
inslot, u32 outslot, u16 secretlen, u8 *keymod, u16 keymodlen);
```

参数：

dev：指向为管理安全存储器子系统的资源而建立的结构体。

unit：通过调用 sm_detect_keystore_units()检测到的一个单元。

inslot：保存输入密码的插槽，由 sm_keystore_slot_load()加载到该插槽中。请注意，应尽快覆盖或释放包含此密码的插槽，因为此时它包含明文。

outslot：已分配的插槽，将封装的输出保存为安全存储器 Blob。

secretlen：要封装的密码长度，不包括任何 Blob 存储开销（Blob 密钥、MAC 等）。

keymod：用于封装的密钥修改组件。密钥修饰符允许在封装过程中使用额外的密码。解封装也需要相同的修饰符。

keymodlen：密钥修饰符的长度（字节）。

返回：

- 零表示成功
- 发生故障时，则显示 CAAM 作业状态

8.1.16 解封密钥库中的数据

将密钥库中的数据解封装为黑密钥 Blob，以便在其他加密操作中使用。黑密钥 Blob 支持在主存储器中使用密钥，而不会将其作为明文公开。

```
int sm_keystore_slot_decapsulate(struct device *dev, u32 unit, u32
inslot, u32 outslot, u16 secretlen, u8 *keymod, u16 keymodlen);
```

参数：

dev：指向为管理安全存储器子系统的资源而建立的结构体。

unit：通过调用 sm_detect_keystore_units()检测到的一个单元。

inslot：保存输入数据的插槽，由先前调用 sm_keystore_slot_encapsulate()处理，并包含安全存储器 Blob。

outslot：已分配的插槽，将封装的输出数据保存为黑密钥 Blob。

secretlen：要封装的密码长度，不包括任何 Blob 存储开销。

keymod：封装时指定的密钥修改组件。

keymodlen：密钥修饰符的长度（字节）。

返回：

- 零表示成功
- 发生故障时，则显示 CAAM 作业状态

8.1.17 从密钥库插槽读取数据

从密钥库插槽提取数据，存回用户缓冲区。通常在进行其他操作（例如，解封）后使用。

```
int sm_keystore_slot_read(struct device *dev, u32 unit, u32 slot, u32
key_length, u8 *key_data);
```

参数：

dev：指向为管理安全存储器子系统的资源而建立的结构体。

unit：通过调用 sm_detect_keystore_units()检测到的一个单元。

Slot：可读取数据的已分配插槽。

key_length：要从插槽中读取的信息长度（字节）。

key_data 指针：指向保存提取数据的缓冲区，而且必须是连续缓冲区。

返回：

- 如果成功完成，则为零。
- 如果请求的大小超过插槽可以容纳的大小，则执行-EFBIG。

8.1.18 将插槽释放回密钥库

将密钥库插槽释放回可用池。在取消分配之前，存储的信息会被清除。

```
int sm_keystore_slot_dealloc(struct device *dev, u32 unit, u32 slot);
```

参数：

`dev`：指向为管理安全存储器子系统的资源而建立的结构体。

`unit`：通过调用 sm_detect_keystore_units()检测到的一个单元。

`slot`：要释放回存储区的已分配插槽的编号。

返回：

- 如果成功完成，则为零。
- 如果指定了未分配的插槽，则为-EINVAL

安全存储器驱动程序/密钥库 API 的配置取决于以下内核配置参数：

```
CRYPTO_DEV_FSL_CAAM_SM
```

在内核版本中打开安全存储器驱动程序。

```
CRYPTO_DEV_FSL_CAAM_SM_SLOTSIZE
```

配置安全存储器“插槽”的大小。

每个安全存储单元是内部存储块，具体大小取决于实施情况。该存储块可细分为若干个大小可通过该值选择的逻辑“插槽”。这些插槽需要能够容纳预期的最大密码，还要考虑 Blob 参数（Blob 密钥和 MAC，通常不超过 48 个字节）的开销。

这些值可以选择为 2 的幂，范围在 32 到 512 字节之间。默认值为 7，大小为 128 字节。

```
CRYPTO_DEV_FSL_CAAM_SM_TEST
```

启用静态试验/示例模块的操作，该模块显示如何使用 API，同时验证其功能。测试模块按照以下流程工作：

- 创建许多已知的明文密码（3 种尺寸）
- 分配了安全存储器插槽
- 将这些密钥插入安全存储器插槽并封装
- 将这些密钥解封为黑密钥
- 使用黑密钥加密 DES、AES128 和 AES256 明文。因为它使用对称密码，所以相同的密钥加密/解密结果将是等效的。
- 使用等效的明文密钥对加密缓冲区进行解密。
- 将解密结果与原始密文进行比较。如果匹配，则此测试报告每个测试的密钥用例都是正常的。

控制台上报告的正常输出如下：

```
platform caam_sm.0: caam_sm_test: 8-byte key test match OK platform
```

```
caam_sm.0: caam_sm_test: 16-byte key test match OK platform caam_sm.0:
caam_sm_test: 32-byte key test match OK
```

- 安全存储器驱动程序目前未作为内核模块实施。
- 目前实施仅限于内核模式操作。
- 目前可能出现一种情况。将来，当作业环可以在不同的系统分区中独立运行时，应该考虑使用多实例安全存储器驱动程序。
- 所有存储请求仅限于单个插槽的存储大小（构建时可配置长度）。只要可以连续分配多个插槽，就允许密码跨越多个插槽。
- 所有页面/分区的插槽大小都是固定的。
- 封装/解封接口可以指定身份验证；底层接口不请求身份验证。
- 封装/解封接口返回作业状态；此状态应从 `errno.h` 转换为有意义的错误消息。

8.1.19 CAAM/SNV——安全违规处理接口概述

本章介绍了 SNVS 安全违规的驱动程序组件和控制接口的原型。它提供了一种安装、管理和执行应用定义的处理程序的方法，该处理程序可处理安全违规事件，以响应系统中发生的事情。

SNVS 可持续监控运行的系统中许多潜在的攻击向量。如果感测到其中一个附加向量发生的事情（例如，已检测到安全违规），SNVS 可以同时擦除关键的安全参数并转换到故障状态。生成一个中断则表示违规已发生。这个中断可以分派应用定义的例程，在违规后执行清理操作，可能导致安全服务的有序关闭。

因此，此接口的目的是允许系统级服务为这些类型的事件安装处理程序。通过此接口，系统设计人员能够选择如何应对特定的安全违规原因，使用为其系统特定要求编写的简单函数调用实现。

8.1.20 操作

对于现有平台，SNVS 中可能有 6 种安全违规中断原因，这些违规原因中的 5 个通常和使用有关，这些原因被定义为：

- SECVIO_CAUSE_CAAM_VIOLATION - CAAM/SNVS 内部检测到的违规
- SECVIO_CAUSE JTAG_ALARM – 检测到 JTAG 操作
- SECVIO_CAUSE_WATCHDOG – 看门狗过期
- SECVIO_CAUSE_EXTERNAL_BOOT - 外部启动加载操作
- SECVIO_CAUSE_TAMPER_DETECT - 触发篡改检测逻辑

每个原因都可以通过此 API 配套的驱动程序与应用定义的处理程序相关联。如果未指定任何处理程序，则将调用默认处理程序。这个处理程序只不过是识别系统控制台的中断原因。

8.1.21 配置接口

以下接口可用于从驱动程序的调度表中定义或删除应用定义的违规处理程序。

8.1.22 安装处理程序

```
int caam_secvio_install_handler(struct device *dev, enum secvio_cause
cause, void (*handler)(struct device *dev, u32 cause, void *ext), u8
*cause_description, void *ext);
```

参数：

Dev：指向拥有 SNVS 的设备。

Cause：上述列举的原因列表中的中断源原因。

handler：应用定义的处理程序，使用 dev、源原因和本地定义的处理程序参数进行调用。

cause_description：指向一个字符串来覆盖默认的原因名称，它可以用作错误消息等的替代项。如果保留为空，则使用默认描述字符串。ext 指针，指向处理程序需要的任何额外数据。

返回：

- 零表示成功。
- 如果参数无效或无法使用，则为-EINVAL。

8.1.23 删除已安装的驱动程序

```
int caam_secvio_remove_handler(struct device *dev, enum secvio_cause
cause);
```

参数：

Dev：指向拥有 SNVS 的设备。

Cause：中断源原因。

返回：

- 零表示成功。
- 如果参数无效或无法使用，则为-EINVAL。

8.1.24 驱动程序配置 CAAM/SNVS

```
CRYPTO_DEV_FSL_CAAM_SECVIO
```

支持将安全违规驱动程序和配置接口作为 build 配置的一部分。当前形式的驱动程序不能作为模块来构建。

8.2 显示内容完整性校验程序（DCIC）

8.2.1 简介

DCIC 的目标是验证发送到显示器的关键安全信息是否未损坏。

DCIC 具有以下功能：

- 像素时钟高达 148.5 MHz
- 显示接口控制信号的可配置极性
- 24 位像素数据总线
- 多达 16 个矩形 ROI，位置和大小可配置
- 为每个 ROI 独立进行 CRC32 签名计算
- 外部控制器失配指示信号

8.2.2 源代码结构

表 88. DCIC 驱动程序文件

文件	说明
drivers/video/fbdev/mxc/mxc_dcic.c	DCIC 驱动程序源代码
include/uapi/linux/mxc_dcic.h	DCIC 用户空间标头

8.2.3 菜单配置选项

在菜单配置中启用以下模块：

Device Drivers -> Graphics support -> MXC DCIC

8.2.4 DTS 配置

```
dcic_id = <0>;  /* DCIC device index 0-dcic1, i-dcic2 */
dcic_mux = "dcic-lcdif1"; /* DCIC input select */
```

表 89. DCIC 输入选择

模块	i.MX 6SoloX	i.MX 6Dual/6Quad
DCIC1	dcic_lvds dcic_lcdif1	dcic-ipu0-di1 dcic-lvds0 dcic-lvds1 dcic-hdmi
DCIC2	dcic_lvds dcic_lcdif2	dcic-ipu0-di0/dcic-ipu1-di0 dcic-lvds0 dcic- lvds1 dcic-mipi_dpi

8.2.5 IOCTL 函数

DCIC 驱动程序支持以下 IOCTL：

- DCIC_IOC_CONFIG_DCIC：配置 DCIC 输入 CLK、VSYNC、HSYNC 和数据信号极性。
- DCIC_IOC_CONFIG_ROI：配置 ROI 块大小和参考签名。
- DCIC_IOC_GET_RESULT：获取 ROI 计算签名的结果。

8.2.6 结构

```
struct roi_params {
    unsigned int roi_n;      /* ROI index */
    unsigned int ref_sig;   /* Reference CRC32 */
    unsigned int start_y;   /* start vertical lines of ROI */
    unsigned int start_x;  /* start horizon lines of ROI */
    unsigned int end_y;   /* end vertical lines of ROI */
```

