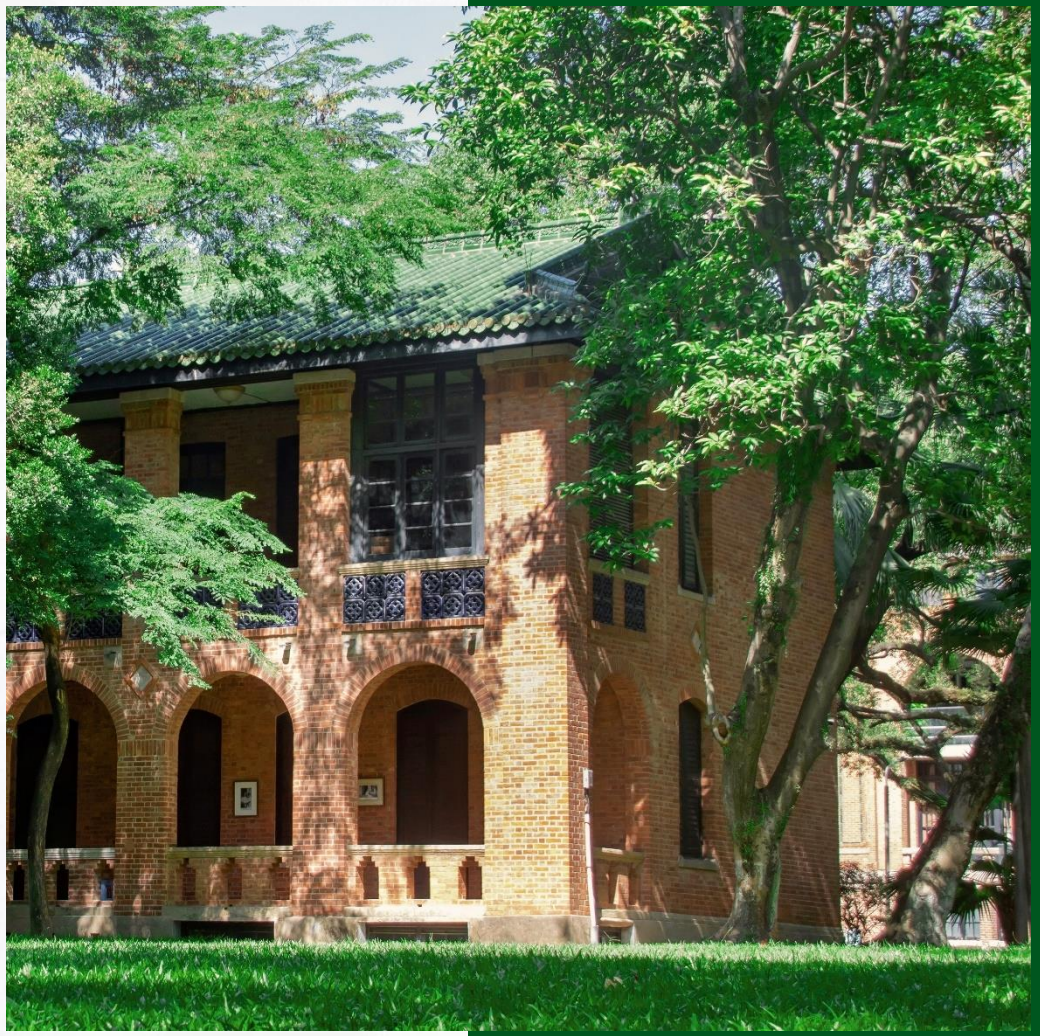
中山大學
SUN YAT-SEN UNIVERSITY

鸿蒙操作系统
输入输出管理



HarmonyOS设备驱动管理

1. HarmonyOS采用多内核（Linux内核或者LiteOS）设计，支持系统在不同资源容量的设备部署。
2. 当相同的硬件部署不同内核时，如何能够让设备驱动程序在不同内核间平滑迁移，消除驱动代码移植适配和维护的负担，是OpenHarmony驱动子系统需要解决的重要问题。
3. OpenHarmony驱动子系统的关键特性和能力：
   - 弹性化的框架能力:通过构建弹性化的框架能力，可支持在百K级别到百兆级容量的终端产品形态部署。
   - 规范化的驱动接口:定义了常见驱动接口，为驱动开发者和使用者提供丰富、稳定接口。
   - 组件化的驱动模型：为开发者提供更精细化的驱动管理，开发者可以对驱动进行组件化拆分。
   - 归一化的配置界面：提供统一的配置界面，构建跨平台的配置转换和生成工具。