```
    unsigned int end_x;   /* end horizon lines of ROI */
    char freeze;                /*  state of ROI */
};
```

8.2.7 DCIC CRC 计算函数

本单元测试中用四个函数来计算参考签名：

```
crc32_calc_18of24bit()    /* CRC calculate 18 bit of 24  */
crc32_calc_24bit()            /* CRC calculate 24  */
crc32_calc_24of16bit()  /* CRC calculate 24 bit of 16  */
crc32_calc_18of16bit() /* CRC calculate 18 bit of 16  */
```

DCIC 根据显示总线宽度计算 CRC，但显示总线宽度并不总是与每像素字节数（bpp）对齐，上述四种函数可以覆盖不同的显示总线宽度和每像素字节数。

8.3 智能卡接口——用户识别模块（SIM）

8.3.1 简介

用户识别模块（SIM）可促进与 SIM 卡或 Eurochip 预付费电话卡的通信，并与 ISO/IEC 7816-3 标准兼容。SIM 模块有一个端口，可与各种卡对接。与微控制器单元（MCU）的接口采用 32 位连接，请参见参考文档《IP 总线规范》。

8.3.2 操作模式

SIM 模块 I/O 接口可采用以下三种操作模式之一进行操作。

- 双线接口：IC 引脚 RX 和 TX 均可连接智能卡。
- 外部单线接口：IC 引脚 RX 和 TX 从外部连接到 IC，并路由到智能卡。
- 内部单线接口：IC 引脚 TX 路由至智能卡。接收引脚 RX 与 IC 内部的 TX 引脚相连。

8.3.3 外部信号说明

- SIM_CLK：SIM 模块为智能卡提供的时钟。典型频率为 1MHz 至 5MHz。该时钟是 SIM_TRXD 引脚上数据速率的 372 倍。
- SIM_RST_B：从 SIM 卡到智能卡复位信号。
- SIM_SVEN：智能卡电源使能控制信号。
- SIM_TRXD：SIM 模块向智能卡发送/SIM 模块从智能卡接收的数据。
- SIM_PD：智能卡插入检测。

8.3.4 源代码结构

表 90. SIM 来源

文件	说明
drivers/mxc/sim/imx_sim.c	SIM 驱动程序
drivers/mxc/sim/imx_envsim.c	SIM Env

8.3.5 菜单配置选项

通过 menuconfig 配置内核选项来启用模块：

Device Drivers > MXC support drivers > MXC SIM Support

8.3.6 软件框架

下图显示了 SIM TX 和 RX 软件流程。

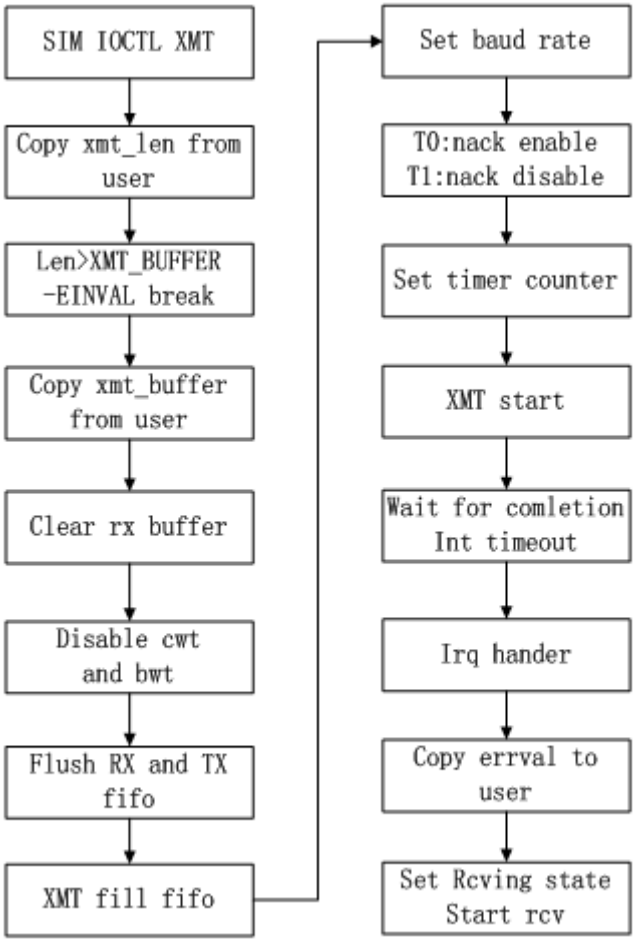


图 35. SIM TX（发射）流程

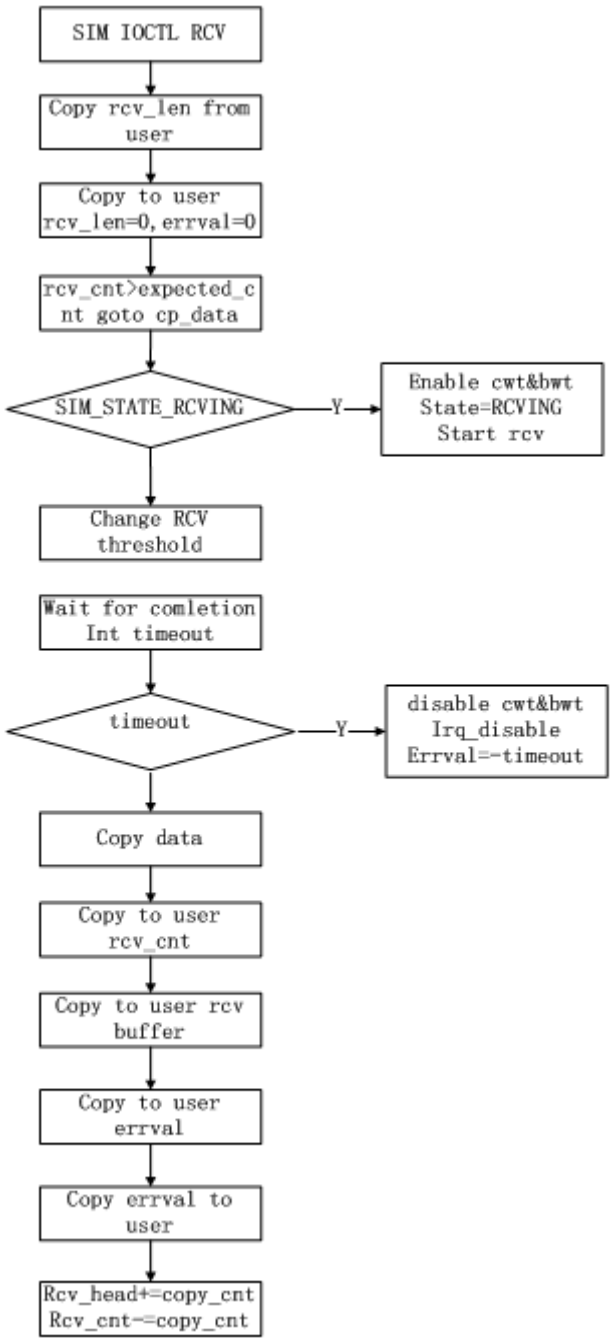


图 36. SIM RX（接收）流程

8.4 安全非易失性存储（SNVS）

8.4.1 简介

如需了解与安全非易失性存储（SNVS）相关的更多信息，请参阅相关 SoC 的《i.MX 安全手册》。

SNVS 是与 CAAM 和 SRTC 对接的模块。

如需了解与 CAAM 相关的 SNVS 服务内容，请参阅驱动程序配置 CAAM/SNVS 小节。

如需了解与 SRTC 相关的 SNVS 服务内容，请参阅 SRTC 简介小节

8.5 SNVS 实时时钟（SRTC）

8.5.1 简介

实时时钟（RTC）模块用于保存时间和日期。它为用户提供了可证明的时间，并且在检测到篡改计数器时会发出告警。

8.5.2 硬件操作

RTC 是由系统控制器固件提供的伪定时器，仅支持读取/设置时间、读取/设置闹钟等基本功能。

8.5.3 软件操作

下面章节介绍 RTC 驱动程序的软件操作。

8.5.4 驱动程序功能

RTC 驱动程序包括以下功能：

- 实施 Linux 操作系统所需的功能，提供实时时钟和闹钟中断功能
- 闹钟将系统从低功耗模式唤醒

8.5.5 源代码结构

RTC 模块在 drivers/rtc 中实施。

表 91. RTC 驱动程序文件

文件	说明
drivers/rtc/rtc-imxdi.c	MX6 RTC 驱动程序
drivers/rtc/rtc-imx-sc.c	MX8 RTC 系统控制器驱动程序
drivers/rtc/rtc-imx-rpmsg.c	RPMSG RTC 驱动程序

8.5.6 菜单配置选项

在菜单配置中启用以下模块：

For i.MX 6 select Device Drivers > Real Time Clock > Freescale IMX DryIce Real Time Clock

For i.MX 8 with SC select Device Drivers > Real Time Clock > NXP SC RTC support

For RPMSG select Device Drivers > Real Time Clock > NXP RPMSG RTC support

第 9 章
恩智浦 eIQ® 机器学习（ML）

9.1 恩智浦 eIQ 机器学习概述

9.1.1 机器学习简介

机器学习（ML）是起源于 20 世纪 60 年代的一个计算机科学领域。机器学习提供了能够在数据中发现模式和规则的算法。机器学习是一种算法，允许软件应用程序在无需显式编程的情况下更准确地预测结果。机器学习的基本前提是构建能够接收输入数据并使用统计分析来预测输出的算法，同时随着新数据的出现而更新输出。2010 年，一股称为深度学习的巨大热潮开始了，它是机器学习基于神经网络（NN）的一个快速增长的分支。深度学习受人脑启发，在各种任务（如计算机视觉（CV）、自然语言处理（NLP））中取得了重大的突破。神经网络能够从数百万个例子中学习复杂的模式。机器学习有望在嵌入式世界实现巨大的适配性，而这正是恩智浦领先的领域。恩智浦继续为客户提供支持，为 i.MX 打造了 eIQ 机器学习软件，这是一套机器学习工具，支持在 i.MX 8 QuadMax 设备上开发并部署 ML 应用程序。本章概述了恩智浦 eIQ 机器学习技术的具体服务领域。如需了解机器学习命令的执行细节，请参阅《*i.MX 机器学习用户指南*》（IMXMLUG）。

9.1.2 OpenCV

OpenCV 是开源计算机视觉库，其名为 ML 的模块提供传统的机器学习算法。OpenCV 的另一个重要模块是 DNN，为神经网络算法提供支持。

OpenCV 为神经网络推理（DNN 模块）和经典机器学习算法（ML 模块）提供了统一的解决方案。OpenCV 包含许多计算机视觉功能，可以更轻松地在短时间内构建复杂的机器学习应用程序，而且不用依赖其他库。

OpenCV 不仅在计算机视觉领域得到了大量应用，还得到了强大且非常活跃的社区支持。关键算法进行了专门优化，适用于各种设备和指令集。对于 i.MX，OpenCV 使用 Arm NEON 加速。Arm Neon 技术是面向 Arm Cortex-A 系列的高级 SIMD(单指令多数据)架构扩展。Neon 技术加速音视频编/解码、用户界面、2D/3D 图形或游戏来改善多媒体用户体验。Neon 还可以加速信号处理算法和功能，以加快音视频处理、语音和人脸识别、计算机视觉和深度学习等应用的速度。

OpenCV DNN 模块的核心是推理引擎，不提供任何用于神经网络训练的功能。 如需详细了解有关支持的型号和支持的层，请查看恩智浦 OpenCV 深度学习页面。

OpenCV ML 模块包含可解决机器学习问题的类和函数，例如分类、回归或聚类，涉及支持向量机（SVM）、决策树、随机树、期望最大化、k 近邻、经典贝叶斯分类器、逻辑回归和增强树等算法。

9.1.3 Arm 计算

Arm 计算库是面向 Arm CPU 和 GPU 架构而优化的底层函数集合，适用于图像处理、计算机视觉和机器学习。Arm 计算机是便捷的优化函数存储库，开发人员可以单独获取这些函数，也可以将其用作复杂流程的一部分来加速算法和应用。 Arm 计算库也支持 NEON 加速。Arm 计算机可以通过使用带随机权重和输入的 DNN 模型以及使用图形 API 的 AlexNet 进行实例展示。

9.1.4 TensorFlow Lite

TensorFlow Lite 是 TensorFlow 的轻量级版本，也是 TensorFlow 接下来要推出的产品。TensorFlow Lite 是一个开源软件库，专注于在移动嵌入式设备上运行机器学习模型（请参见 www.tensorflow.org/lite）。它支持设备端机器学习推理，具有低延迟和较小的二进制文件。TensorFlow Lite 还支持使用 Android OS 神经网络 API 进行硬件加速。

TensorFlow Lite 支持一组针对移动平台进行了调整的核心运算符，包括量化和浮点运算。它们结合预融合的激活和偏置来进一步提高性能和量化精度。此外，TensorFlow Lite 还支持在模型中使用自定义操作。

TensorFlow Lite 基于 FlatBuffers 定义了一种新的模型文件格式。FlatBuffers 是一个开源、高效的跨平台序列化库。它与协议缓冲区类似，但主要区别在于，FlatBuffers 在访问数据之前通常与每个对象被分配的内存相结合，不需要对二级表示进行解析/解包。此外，FlatBuffers 的代码占用空间比协议缓冲区小一个数量级。

TensorFlow Lite 拥有新的面向移动端优化的解析器，其关键目标是保持应用程序的精简和快速。解析器使用静态图形排序和自定义(少量动态)内存分配器来确保最小的负载、初始化和执行延迟。

9.1.5 Arm NN

Arm NN 是 Arm 开发的开源推理引擎框架，支持多种神经网络模型格式，例如：

- TensorFlow Lite
- ONNX

对于 i.MX 8，Arm NN 在带有 NEON 的 CPU 上运行，支持多核。如需了解 Arm NN 的更多信息，请查看 Arm NN SDK 官方网页。

9.1.6 ONNX Runtime

ONNX Runtime 是微软开发的一款开源推理引擎框架，支持 ONNX 模型格式。ONNX Runtime 在配备 NEON 的 CPU 上运行，它还支持使用执行提供程序的 GPU/NPU 硬件加速器。如需了解 ONNX Runtime 的更多信息，请查看 ONNX Runtime 项目官方网页。

9.1.7 PyTorch

PyTorch 是一个开源机器学习库，用于计算机视觉和自然语言处理等应用程序。它是一款免费的开源软件。PyTorch 为张量计算等高级功能提供了 Python 包。如需了解 PyTorch 的更多信息，请查看官方网页 www.pytorch.org。

9.1.8 DeepViewRT™

DeepViewRT 是一款针对恩智浦微处理器和微控制器优化的专有神经网络推理引擎，它不仅实施自己的计算引擎，还能够利用流行的第三方计算引擎。

9.1.9 TVM

TVM 是面向 CPU、GPU 和专用加速器的开放式深度学习编译器堆栈。它的目标是缩小着眼于生产力的深度学习框架与以性能或效率为导向的后端硬件的差距。

第 10 章
单元测试

10.1 系统

10.1.1 Oprofile Oprofile

10.1.1.1 测试名称

- autorun-oprofile.sh

10.1.1.1.1 位置