HarmonyOS设备驱动框架

HDF（Hardware Driver Foundation）驱动框架，为驱动开发者提供驱动框架能力，包括驱动加载、驱动服务管理和驱动消息机制。 HDF构建统一的驱动架构平台，为驱动开发者提供更精准、更高效的开发环境，力求做到一次开发，多系统部署。

图9-1 HDF驱动框架

HarmonyOS设备驱动框架

1. HDI（Hardware Device Interface，硬件设备统一接口）层：通过规范化的设备接口标准，为系统提供统一、稳定的硬件设备操作接口。
2. HDF驱动框架：提供统一的硬件资源管理、驱动加载管理、设备节点管理、设备电源管理以及驱动服务模型等功能。
3. 平台驱动：为外设驱动提供Board硬件操作统一接口，同时对Board硬件操作进行统一的适配接口抽象以便于不同平台迁移。

图9-1 HDF驱动框架

HarmonyOS设备驱动框架

4. 外设驱动模型：面向外设驱动，提供常见的驱动抽象模型，提供标准化的器件驱动；提供驱动模型抽象，屏蔽驱动与不同系统组件间的交互。
5. 操作系统抽象层（OSAL，Operating System Abstraction Layer）：提供统一封装的内核操作相关接口，屏蔽不同系统操作差异。