```
/unit_tests/OProfile/
```

10.1.1.1.2 功能

OProfile 是一个系统范围的分析器，能够以较低的开销分析所有运行的代码。OProfile 由一个内核驱动程序、一个收集样本数据的守护进程和几个将数据转换为信息的后期分析工具组成。

10.1.1.1.3 配置

无

10.1.1.1.4 用例和预期输出

```
./autorun-oprofile.sh
```

10.1.2 Owire

10.1.2.1 测试名称

- autorun-owire.sh

10.1.2.1.1 位置

```
/unit_tests/OWire/
```

10.1.2.1.2 功能

测试 EEPROM 功能。

10.1.2.1.3 配置

无

10.1.2.1.4 用例和预期输出

```
./autorun-owire.sh
```

10.1.3 电源管理

10.1.3.1 测试名称

- /unit_tests/Power_Management/suspend_random_auto.sh
- /unit_tests/Power_Management/suspend_quick_auto.sh

10.1.3.1.1 位置

```
/unit_tests/Power_Management/
```

10.1.3.1.2 功能

启用低功耗模式并唤醒所有 i.MX 板上的不同内核。

10.1.3.1.3 配置

无

10.1.3.1.4 用例和预期输出

```
$ /unit_tests/Power_Management/suspend_random_auto.sh
or
$ /unit_tests/Power_Management/suspend_quick_auto.sh
```

i.MX 7D Sabre SD 板的预期输出：

```
# /unit_tests/Power_Management/suspend_random_auto.sh
rtcwakeup.out: wakeup from "mem" using rtc0 at Wed Feb 22 22:55:29 2017
PM: Syncing filesystems ... done.
Freezing user space processes ... (elapsed 0.001 seconds) done.
Freezing remaining freezable tasks ... (elapsed 0.001 seconds) done.
Suspending console(s) (use no_console_suspend to debug)
PM: suspend of devices complete after 632.862 msecs
PM: suspend devices took 0.640 seconds
PM: late suspend of devices complete after 1.258 msecs
PM: noirq suspend of devices complete after 1.198 msecs
Disabling non-boot CPUs ...
CPU1: shutdown
Turn off Mega/Fast mix in DSM
Enabling non-boot CPUs ...
CPU1 is up
PM: noirq resume of devices complete after 0.832 msecs
imx-sdma 30bd0000.sdma: loaded firmware 4.2
PM: early resume of devices complete after 0.930 msecs
PM: resume of devices complete after 483.310 msecs
PM: resume devices took 0.480 seconds
Restarting tasks ... done.
============================
suspend 0 times
============================
wakeup 7 seconds, sleep 16 seconds
rtcwakeup.out: wakeup from "mem" using rtc0 at Wed Feb 22 22:55:42 2017
PM: Syncing filesystems ... done.
Freezing user space processes ... (elapsed 0.001 seconds) done.
Freezing remaining freezable tasks ... (elapsed 0.001 seconds) done.
Suspending console(s) (use no_console_suspend to debug)
```

```
PM: suspend of devices complete after 630.328 msecs
PM: suspend devices took 0.640 seconds
PM: late suspend of devices complete after 1.252 msecs
PM: noirq suspend of devices complete after 1.203 msecs
Disabling non-boot CPUs ...
CPU1: shutdown
Turn off Mega/Fast mix in DSM
Enabling non-boot CPUs ...
CPU1 is up
PM: noirq resume of devices complete after 0.777 msecs
imx-sdma 30bd0000.sdma: loaded firmware 4.2
PM: early resume of devices complete after 0.873 msecs
PM: resume of devices complete after 483.406 msecs
PM: resume devices took 0.480 seconds
Restarting tasks ... done.
============================
suspend 1 times
============================
wakeup 11 seconds, sleep 20 seconds
rtcwakeup.out: wakeup from "mem" using rtc0 at Wed Feb 22 22:56:10 2017
37PM: Syncing filesystems ... done.
Freezing user space processes ... (elapsed 0.001 seconds) done.
Freezing remaining freezable tasks ... (elapsed 0.001 seconds) done.
Suspending console(s) (use no_console_suspend to debug)
PM: suspend of devices complete after 651.761 msecs
PM: suspend devices took 0.660 seconds
PM: late suspend of devices complete after 1.245 msecs
PM: noirq suspend of devices complete after 1.193 msecs
Disabling non-boot CPUs ...
CPU1: shutdown
Turn off Mega/Fast mix in DSM
Enabling non-boot CPUs ...
CPU1 is up
PM: noirq resume of devices complete after 0.728 msecs
imx-sdma 30bd0000.sdma: loaded firmware 4.2
PM: early resume of devices complete after 0.859 msecs
PM: resume of devices complete after 483.441 msecs
PM: resume devices took 0.480 seconds
Restarting tasks ... done.
============================
suspend 2 times
============================
wakeup 3 seconds, sleep 12 seconds
rtcwakeup.out: wakeup from "mem" using rtc0 at Wed Feb 22 22:56:34 2017
PM: Syncing filesystems ... done.
Freezing user space processes ... (elapsed 0.001 seconds) done.
Freezing remaining freezable tasks ... (elapsed 0.001 seconds) done.
Suspending console(s) (use no_console_suspend to debug)
PM: suspend of devices complete after 641.321 msecs
PM: suspend devices took 0.650 seconds
PM: late suspend of devices complete after 1.258 msecs
PM: noirq suspend of devices complete after 1.195 msecs
Disabling non-boot CPUs ...
CPU1: shutdown
Turn off Mega/Fast mix in DSM
Enabling non-boot CPUs ...
CPU1 is up
PM: noirq resume of devices complete after 0.730 msecs
imx-sdma 30bd0000.sdma: loaded firmware 4.2
PM: early resume of devices complete after 0.857 msecs
```

```
PM: resume of devices complete after 483.451 msecs
PM: resume devices took 0.480 seconds
Restarting tasks ... done.
============================
suspend 3 times
============================
wakeup 9 seconds, sleep 8 seconds
rtcwakeup.out: wakeup from "mem" using rtc0 at Wed Feb 22 22:56:56 2017
PM: Syncing filesystems ... done.
Freezing user space processes ... (elapsed 0.001 seconds) done.
Freezing remaining freezable tasks ... (elapsed 0.001 seconds) done.
38Suspending console(s) (use no_console_suspend to debug)
PM: suspend of devices complete after 641.492 msecs
PM: suspend devices took 0.650 seconds
PM: late suspend of devices complete after 1.255 msecs
PM: noirq suspend of devices complete after 1.201 msecs
Disabling non-boot CPUs ...
CPU1: shutdown
Turn off Mega/Fast mix in DSM
Enabling non-boot CPUs ...
CPU1 is up
PM: noirq resume of devices complete after 0.731 msecs
imx-sdma 30bd0000.sdma: loaded firmware 4.2
PM: early resume of devices complete after 0.861 msecs
PM: resume of devices complete after 483.476 msecs
PM: resume devices took 0.480 seconds
Restarting tasks ... done.
^c
```