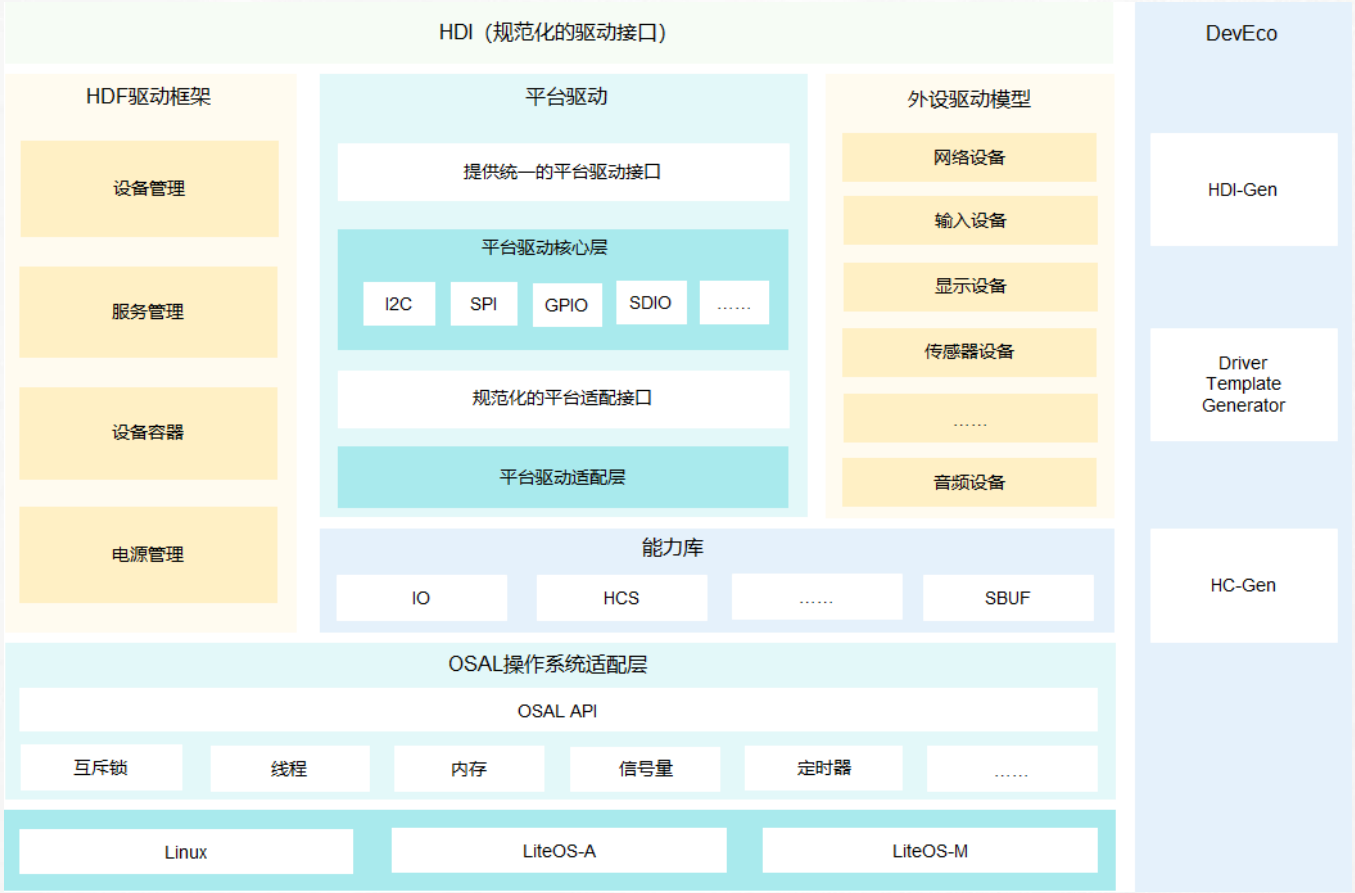


图9-1 HDF驱动框架

HarmonyOS设备驱动框架

HDF设备驱动模型

1. HDF框架将一类设备驱动放在同一个Host（设备容器）里面，用于管理一组设备的启动加载等过程。在划分Host时，驱动程序是部署在一个Host还是部署在不同的Host，主要根据驱动程序之间是否存在耦合性。
2. Device对应一个真实的物理设备。DeviceNode是设备的一个部件，Device至少有一个DeviceNode。每个DeviceNode可以发布一个设备服务。
3. 驱动即驱动程序，每个DevicdNode唯一对应一个驱动，实现和硬件的功能交互。