```
# /unit_tests/Power_Management/suspend_quick_auto.sh
rtcwakeup.out: wakeup from "mem" using rtc0 at Wed Feb 22 23:01:16 2017
PM: Syncing filesystems ... done.
Freezing user space processes ... (elapsed 0.001 seconds) done.
Freezing remaining freezable tasks ... (elapsed 0.001 seconds) done.
Suspending console(s) (use no_console_suspend to debug)
PM: suspend of devices complete after 632.891 msecs
PM: suspend devices took 0.640 seconds
PM: late suspend of devices complete after 1.254 msecs
PM: noirq suspend of devices complete after 1.200 msecs
Disabling non-boot CPUs ...
CPU1: shutdown
Turn off Mega/Fast mix in DSM
Enabling non-boot CPUs ...
CPU1 is up
PM: noirq resume of devices complete after 0.734 msecs
imx-sdma 30bd0000.sdma: loaded firmware 4.2
PM: early resume of devices complete after 0.862 msecs
PM: resume of devices complete after 483.417 msecs
PM: resume devices took 0.480 seconds
Restarting tasks ... done.
==============================
suspend 1 times
==============================
rtcwakeup.out: wakeup from "mem" using rtc0 at Wed Feb 22 23:01:19 2017
PM: Syncing filesystems ... done.
Freezing user space processes ... (elapsed 0.001 seconds) done.
Freezing remaining freezable tasks ... (elapsed 0.001 seconds) done.
Suspending console(s) (use no_console_suspend to debug)
```

```
PM: suspend of devices complete after 631.833 msecs
39PM: suspend devices took 0.640 seconds
PM: late suspend of devices complete after 1.253 msecs
PM: noirq suspend of devices complete after 1.242 msecs
Disabling non-boot CPUs ...
CPU1: shutdown
Turn off Mega/Fast mix in DSM
Enabling non-boot CPUs ...
CPU1 is up
PM: noirq resume of devices complete after 0.729 msecs
imx-sdma 30bd0000.sdma: loaded firmware 4.2
PM: early resume of devices complete after 0.862 msecs
PM: resume of devices complete after 483.416 msecs
PM: resume devices took 0.480 seconds
Restarting tasks ... done.
==============================
suspend 2 times
==============================
rtcwakeup.out: wakeup from "mem" using rtc0 at Wed Feb 22 23:01:22 2017
PM: Syncing filesystems ... done.
Freezing user space processes ... (elapsed 0.001 seconds) done.
Freezing remaining freezable tasks ... (elapsed 0.001 seconds) done.
Suspending console(s) (use no_console_suspend to debug)
PM: suspend of devices complete after 633.624 msecs
PM: suspend devices took 0.640 seconds
PM: late suspend of devices complete after 1.252 msecs
PM: noirq suspend of devices complete after 1.204 msecs
Disabling non-boot CPUs ...
CPU1: shutdown
Turn off Mega/Fast mix in DSM
Enabling non-boot CPUs ...
CPU1 is up
PM: noirq resume of devices complete after 0.733 msecs
imx-sdma 30bd0000.sdma: loaded firmware 4.2
PM: early resume of devices complete after 0.853 msecs
PM: resume of devices complete after 483.450 msecs
PM: resume devices took 0.480 seconds
Restarting tasks ... done.
^c
```

10.1.4 远程处理器消息传送

10.1.4.1 测试名称

- mxc_mcc_tty_test.out

10.1.4.1.1 位置

```
/unit_tests/Remote_Processor_Messaging
```

10.1.4.1.2 功能

测试 Cortex-A 和 Cortex-M 内核之间的通信。

10.1.4.1.3 配置

要运行 i.MX RPMsg 测试程序，请执行以下操作：确保使用正确的 Cortex-M4 处理器 RTOS 和 Linux 镜像。参见 i.MX7Dual 板的以下示例：

- rpmsg_pingpong_sdk_7dsdb.bin →在 i.MX7Dual SDB 板上使用的乒乓测试
- rpmsg_str_echo_sdk_7dsdb.bin →在 i.MX7Dual SDB 板上使用的 tty 字符串回声测试
- rpmsg_pingpong_sdk_7dval.bin →在 i.MX7Dual 12x12 LPDDR3 ARM2 板上使用的乒乓测试
- rpmsg_str_echo_sdk_7dval.bin →在 i.MX7Dual 12x12 LPDDR3 ARM2 板上使用的 tty 字符串回声测试

加载 Cortex-M4 处理器 RTOS 镜像，并在 U-Boot 中启动。通过 TFTP 服务器或 U-Boot 中可启动的 SD 卡加载 Cortex-M4 处理器 RTOS 镜像。

- 通过 TFTP 服务器加载 Cortex-M4 处理器 RTOS 镜像：
  - 启动 U-Boot 并停止。
  - 使用以下命令对响应 Cortex-M4 处理器 RTOS 镜像进行 TFTP 并启动它。

```
=> dhcp 0x7f8000 10.192.242.53:rpmsg_pingpong_sdk_7dval.bin; bootaux 0x7f8000
```

- 通过 SD 卡加载 Cortex-M4 处理器 RTOS 镜像：
  - 使用 MFG 工具创建可启动的 SD 卡。
  - 将 Cortex-M4 处理器 RTOS 文件复制到使用 VFAT 文件系统格式化的第一个分区。
- 更改 U-Boot 的默认 Cortex-M4 处理器 RTOS 名称。

```
=> setenv m4image '<The name of the M4/RTOS image>';save
```

设置 Cortex-M4 处理器使用的 bootargs 参数。

```
=> setenv run_m4_tcm 'if run loadm4image; then cp.b ${loadaddr} 0x7f8000 0x8000;
=> bootaux 0x7f8000; fi'; save
```

添加 run run_m4_tcm，可修改原始 bootcmd 参数。

```
=> setenv bootcmd "run run_m4_tcm; <original contents of the bootcmd>"; save
```

注意

注：当 Cortex-M4 处理器 RTOS 镜像运行时，i.MX 6SoloX 需要“uart_from_osc”。因此，应该在 i.MX 6SoloX 上修改 U-Boot 的 mmcargs。

```
=> setenv mmcargs 'setenv bootargs console=${console},${baudrate} root=${mmcroot}, uart_from_osc';save
```

运行 RPMsg 测试程序。*确保构建了 imx_rpmsg_pingpong.ko 和 imx_rpmsg_tty.ko。*使用 inmod imx_rpmsg_pingpong.ko 或 inmod imx_rpmsg_tty.ko 运行测试程序。

注意

注：请勿同时运行不同的测试程序。

10.1.4.1.4 用例和预期输出

运行以下命令，并在启动 RPMsg TTY 测试时确保 RPMsg TTY 接收程序在后端运行。

```
# ./mxc_mcc_tty_test.out /dev/ttyRPMSG30 115200 R 100 1000 &
Expected output:
mxc_mcc_tty_test.out:
$ insmod imx_rpmsg_tty.ko
$ imx_rpmsg_tty rpmsg0: new channel: 0x400 -> 0x1!
$ Install rpmsg tty driver!
$ echo deadbeaf > /dev/ttyRPMSG30
$ imx_rpmsg_tty rpmsg0: msg(<- src 0x1) deadbeaf len 8
```

10.1.5 看门狗（WDOG）

10.1.5.1 测试名称

- autorun-wdog.sh
- wdt_driver_test.out

10.1.5.1.1 位置

```
/unit_tests/Watchdog/
```

10.1.5.1.2 功能

测试看门狗定时器模块，该模块通过避免意外挂起、无限循环或编程错误来防止系统故障。

10.1.5.1.3 配置

无

10.1.5.1.4 用例和预期输出

```
Use case
./autorun-wdog.sh
or
./wdt_driver_test.out 1 2 0 &
Expected output
This should generate a reset after 3 seconds (a 1 second time-out and a 2 second sleep).
or
./wdt_driver_test.out 2 1 0
The system should keep running without being reset. This test requires the kernel to be executed
with the "jtag=on" on some platforms. Press "Ctrl+C" to terminate this test program.
```

10.2 存储

10.2.1 媒体本地总线

10.2.1.1 测试名称

- mxc_mlb150_test

10.2.1.1.1 位置

```
/unit_tests/Media_Local_Bus/
```

10.2.1.1.2 功能

MediaLB 是 PCB 上或芯片间通信总线，可实现通用硬件接口和软件 API 库的标准化。

10.2.1.1.3 配置

在菜单配置中启用以下模块：

```
Device Drivers > MXC support drivers > MXC Media Local Bus Driver > MLB support
```

仅在 i.MX6SX、i.MX6QP、i.MX6Q、i.MX6DL 上支持测试

10.2.1.1.4 用例和预期输出

```
./mxc_mlb150_test [-v] [-h] [-b] [-f fps] [-t casetype] [-q sync quadlets] [-p isoc
packet length]\n"
-v verbose
-h help
-b block io test
-f FPS, 256/512/1024/2048/3072/4096/6144
-t CASE, CASE can be 'sync', 'ctrl', 'async', 'isoc'
-q SYNC QUADLETS, quadlets per frame in sync mode, can be 1, 2, or 3
-p Packet length, package length in isoc mode, can be 188 or 196
```

10.2.2 MMC/SD/SDIO 主机

10.2.2.1 测试名称

- autorun-mmc-blockrw.sh
- autorun-mmc-fdisk.sh
- autorun-mmc-fs.sh
- autorun-mmc-mkfs.sh
- autorun-mmc.sh

10.2.2.1.1 位置

```
/unit_tests/MMC_SD_SDIO/
```

10.2.2.1.2 功能

一系列 MMC SD SDIO 测试执行以下指令：

- MMC/SD 读写测试
- MMC/SD 块读写测试
- MMC/SD fdisk 测试
- MMC/SD 文件系统测试
- MMC/SD mkfs 测试

10.2.2.1.3 配置

无

10.2.2.1.4 用例和预期输出

如果所有测试成功，则返回“通过测试”。

```
./autorun-mmc-blockrw.sh
./autorun-mmc-fdisk.sh
./autorun-mmc-fs.sh
./autorun-mmc-mkfs.sh
./autorun-mmc.sh
```

10.2.3 MMDC

10.2.3.1 测试名称

- mmdc2

10.2.3.1.1 位置

/unit_tests/MMDC/

10.2.3.1.2 功能

MMDC profiling utility.

10.2.3.1.3 配置

以下参数允许自定义 mmcd2 测试：

- 导出 MMDC_SLEEPTIME——定义分析持续时间（默认为 500ms）
- 导出 MMDC_LOOPCOUNT——定义分析时间（默认为 1，1 表示无限循环）
- 导出 MMDC_CUST_MADPCR1——定制 madpcr1

10.2.3.1.4 用例和预期输出

预期输出将打印性能分析结果

```
./mmdc2 [ARM:DSP1:DSP2:GPU2D:GPU2D1:GPU2D2:GPU3D:GPU3D2:GPUVG:VPU:M4:PXP:USB:SUM]
```

10.2.4 SATA

10.2.4.1 测试名称

- autorun-ata.sh

10.2.4.1.1 位置

/unit_tests/SATA/

10.2.4.1.2 功能

此测试将数据写入连接到 i.MX 板上 SATA 连接器的 SATA 驱动器。然后，读回数据，并与写入的数据进行比较。

10.2.4.1.3 配置

所需模块：pata_fsl.ko。所需硬件：SATA 驱动器。只有 i.MX 6 Quad 和 QuadPlus 支持 SATA。

10.2.4.1.4 用例和预期输出

```
./autorun-ata.sh
Expected output
Test should return "HDD test passes" if successful.
```

10.3 连接

10.3.1 增强型可配置串行外设接口（ECSPI）

10.3.1.1 测试名称

- mxc_spi_test1.out

10.3.1.1.1 位置

```
/unit_tests/ECSPI/
```

10.3.1.1.2 功能

该测试将最后一个参数的字节发送到特定的 SPI 设备。最大传输字节数：每字位数小于 8 位（含 8 位）时为 4096 字节，每字位数在 9 ~ 16 位之间时为 2048 字节，每字位数大于 17 位（含 17 位）时为 1024 字节。SPI 将从用户接收的数据写入 Tx FIFO，并等待 Rx FIFO 中的数据。一旦数据在 Rx FIFO 中就绪，就会被读取并发送给用户。

10.3.1.1.3 配置

对于 i.MX 6QuadPlus/Quad/Dual 自动电路板，这需要 ecspi 设备树。默认设备树禁用此功能。

10.3.1.1.4 用例和预期输出

```
./mxc_spi_test1.out -D 0 -s 1000000 -b 8 E6E0
./mxc_spi_test1.out -D 1 -s 1000000 -b 8 -H -O -C E6E0E6E00001E6E00000
Usage:
./mxc_spi_test1.out [-D spi_no] [-s speed] [-b bits_per_word] [-H] [-O] [-C] $lt;value>
<spi_no> - CSPI Module number in [0, 1, 2]
<speed> - Max transfer speed
<bits_per_word> - bits per word
-H - Phase 1 operation of clock
-O - Active low polarity of clock
-C - Active high for chip select
<value> - Actual values to be sent
```

10.3.2 ETM

10.3.2.1 测试名称

- etm

10.3.2.1.1 位置

```
/unit_tests/ETM/
```

10.3.2.1.2 功能

嵌入式跟踪宏单元（ETM）是一个可选的调试组件，可以重建程序执行。ETM 被设计为高速、低功耗的调试工具，只支持指令跟踪。这确保了占用面积最小化，并减少了栅极数量。

10.3.2.1.3 配置

10.3.2.1.4 用例和预期输出

```
# ./etm -h
Usage: ./etm [options]
Options:
--etm-3.3 ETM v3.3 trace data
--etm-3.4-alt-branch ETM v3.4 trace data with alternative branch encoding
--pft-1.1 PFT v1.1 trace data
--cycle-accurate Cycle-accurate tracing was enabled (Default 1)
--contextid-bytes Number of Context ID bytes (Default 4)
--formatter Enable Formatter Unpacking
--sourceid-match Enable Source ID from formatter. Also enables formatter
--print-long-waits Highlight long waits
--print-input Print input data
--print-config Print configuration data
--help Print usage information
```

10.3.3 内部集成电路（I2C）

10.3.3.1 测试名称

- mxc_i2c_slave_test.out

10.3.3.1.1 位置

```
/unit_tests/I2C/
```

10.3.3.1.2 功能

10.3.3.1.3 配置

无

10.3.3.1.4 用例和预期输出

10.3.4 IIM

10.3.4.1 测试名称

- mxc_iim_test.out

10.3.4.1.1 位置

```
/unit_tests/IIM_Driver/
```

10.3.4.1.2 功能

此测试可以从 bank 读取 iim 值或将值融合到 bank。

10.3.4.1.3 配置

无

10.3.4.1.4 用例和预期输出

对于读取和熔丝测试，输入值均应为十六进制格式。

```
Below is the usage for the read case.
./mxc_iim_test read -d <bank_addr>
<bank_addr> - bank address in fuse map file.
read - Indicate that this is a read operation.
Example:
./mxc_iim_test.out read -d 0xc30
Below is the usage for the fuse case.
./mxc_iim_test fuse -d <bank_addr> -v <value>
<bank_addr> - bank address in fuse map file.
<value> - Value to be writen to a bank.
fuse - Indicate that this is a write operation.
Example:
./mxc_iim_test.out fuse 0xc30 -v 0x40
```

10.3.5 键盘

10.3.5.1 测试名称

- autorun-keypad.sh
- mxc_keyb_test.sh

10.3.5.1.1 位置

```
/unit_tests/Keyboard/
```

10.3.5.1.2 功能

通过 USB 测试键盘输入。

10.3.5.1.3 配置

将键盘连接到 USB OTG 端口。

10.3.5.1.4 用例和预期输出

```
./autorun-keypad.sh
Outputs:
Print "PASS" status
./mxc_keyb_test.sh
```

```
Output:
An event will occur when a key is pressed
```

10.3.6 低功耗通用异步收发器（LPUART）

10.3.6.1 测试名称

- autorun-mxc_uart.sh
- mxc_uart_stress_test.out
- mxc_uart_test.out
- mxc_uart_xmit_test.out

10.3.6.1.1 位置

```
/unit_tests/UART/
```

10.3.6.1.2 功能

这些测试执行底层 UART 驱动程序，该驱动程序负责向核心 UART 驱动程序提供诸如 UART 端口信息和一组控制函数等信息。

10.3.6.1.3 配置

无

10.3.6.1.4 用例和预期输出

```
./autorun-mxc_uart.sh
./mxc_uart_stress_test.out /dev/ttymxc#
./mxc_uart_test.out /dev/ttymxc#
./mxc_uart_xmit_test.out /dev/ttymxc#
```

10.3.7 USB

10.3.7.1 测试名称

- autorun-usb-gadget.sh
- autorun-usb-host.sh

10.3.7.1.1 位置

```
/unit_tests/USB/
```

10.3.7.1.2 功能

该测试执行通用串行总线（USB）驱动程序，该驱动程序实施与 CHIPIDEA USB-HS OTG 控制器对接的标准 Linux 驱动程序接口。USB 提供了一种通用链路，可用于各种 PC 到外设的互连。它支持即插即用、端口扩展以及任何使用相同类型端口的新 USB 外设。

10.3.7.1.3 配置

所需模块：

- /lib/modules/$(kernel_version)/kernel/drivers/usb/gadget/g_ether.ko

- /lib/modules/$(kernel_version)/kernel/drivers/usb/gadget/arcotg_udc.ko
- /lib/modules/$(kernel_version)/kernel/drivers/usb/host/ehci-hcd.ko

10.3.7.1.4 用例和预期输出

```
./autorun-usb-gadget.sh
or
./autorun-usb-host.sh
```

10.4 图形

10.4.1 图形处理单元（GPU）

10.4.1.1 测试名称

- gpu.sh
- gpuinfo.sh

10.4.1.1.1 位置

```
/unit_tests/GPU
```

10.4.1.1.2 功能

GPU 功能测试

- tutorial3：测试 OpenGL ES 1.1 的基本功能
- tutorial4_es20：测试 OpenGL ES 2.0 的基本功能
- tiger：测试 OpenVG 1.1 的基本功能
- tvui：测试 Raster 2D 和 LibVivanteDK API

10.4.1.1.3 配置

为了使 gpu.sh 和 gpuinfo.sh 运行，请在目标电路板 defconfig 文件中添加下行内容：

- CONFIG_MXC_GPU_VIV=y

所需硬件：LVDS 显示屏和带有 GPU 的 i.MX SoC

10.4.1.1.4 用例和预期输出

```
./gpu.sh
```

- 预期输出为屏幕上正确绘制的帧。

- tutorial3：在屏幕中间旋转的带有纹理的立方体
- tutorial4_es20：在大球体内绘制玻璃球体（环境映射）。玻璃球体同时显示反射和折射效果。
- tiger：屏幕上旋转的老虎。
- tvui：绘制多个影片片段和一个电视控制屏。

输出示例如下：

```
# ./gpu.sh
---- Running < gpu.sh > test ----
/unit_tests/GPU /unit_tests/GPU
Rendered 100 frames in 624 milliseconds: 160.26 fps
id=43, a,b,g,r=0,8,8,8, d,s=16,0, AA=0,openvgbit=71
frames:100 -- fps:58.997051
press ESC to escape...
./gpu.sh: line 28: cd: /opt/viv_samples/hal/: No such file or directory
/unit_tests/GPU
---- Test < gpu.sh > ended ----
```

```
./gpuinfo.sh
```

- 控制台上打印有关 GPU 的信息。

```
# ./gpuinfo.sh
----  Running < gpuinfo.sh > test  ----
GPU Info
gpu : 0
model : 2000
revision : 5108
product : 0
eco : 0
gpu : 8
model : 320
revision : 5007
product : 0
eco : 0
gpu : 9
model : 355
revision : 1215
product : 0
eco : 0
VIDEO MEMORY:
gcvPOOL_SYSTEM:
Free : 134217728 B
Used : 0 B
Total : 134217728 B
gcvPOOL_CONTIGUOUS:
Used : 0 B
gcvPOOL_VIRTUAL:
Used : 0 B
NON PAGED MEMORY:
Used : 0 B
Paged memory Info
lowMem: 0 bytes
highMem: 0 bytes
CMA memory info
cma: 138485760 bytes
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
Idle percentage:0.000.000.000.000.000.00%
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
---- Test < gpuinfo.sh > ended ----
```

10.5 视频

10.5.1 显示器

10.5.1.1 测试名称

- autorun-fb.sh
- mxc_tve_test.sh
- mxc_fb_test.out
- mxc_epdc_fb_test.out
- mxc_epdc_v2_fb_test.out
- mxc_spdc_fb_test.out
- mxc_fb_vsync_test.out

10.5.1.1.1 位置