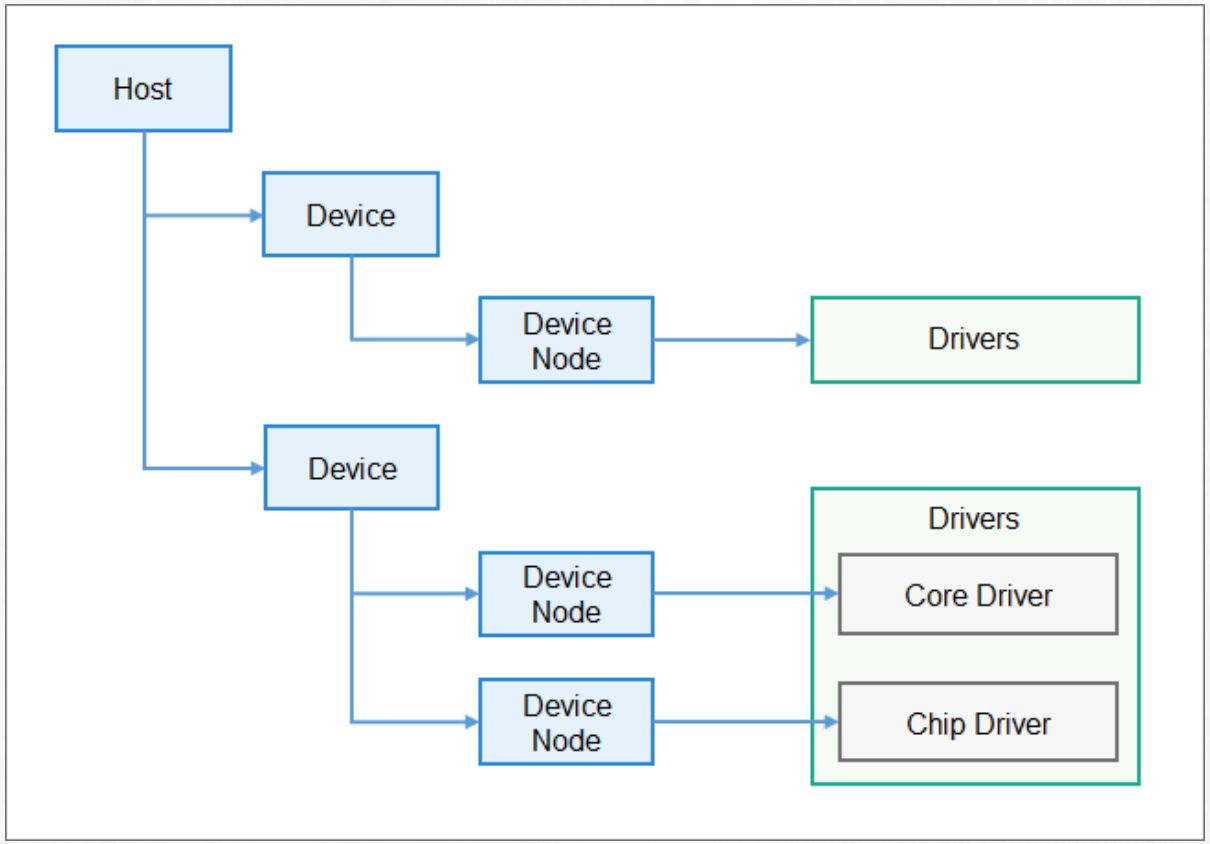


图9-2 HDF驱动模型

中断

1. 中断是指出现需要时，CPU暂停执行当前程序，转而执行新程序的过程。即在程序运行过程中，出现了一个必须由CPU立即处理的事务。此时，CPU暂时中止当前程序的执行转而处理这个事务，这个过程就叫做中断。
2. 外设可以在没有CPU介入的情况下完成一定的工作，但某些情况下也需要CPU为其执行一定的工作。通过中断机制，在外设不需要CPU介入时，CPU可以执行其它任务，而当外设需要CPU时，将通过产生中断信号使CPU立即中断当前任务来响应中断请求。这样可以使CPU避免把大量时间耗费在等待、查询外设状态的操作上，大大提高系统实时性以及执行效率。
3. Huawei LiteOS的中断特性：
   - 中断共享，且可配置。
   - 中断嵌套，即高优先级的中断可抢占低优先级的中断，且可配置。
   - 使用独立中断栈，可配置。
   - 可配置支持的中断优先级个数。
   - 可配置支持的中断数。

中断

4. 中断控制器一方面接收其它外设中断引脚的输入，另一方面会发出中断信号给CPU。可以通过对中断控制器编程来打开和关闭中断源、设置中断源的优先级和触发方式。常用的中断控制器有VIC（Vector Interrupt Controller）和GIC（General Interrupt Controller）。在ARM Cortex-A7中使用的中断控制器是GIC。
5. CPU收到中断控制器发送的中断信号后，中断当前任务来响应中断请求。
6. 异常指可以打断CPU正常运行流程的一些事情，如未定义指令异常、试图修改只读的数据异常、不对齐的地址访问异常等。
7. 中断和异常处理的入口为中断向量表，中断向量表包含各个中断和异常处理的入口函数。

图9-3 中断向量表

GPIO

1. GPIO（General-purpose input/output）即通用型输入输出。通常，GPIO控制器通过分组的方式管理所有GPIO管脚，每组GPIO有一个或多个寄存器与之关联，通过读写寄存器完成对GPIO管脚的操作。
2. GPIO又俗称为I/O口，I指的是输入(in），O指的是输出（out）。可以通过软件来控制其输入和输出，即I/O控制。
   - GPIO输入：输入是检测各个引脚上的电平状态，高电平或者低电平状态。
   - GPIO输出：输出是当需要控制引脚电平的高低时需要用到输出功能。
3. 在HDF框架中，同类型设备对象较多时（可能同时存在十几个同类型配置器），若采用独立服务模式，则需要配置更多的设备节点，且相关服务会占据更多的内存资源。
4. 相反，采用统一服务模式可以使用一个设备服务作为管理器，统一处理所有同类型对象的外部访问，实现便捷管理和节约资源的目的。

GPIO模块各分层作用

1. 接口层：提供操作GPIO管脚的标准方法。
2. 核心层：提供GPIO管脚资源匹配，GPIO管脚控制器的添加、移除以及管理的能力，通过钩子函数与适配层交互，供芯片厂家快速接入HDF框架。
3. 适配层：由驱动适配者将钩子函数的功能实例化，实现与硬件相关的具体功能。