```
/unit_tests/Display/
```

10.5.1.1.2 功能

显示目录下的测试可测试 i.MX 系列电路板提供的一些显示选项。一些可以测试的器件包括 LVDS、HDMI 和 EPDC 屏。

具体地说，‘mxc_fb_test.out’测试以下功能：

- 基本 fb 操作
- 将背景/前景设置为 16 bpp fb
- 全局 Alpha 混合
- 色键测试
- 帧缓冲区平移
- 伽马测试

此外，EPDC 测试（‘mxc_epdc_fb_test.out’和‘mxc_epdc_v2_fb_test.out’）可测试以下功能：

- 基本更新
- 旋转更新
- 灰度帧缓冲区更新
- 自动波形选择更新
- 平移更新
- 覆盖更新
- 自动更新
- 动画模式更新
- 快速更新
- 部分更新到完全更新的过渡
- 测试像素移位效果
- 色彩映射表更新
- 碰撞测试模式

- 压力测试
- RGB565、Y8 帧缓冲区格式
- 0 度、90 度、180 度、270 度帧缓冲区旋转
- 帧缓冲区平移
- 使用备用帧缓冲区
- 自动波形模式选择
- 自动更新模式
- 强制单色更新功能和动画模式更新
- 支持使用灰度色彩映射表
- 快照、队列和合并更新方案
- EPDC 碰撞测试模式

10.5.1.1.3 配置

为了运行某些测试，需要更改目标电路板的 defconfig 文件。这些更改将添加以下测试所依赖的功能。

对于 autorun-fb.sh，‘mxc_fb_test.out’和‘mxc_fb_vsync_test.out’，将以下内容添加到目标电路板 defconfig 文件：

```
CONFIG_FB=y
CONFIG_FB_MXC=y
CONFIG_FB_MXC_EDID=y
CONFIG_FB_MXC_SYNC_PANEL=y
CONFIG_FB_MXC_LDB=y
CONFIG_FB_MXC_HDMI=y
```

对于‘mxc_epdc_fb_test.out’和‘mxc_epdc_v2_fb_test.out’，将以下内容添加到目标电路板 defconfig 文件中：

```
CONFIG_FB=y
CONFIG_FB_MXC=y
CONFIG_FB_MXC_EINK_PANEL=y
CONFIG_MFD_MAX17135=y
CONFIG_REGULATOR_MAX17135=y
CONFIG_MXC_PXP=y
CONFIG_DMA_ENGINE=y
```

10.5.1.1.4 用例和预期输出

```
# ./autorun-fb.sh
```

预期输出为：

```
---- Running < autorun-fb.sh > test ----
Checking for devnode: /dev/fb0
autorun-fb.sh: PASS devnode found: /dev/fb0
FB Blank test
Screen should be off
FB Color test
Setting FB to 16-bpp
```

```
Setting FB to 24-bpp
Setting FB to 32-bpp
FB panning test
autorun-fb.sh: Exiting PASS
Exiting PASS.
```

```
# ./mxc_tve_test.sh
```

预期输出为：

```
---- Running < mxc_tve_test.sh > test ----
Setting TV to NTSC mode
/unit_tests/Display/mxc_tve_test.sh: line 9: echo: write error: Invalid argument
/unit_tests/Display/mxc_tve_test.sh: line 11: /unit_tests/mxc_v4l2_output.out: No such
file or directory
Blank the display
Unblank the display
Setting TV to PAL mode
/unit_tests/Display/mxc_tve_test.sh: line 22: echo: write error: Invalid argument
/unit_tests/Display/mxc_tve_test.sh: line 23: /unit_tests/mxc_v4l2_output.out: No such
file or directory
Blank the display
Unblank the display
```

```
# ./mxc_fb_test.out
```

预期输出如下所示。没有任何失败消息显示时，此测试应为通过，并且屏上的显示应该是正确的。界面上应反映每项测试的一系列更新。对于几乎所有的测试，调试控制台中打印输出的文本描述了应在界面上观察到的图像。对于 i.MX6Quad fb0 和 fb1 测试，fb0 为后台帧缓冲区，fb1 则为前景覆盖帧缓冲区。

```
Opened fb: /dev/fb0 (DISP4 BG - DI1)
DISP4 BG - DI1: screen info: 1024x768 (virtual: 1024x1536) @ 32-bpp
Opened fb: /dev/fb1 (DISP4 FG)
DISP4 FG: screen info: 240x320 (virtual: 240x960) @ 16-bpp
@DISP4 BG - DI1: Set colorspace to 16-bpp
@DISP4 FG: Set colorspace to 16-bpp
Prepared DISP4 BG - DI1 (black) and DISP4 FG (white). Verify the screen and press any
key to continue!
@DISP4 BG - DI1: Succesfully changed screen to 1024x768 (virtual: 1024x768) @16-bpp
@DISP4 FG: Succesfully changed screen to 240x320 (virtual: 240x320) @16-bpp
Testing global alpha blending...
Fill the FG in black (screen is 240x320 @ 16-bpp)
Fill the BG in white (screen is 1024x768 @ 16-bpp)
Alpha is 0, FG is opaque
Alpha is 255, BG is opaque
Color key enabled
Color key disabled
Global alpha disabled
Pan test start.
@DISP4 FG: Set the colorspace to 16-bpp
Pan test done.
@DISP4 BG - DI1: Set colorspace to 16-bpp
Pan test start.
```

```
@DISP4 BG - DI1: Set the colorspace to 16-bpp
Pan test done.
Gamma test start.
Gamma 0.800000
Gamma 1.000000
Gamma 1.500000
Gamma 2.200000
Gamma 2.400000
Gamma test end.
Test bpp start
@DISP4 BG - DI1: Set colorspace to 32-bpp
@DISP4 BG - DI1: Fill the screen in red
@DISP4 BG - DI1: Set colorspace to 24-bpp
@DISP4 BG - DI1: Fill the screen in blue
@DISP4 BG - DI1: Set colorspace to 16-bpp
@DISP4 BG - DI1: Fill the screen in green
Test bpp end
```

```
# ./mxc_epdc_fb_test.out [-h] [-a] [-n]
EPDC framebuffer driver test program.
Usage: mxc_epdc_fb_test [-h] [-a] [-p delay] [-u s/q/m] [-n <expression>]
-h Print this message
-a Enabled animation waveforms for fast updates (tests 8-9)
-p Provide a power down delay (in ms) for the EPDC driver
0 - Immediate (default)
-1 - Never
<ms> - After <ms> milliseconds
-u Select an update scheme
s - Snapshot update scheme
q - Queue update scheme
m - Queue and merge update scheme (default)
-n Execute the tests specified in expression
Expression is a set of comma-separated numeric ranges
If not specified, all tests except Stress are run
Example:
./mxc_epdc_fb_test.out -n 1-3,5,7
EPDC tests:
1 - Basic Updates
2 - Rotation Updates
3 - Grayscale Framebuffer Updates
4 - Auto-waveform Selection Updates
5 - Panning Updates
6 - Overlay Updates
7 - Auto-Updates
8 - Animation Mode Updates
9 - Fast Updates
10 - Partial to Full Update Transitions
11 - Test Pixel Shifting Effect
12 - Colormap Updates
13 - Collision Test Mode
14 - Stress Test
15 - Dithering Y8->Y1 Test
16 - Dithering Y8->Y4 Test
17 - Hardware Dithering Test
18 - Advanced Algorithm Test
```

没有任何失败消息显示时，全套测试应为通过。界面上应反映每项测试的一系列更新。对于几乎所有的测试，调试控制台中打印输出的文本描述了应在界面上观察到的图像。

mxc_epdc_v2_fb_test.out：没有任何失败消息显示时，全套测试应为通过。界面上应反映每项测试的一系列更新。对于几乎所有的测试，调试控制台中打印输出的文本描述了应在界面上观察到的图像。

```
# ./mxc_spdc_fb_test.out
---- Running < ./mxc_spdc_fb_test.out > test ----
Unable to open /dev/fb5
```

```
# ./mxc_fb_vsync_test.out
Usage:
/unit_tests/Display# ./mxc_fb_vsync_test.out <fb #> <count>
<fb #> the framebuffer number
<count> the frames to be rendered
Example:
/unit_tests/Display# echo 0 > /sys/class/graphics/fb0/blank
/unit_tests/Display# ./mxc_fb_vsync_test.out 0 100
```

使用 100 作为<count>参数时，预期输出如下

```
total time for 100 frames = 1655674 us = 60 fps
```

10.5.2 高清多媒体接口（HDMI）和显示端口（DP）概述

10.5.2.1 测试名称

- mxc_cec_test.out

10.5.2.1.1 位置

`/unit_tests/HDMI/`

10.5.2.1.2 功能

验证 HDMI CEC 功能并向 HDMI 接收器发送断电命令。

10.5.2.1.3 配置

要使 mxc_cec_test.out 正常工作，请将以下行添加到目标电路板的 defconfig 文件中：

```
CONFIG_MXC_HDMI_CEC=y
```

硬件应支持 HDMI，电视应支持 HDMI CEC

10.5.2.1.4 用例和预期输出

```
./mxc_cec_test.out
```

10.5.3 视频处理单元（VPU）

10.5.3.1 i.MX 6 测试

- autorun-vpu.sh

- mxc_vpu_test.out

10.5.3.1.1 位置

```
/unit_tests/VPU/
```

10.5.3.1.2 功能

VPU 测试在视频处理单元（VPU）上执行以下选项：

- 解码流并在 LCD 上显示。
- 解码流并保存到文件。
- 使用配置文件解码流。
- 编码 YUV 流并保存到文件中。
- 对摄像头中的图像进行编码，并将其解码后显示在 LCD 上。
- 同时解码不同格式的多个流。
- 同时解码和编码。
- 输出到电视输出。
- 使用 VDI 测试 VPU（通过 IPU 进行硬件去隔行）。

10.5.3.1.3 配置

这些测试需要/usr/lib/下的 libvpu.so 和 LCD 显示屏。此测试需要 i.MX 6QuadPlus/Quad/Dual SoC。

10.5.3.1.4 用例和预期输出

```
./autorun-vpu.sh
Decode one stream and display on the LCD.
To test MPEG-4 decode and display to screen:
./mxc_vpu_test.out -D "-i /usr/vectors/file.m4v -f 0"
To test H.263 decode and display to screen:
./mxc_vpu_test.out -D "-i /usr/vectors/file.263 -f 1"
To test H.264 decode and display to screen:
./mxc_vpu_test.out -D "-i /usr/vectors/file.264 -f 2"
You can get the mp4 test file from the imx-test.git server.
It is located under test/mxc_vpu_test/configs/akiyo.mp4.
Decode a stream and save to a file.
To test MPEG-4 decode and save to file:
./mxc_vpu_test.out -D "-i /usr/vectors/file.m4v -f 0 -o out.yuv"
To test H.263 decode and save to file:
./mxc_vpu_test.out -D "-i /usr/vectors/file.263 -f 1 -o out.yuv"
To test H.264 decode and save to file:
./mxc_vpu_test.out -D "-i /usr/vectors/file.264 -f 2 -o out.yuv"
Decode a stream using a config file.
Change options in config file, e.g, config_dec. Input correct input filename, output filename, format,
./mxc_vpu_test.out -C config_dec
Encode a YUV stream and save to a file.
To test MPEG-4 encode and save to a file:
./mxc_vpu_test.out -E "-i file.yuv -w 240 -h 320 -f 0 -o file.mpeg4"
To test H.263 encode and save to a file:
./mxc_vpu_test.out -E "-i file.yuv -w 240 -h 320 -f 1 -o file.263"
To test H.264 encode and save to a file:
./mxc_vpu_test.out -E "-i file.yuv -w 240 -h 320 -f 2 -o file.264"
Encode an image from the camera and decode it to display on the LCD.
```