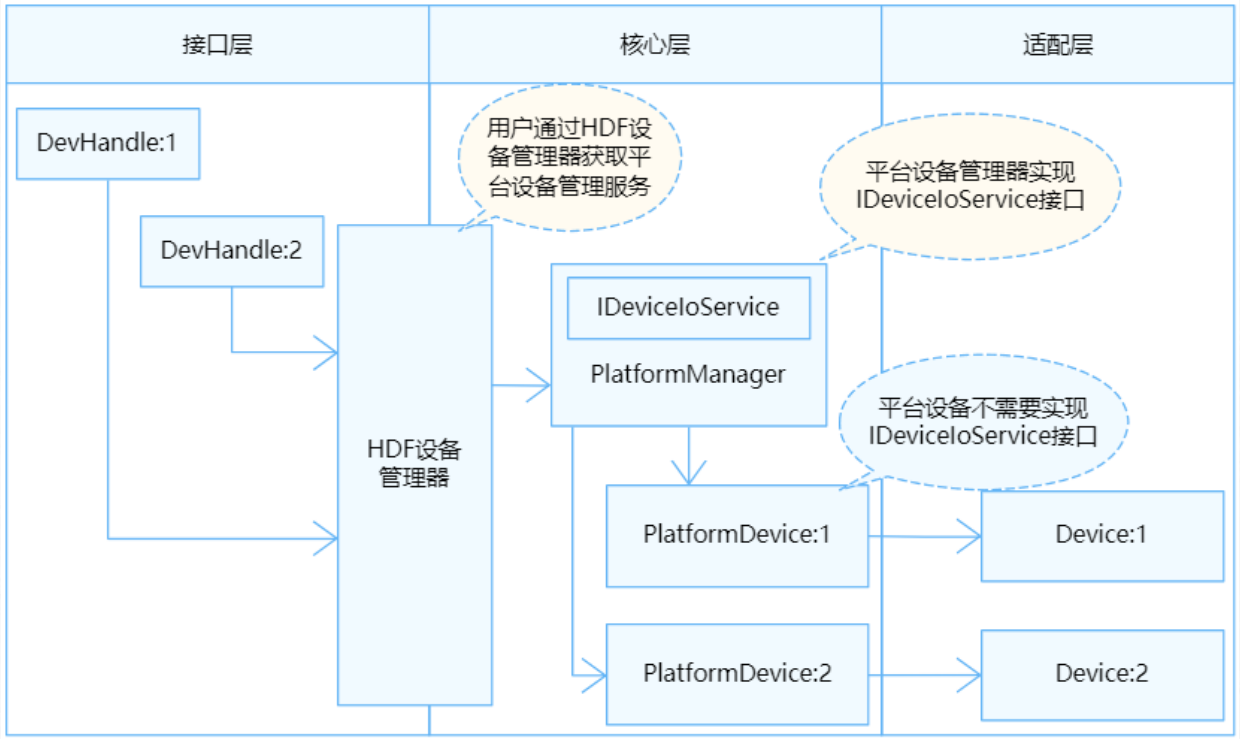


图9-4 GPIO统一服务模式结构图

GPIO使用流程

1. 确定GPIO管脚号
   - 可以根据SOC芯片规则进行计算通过管脚别名获取管脚号。
2. 设置GPIO管脚方向
   - 在进行GPIO管脚读写前，需要先通过GpioSetDir函数设置GPIO管脚方向。
3. 读写GPIO管脚电平值
   - 使用GpioRead()和GpioWrite()函数来对GPIO管脚写入电平值。
4. 设置GPIO管脚中断
   - 如果要为一个GPIO管脚设置中断响应程序，则可以通过调用GpioSetIrq()来设置对应的中断相应程序。
   - 同一时间，只能为某个GPIO管脚设置一个中断服务函数，如果重复调用GpioSetIrq()函数，则之前设置的中断服务函数会被取代。
5. 使能GPIO管脚中断
   - 在中断服务程序设置完成后，还需要先通过GpioEnableIrq()函数使能GPIO管脚的中断。