```
To encode an MPEG4 image from the camera and display on the LCD: that
./mxc_vpu_test.out -L "-f 0 -w 1280 -h 720"
To encode an H263 image from the camera and display on the LCD:
./mxc_vpu_test.out -L "-f 1 -w 1280 -h 720"
To encode an H264 image from the camera and display on the LCD:
./mxc_vpu_test.out -L "-f 2 -w 1280 -h 720"
Decode multiple streams with different formats simultaneously.
Decoder one H264 and one MPEG4 streams.
./mxc_vpu_test.out -D "-i/vectors/file.264 -f 2" -D "-i ./akiyo.mp4 -f 0 -o akiyo.yuv"
Decode and encode simultaneously.
Encode one MPEG-4 stream and decode one H.264 stream simultaneously.
./mxc_vpu_test.out -E "-w 176 -h 144 -f 0 -o enc.m4v" -D "-i/vectors/file.264 -f
Test VPU with TV out.
Decoder one stream as normal VPU test. For example, H264 video stream:
./mxc_vpu_test.out -D "-i filename -f 2"
Test VPU with VDI (HW deinterlace via IPU).
Select one stream with top and bottom fields are interlaced.
av_stress2_dsmcc4m_1_C1_V11_A6.mp4_track1.h264
To decode the stream and display on LCD:
./mxc_vpu_test.out -D "-i av_stress2_dsmcc4m_1_C1_V11_A6.mp4_track1.h264 -f2"
To decode the stream and display on LCD using high motion stream video De Interlacing algorithm:
./mxc_vpu_test.out -D "-i av_stress2_dsmcc4m_1_C1_V11_A6.mp4_track1.h264 -v h -f2"
To decode the stream and display on LCD using low motion stream video De Interlacing algorithm:
./mxc_vpu_test.out -D "-i av_stress2_dsmcc4m_1_C1_V11_A6.mp4_track1.h264 -v l -f2"
To decode the stream and display on LCD having input in NV12 pixel format:
./mxc_vpu_test.out -D "-i av_stress2_dsmcc4m_1_C1_V11_A6.mp4_track1.h264 -v
```

10.5.3.2 i.MX 8M Quad 的测试

10.5.3.2.1 位置

```
/unit_tests/VPU/hantro
```

10.5.3.2.2 功能

VPU 测试在 VPU 上执行以下选项：

- 解码流并保存到文件。

10.5.3.2.3 用例和预期输出

解码不同编解码器的示例：

```
/unit_tests/VPU/hantro/g2dec -P -b -ibs -Oout.yuv test.hevc
/unit_tests/VPU/hantro/g2dec -P -b -iivf -Oout.yuv test.vp9
/unit_tests/VPU/hantro/hx170dec -P -Oout.yuv test.h264
/unit_tests/VPU/hantro/mx170dec -P -Oout.yuv test.mpeg4
/unit_tests/VPU/hantro/m2x170dec -P -Oout.yuv test.mpeg2
/unit_tests/VPU/hantro/vx170dec -P -Oout.yuv test.vc1
/unit_tests/VPU/hantro/vp8x170dec -P -Oout.yuv test.vp8
/unit_tests/VPU/hantro/vp6dec -P -Oout.yuv test.vp6
/unit_tests/VPU/hantro/rvx170dec -P -Oout.yuv test.rv
/unit_tests/VPU/hantro/jx170dec -P -Oout.yuv test.jpg
/unit_tests/VPU/hantro/ax170dec -P -Oout.yuv test.avs
```

10.5.3.3 i.MX 8M Mini 的测试

10.5.3.3.1 位置

```
/unit_tests/VPU/hantro
```

10.5.3.3.2 功能

VPU 测试在 VPU 上执行以下选项：

- 解码流并保存到文件。
- 对 YUV 流进行编码并保存到文件中。

10.5.3.3.3 用例和预期输出

解码器示例：

```
/unit_tests/VPU/hantro/g2dec -P -b -ibs -Oout.yuv test.hevc
/unit_tests/VPU/hantro/g2dec -P -b -iivf -Oout.yuv test.vp9
/unit_tests/VPU/hantro/hx170dec -P -Oout.yuv test.h264
/unit_tests/VPU/hantro/vp8x170dec -P -Oout.yuv test.vp8
```

编码器示例：

```
/unit_tests/VPU/hantro/h264_testenc -w176 -h144 -o temp.h264 -i test.yuv
/unit_tests/VPU/hantro/vp8_testenc -w176 -h144 -o temp.h264 -i test.yuv
```

10.5.3.4 i.MX 8QuadXPlus 和 8QuadMax 的测试

10.5.3.4.1 位置

```
/unit_tests/V4L2_VPU/
```

10.5.3.4.2 功能

VPU 测试在 VPU 上执行以下选项：

- 解码流并保存到文件。
- 对 YUV 流进行编码并保存到文件中。

10.5.3.4.3 用例和预期输出

解码器示例，有助于列出不同编解码器的“ifmt”值：

```
/unit_tests/V4L2_VPU/mxc_v4l2_vpu_dec.out ifile test.hevc ifmt 13 ofmt 1 ofile out.yuv
/unit_tests/V4L2_VPU/mxc_v4l2_vpu_dec.out ifile test.h264 ifmt 1 ofmt 1 ofile out.yuv
```

编码器示例（仅限 H.264）：

```
/unit_tests/V4L2_VPU/mxc_v4l2_vpu_enc.out ifile --key 0 --name input_720p.nv12 --fmt nv12 --size 1280
720 --loop 1 encoder --key 1 --source 0 --size 1280
720 --framerate 30 --bitrate 3000000 --lowlatency 1 ofile --key 2 --source 1 --name test.h264
```

10.5.4 JPEG 编码器和解码器

10.5.4.1 测试名称

- encoder_test
- decoder_test

10.5.4.1.1 位置

```
/unit_tests/JPEG
```

10.5.4.1.2 功能

encoder_test 接收一种支持格式的原始文件作为输入，并生成一个 JPEG 文件作为输出，其分辨率和像素格式与输入相同。应用程序将原始文件填充到一个 V4L2 输出缓冲区中，将其排队放入驱动程序中，并期望在一个采集缓冲区中取出 JPEG 图像。

decoder_test 接收一种支持格式的 JPEG 文件作为输入，并生成一个原始文件作为输出，其分辨率和像素格式与输入相同。应用程序将 jpeg 文件填充到一个 V4L2 输出缓冲区中，将其排队放入驱动程序中，并期望在一个采集缓冲区中取出原始图像。

10.5.4.1.3 配置

无特殊配置。

10.5.4.1.4 用例和预期输出

运行应用程序以获取使用情况：

```
./decoder_test.out
```

使用情况：

```
./decoder_test.out -d </dev/videoX> -f <INPUT_FILENAME> -w <width> -h <height> -p <pixel_format>
```

支持的格式：

```
yuv420: 2-planes, Y and UV-interleaved, same as NV12
yuv422: packed YUYV
rgb24: packed RGB
yuv444: packed YUV
gray: Y8 or Y12 or Single Component
argb: packed ARGB
```

输入文件必须是符合指定宽度、高度和像素格式的 JPEG 文件。输出是当前文件夹中名为“outfile”的原始文件，具有与输入文件相同的宽度、高度和像素格式。

```
./encoder_test.out
```

使用情况：

```
./encoder_test.out -d </dev/videoX> -f <INPUT_FILENAME> -w <width> -h <height> -p <pixel_format>
```

支持的格式：

```
yuv420: 2-planes, Y and UV-interleaved, same as NV12
yuv422: packed YUYV
rgb24: packed RGB
yuv444: packed YUV
gray: Y8 or Y12 or Single Component
argb: packed ARGB
```

输入文件必须是符合指定宽度、高度和像素格式的原始文件。输出是当前文件夹中名为“outfile.jpeg”的 JPEG 文件，具有与输入文件相同的宽度、高度和像素格式。

10.6 音频

10.6.1 高级 Linux 声音架构（ALSA）片上系统（ASoC）声音

10.6.1.1 测试名称

- mxc_tuner_test.out

10.6.1.1.1 位置

```
/unit_tests/ALSA/
```

10.6.1.1.2 功能

使用 ALSA 测试音频功能。

10.6.1.1.3 配置

i.MX 所有产品均支持 ALSA，ALSA 可用于进行播放、录音和扬声器测试。要使用此调谐器测试，需要 i.MX 6 自动参考板专有的调谐器硬件

10.6.1.1.4 用例和预期输出

10.6.2 异步采样率转换器（ASRC）

10.6.2.1 测试名称

- mxc_asrc_test.out

10.6.2.1.1 位置

```
/unit_tests/ASRC
```

10.6.2.1.2 功能

将 WAV 转换为不同的采样率。

10.6.2.1.3 配置

无

10.6.2.1.4 用例和预期输出

```
#/unit_tests/ASRC/mxc_asrc_test.out -to 48000 /unit_tests/ASRC/audio8k16S.wav
```

```
audio48k16S.wav
---- Running < /unit_tests/ASRC/mxc_asrc_test.out >  test ----
Pair A requested
All tests passed with success
```

可以通过以下命令获取 mxc_asrc_test.out 的更多用法：

```
#/unit_tests/ASRC/mxc_asrc_test.out -h
---- Running  < /unit_tests/ASRC/mxc_asrc_test.out > test ----
************************************************
* Test aplication for ASRC
* Options :
-to <output sample rate> <origin.wav$gt; <converted.wav>
<input clock source> <output clock source>
input clock source types are:
0 -- INCLK_NONE
1 -- INCLK_ESAI_RX
2 -- INCLK_SSI1_RX
3 -- INCLK_SSI2_RX
4 -- INCLK_SPDIF_RX
5 -- INCLK_MLB_CLK
6 -- INCLK_ESAI_TX
7 -- INCLK_SSI1_TX
8 -- INCLK_SSI2_TX
9 -- INCLK_SPDIF_TX
10 -- INCLK_ASRCK1_CLK
default option for output clock source is 0
output clock source types are:
0 -- OUTCLK_NONE
1 -- OUTCLK_ESAI_TX
2 -- OUTCLK_SSI1_TX
3 -- OUTCLK_SSI2_TX
4 -- OUTCLK_SPDIF_TX
5 -- OUTCLK_MLB_CLK
6 -- OUTCLK_ESAI_RX
7 -- OUTCLK_SSI1_RX
8 -- OUTCLK_SSI2_RX
9 -- OUTCLK_SPDIF_RX
10 -- OUTCLK_ASRCK1_CLK
default option for output clock source is 10
************************************************
```

10.7 安全性

10.7.1 显示内容完整性校验程序（DCIC）

10.7.1.1 测试名称

- mxc_dcic_test.out

10.7.1.1.1 位置

```
/unit_tests/DCIC/
```

10.7.1.1.2 功能

DCIC 的目标是验证发送到显示器的关键安全信息是否未损坏。

10.7.1.1.3 配置

无

10.7.1.1.4 用例和预期输出

```
# ./mxc_dcic_test.out -bw 18 -dev 1
```

mxc_dcic_test.out 的预期输出：

```
Opened fb0
open /dev/dcic1
bpp=16, bus_width=18
Config ROI=1
Config ROI=3
Config ROI=5
ROI=0,crcRS=0x0, crcCS=0x0
ROI=1,crcRS=0x6cd6b18d, crcCS=0x6cd6b18d
ROI=2,crcRS=0x0, crcCS=0x0
ROI=3,crcRS=0xc9da7ae6, crcCS=0xc9da7ae6
ROI=4,crcRS=0x0, crcCS=0x0
ROI=5,crcRS=0xb5ba1453, crcCS=0xb5ba1453
ROI=6,crcRS=0x0, crcCS=0x0
ROI=7,crcRS=0x0, crcCS=0x0
ROI=8,crcRS=0x0, crcCS=0x0
ROI=9,crcRS=0x0, crcCS=0x0
ROI=10,crcRS=0x0, crcCS=0x0
ROI=11,crcRS=0x0, crcCS=0x0
ROI=12,crcRS=0x0, crcCS=0x0
ROI=13,crcRS=0x0, crcCS=0x0
ROI=14,crcRS=0x0, crcCS=0x0
ROI=15,crcRS=0x0, crcCS=0x0
All ROI CRC check success!
```

10.7.2 SIM

10.7.2.1 测试名称

- mxc_sim_test.out

10.7.2.1.1 位置

```
/unit_tests/SIM/
```

10.7.2.1.2 功能

SIM 卡接口的基本测试。

10.7.2.1.3 配置

无

10.7.2.1.4 用例和预期输出

```
/unit_tests/mxc_sim_test.out
```

```
Expected output
atr[0]= 0x3b atr[1]= 0x68 atr[2]= 0x0 atr[3]= 0x0 atr[4]= 0x0 atr[5]= 0x73 atr[6]=
0xc8
atr[7]= 0x40 atr[8]= 0x0 atr[9]= 0x0 atr[10]= 0x90 atr[11]= 0x0
rx[0] = 0x6e rx[1] = 0x0
rx[0] = 0x6d rx[1] = 0x0
rx[0] = 0x6e rx[1] = 0x0
```

10.7.3 SNVS 实时时钟（SRTC）

10.7.3.1 测试名称

- autorun-rtc.sh
- rtctest.out
- rtcwakeup.out

10.7.3.1.1 位置

```
/unit_tests/SRTC/
```

10.7.3.1.2 功能

这些测试检查用于保持时间和日期的实时时钟（RTC）模块。它为用户提供了可证明的时间，如果检测到对计数器的篡改，它可以发出警报。

10.7.3.1.3 配置

要使 autorun-rtc.sh、rtcest.out 和 rtcwakeup.out 正常工作，请将以下行添加到目标电路板 defconfig 文件中：

```
CONFIG_RTC_DRV_SNVS=y
```

10.7.3.1.4 用例和预期输出

```
./autorun-rtc.sh
or
./rtctest.out $lt;arg>
--full run all tests
--no-periodic don't run periodic interrupt tests
or
./rtcwakeup.out -d rtc0 -m mem -s 10
Expected output
autorun-rtc.sh: Exit with PASS results.
rtctest.out: The program ends with "Test complete" status.
rtcwakeup.out: System is wakeup after 10s.
```

i.MX 7D Sabre SD 的预期输出

- autorun-rtc.sh
-

```
autorun-rtc.sh
i.MX7D
Checking for devnode: /dev/rtc0
autorun-rtc.sh: PASS devnode found: /dev/rtc0
Running test case: ./rtctest.out --no-periodic
```

```
RTC Driver Test Example.
Counting 5 update (1/sec) interrupts from reading /dev/rtc0: 1 2 3 4 5
Again, from using select(2) on /dev/rtc0: 1 2 3 4 5
Current RTC date/time is 21-2-2017, 23:13:07.
Alarm time now set to 23:13:12.
Waiting 5 seconds for alarm... okay. Alarm rang.
*** Test complete ***
Typing "cat /proc/interrupts" will show 1 more events on IRQ rtc.
autorun-rtc.sh: PASS test case: ./rtctest.out --no-periodic
rtc irqs before running unit test: 303
rtc irqs after running unit test: 314
so rtc irqs during test was:
11
checking rtc interrupts PASS autorun-
rtc.sh: Exiting PASS
```

- rtctest.out --full

```
./rtctest.out --full
RTC Driver Test Example.
Counting 5 update (1/sec) interrupts from reading /dev/rtc0: 1 2 3 4 5
Again, from using select(2) on /dev/rtc0: 1 2 3 4 5
Current RTC date/time is 21-2-2017, 23:14:48.
Alarm time now set to 23:14:53.
Waiting 5 seconds for alarm... okay. Alarm rang.
Periodic IRQ rate was 1Hz.
Counting 20 interrupts at:
2Hz:  1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
4Hz:  1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
8Hz:  1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
16Hz: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
32Hz: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
64Hz: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
***      complete ***
Typing "cat /proc/interrupts" will show 131 more events on IRQ rtc.
```

- rtctest.out --no-periodic

```
/rtctest.out --no-periodic
RTC Driver Test Example.
Counting 5 update (1/sec) interrupts from reading /dev/rtc0: 1 2 3 4 5
Again, from using select(2) on /dev/rtc0: 1 2 3 4 5
Current RTC date/time is 21-2-2017, 23:16:24.
Alarm time now set to 23:16:29.
Waiting 5 seconds for alarm... okay. Alarm rang.
*** Test complete ***
Typing "cat /proc/interrupts" will show 1 more events on IRQ rtc.
```

- rtcwakeup.out -d rtc0 -m mem -s 10

```
./rtcwakeup.out -d rtc0 -m mem -s 10
rtcwakeup.out: wakeup from "mem" using rtc0 at Wed Feb 22 23:17:29 2017
PM: Syncing filesystems ... done.
Freezing user space processes ... (elapsed 0.001 seconds) done.
Freezing remaining freezable tasks ... (elapsed 0.001 seconds) done.
Suspending console(s) (use no_console_suspend to debug)
```

```
PM: suspend of devices complete after 639.100 msecs
PM: suspend devices took 0.640 seconds
PM: late suspend of devices complete after 1.236 msecs
PM: noirq suspend of devices complete after 1.202 msecs
Disabling non-boot CPUs ...
CPU1: shutdown
Turn off Mega/Fast mix in DSM
Enabling non-boot CPUs ...
CPU1 is up
PM: noirq resume of devices complete after 0.756 msecs
imx-sdma 30bd0000.sdma: loaded firmware 4.2
PM: early resume of devices complete after 0.972 msecs
PM: resume of devices complete after 483.302 msecs
PM: resume devices took 0.480 seconds
Restarting tasks ... done.
```

第 11 章
修订历史

11.1 修订历史

请参见下表，了解修订历史记录。

表 92. 修订历史

版本号	日期	实质性变更
L4.9.51_imx8qxp-alpha	11/2017	初版发布
L4.9.51_imx8qm-beta1	12/2017	新增 i.MX 8QuadMax
L4.9.51_imx8mq-beta	12/2017	新增 i.MX 8M Quad
L4.9.51_8qm-beta2/8qxp-beta	02/2018	新增 i.MX 8QuadMax Beta2 和 i.MX 8QuadXPlus Beta
L4.9.51_imx8mq-ga	03/2018	新增 i.MX 8M Quad GA
L4.9.88_2.0.0-ga	05/2018	i.MX 7ULP 和 i.MX 8M Quad GA 版本
L4.9.88_2.1.0_8mm-alpha	06/2018	i.MX 8M Mini Alpha 版本
L4.9.88_2.2.0_8qxp-beta2	07/2018	i.MX 8QuadXPlus Beta2 版本
L4.9.123_2.3.0_8mm	09/2018	i.MX 8M Mini GA 版本
L4.14.62_1.0.0_beta	11/2018	i.MX 4.14 内核升级，Yocto Project Sumo 升级
L4.14.78_1.0.0_ga	01/2019	i.MX6、i.MX7、i.MX8 系列 GA 版本
L4.14.98_2.0.0_ga	04/2019	i.MX 4.14 内核升级和电路板更新
L4.19.35_1.0.0	07/2019	i.MX 4.19 Beta 内核和 Yocto Project 更新
L4.19.35_1.1.0	10/2019	i.MX 4.19 内核和 Yocto Project 更新
L5.4.3_1.0.0	03/2020	i.MX 5.4 内核和 Yocto Project 更新
Linux LF5.4.3_2.0.0	04/2020	面向 i.MX 8M Plus 和 8DXL EVK 板的 i.MX 5.4 Alpha 版本
L5.4.24_2.1.0	06/2020	i.MX 8M Plus 的 i.MX 5.4 Beta 版本、8DXL 的 Alpha2 版本，以及已发布 i.MX 电路板的整合 GA
L5.4.47_2.2.0	09/2020	i.MX 8M Plus 的 i.MX 5.4 Beta2 版本、8DXL 的 Beta 版本，以及已发布的 i.MX 电路板的整合 GA
L5.4.70_2.3.0	12/2020	i.MX 5.4 整合 GA，适用于发布 i.MX 电路板，包括 i.MX 8M Plus 和 i.MX 8DXL 电路板
L5.4.70_2.3.0	01/2021	更新了“运行 ARM Cortex-M4 镜像”一节中的命令行
Linux LF5.10.9_1.0.0	03/2021	升级到 5.10.9 内核
Linux LF5.10.35_2.0.0	06/2021	升级到 5.10.35 内核
Linux LF5.10.52_2.1.0	09/2021	更新 i.MX 8ULP Alpha，内核升级到 5.10.52
Linux LF5.10.72_2.2.0	12/2021	将内核升级到 5.10.72 并更新了 BSP

How To Reach Us

Home Page:

nxp.com

Web Support:

nxp.com/support

Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including “typicals,” must be validated for each customer application by customer's technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.

Right to make changes - NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Security — Customer understands that all NXP products may be subject to unidentified or documented vulnerabilities. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer’s applications and products. Customer’s responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer’s applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately. Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP. NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

For more information, please visit: http://www.nxp.com

For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 17 December 2021

Document identifier: IMXLXRM

arm