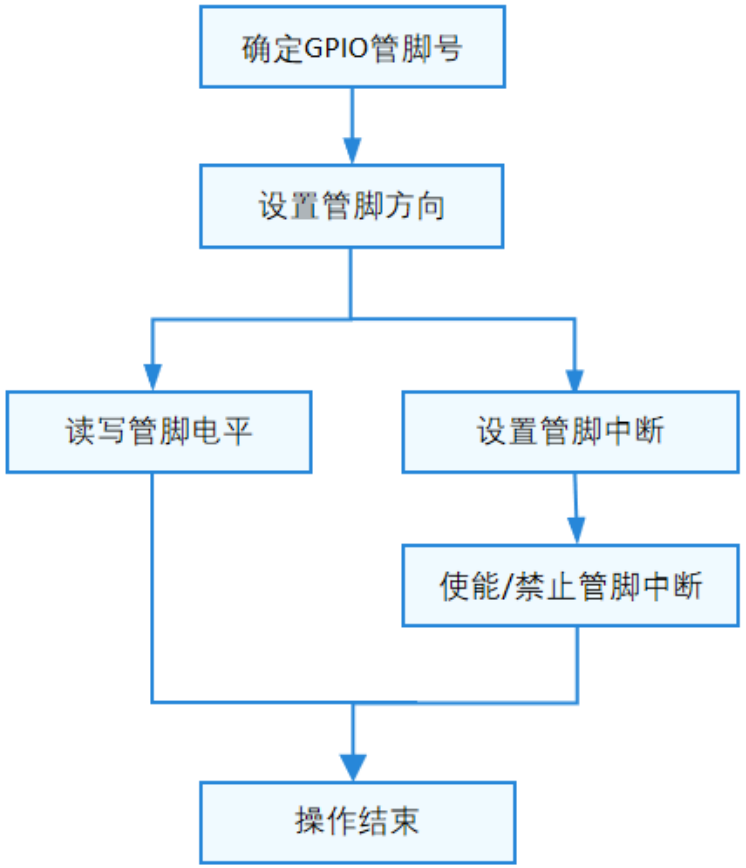


图9-5 GPIO使用流程图

设备驱动程序实例

利用GPIO点亮LED灯

1. 使用GPIO前，需要完成GPIO管脚初始化，明确管脚用途，并创建任务，使LED周期性亮灭，达到闪烁的效果。

```
1  static void LedExampleEntry(void)
2  {
3      osThreadAttr_t attr;
4
5      /* 管脚初始化 */
6      IoTGpioInit(LED_TEST_GPIO);
7      /* 配置9号管脚为输出方向 */
8      IoTGpioSetDir(LED_TEST_GPIO, IOT_GPIO_DIR_OUT);
9
10     attr.name = "LedTask";
11     attr.attr_bits = 0U;
12     attr.cb_mem = NULL;
13     attr.cb_size = 0U;
14     attr.stack_mem = NULL;
15     attr.stack_size = LED_TASK_STACK_SIZE;
16     attr.priority = LED_TASK_PRIO;
17
18     /* 启动任务 */
19     if (osThreadNew((osThreadFunc_t)LedTask, NULL, &attr) == NULL) {
20         printf("[LedExample] Failed to create LedTask!\n");
21     }
22 }
```

- **设备驱动程序实例**

  利用GPIO点亮LED灯

  1. 在循环任务中通过周期性亮灭形式实现LED闪烁。

```
1  static void *LedTask(const char *arg)
2  {
3      (void)arg;
4      while (1) {
5          switch (g_ledState) {
6              case LED_ON:
7                  IoTGpioSetOutputVal(LED_TEST_GPIO, 1);
8                  usleep(LED_INTERVAL_TIME_US);
9                  break;
10             case LED_OFF:
11                 IoTGpioSetOutputVal(LED_TEST_GPIO, 0);
12                 usleep(LED_INTERVAL_TIME_US);
13                 break;
14             case LED_SPARK:
15                 IoTGpioSetOutputVal(LED_TEST_GPIO, 0);
16                 usleep(LED_INTERVAL_TIME_US);
17                 IoTGpioSetOutputVal(LED_TEST_GPIO, 1);
18                 usleep(LED_INTERVAL_TIME_US);
19                 break;
20             default:
21                 usleep(LED_INTERVAL_TIME_US);
22                 break;
23         }
24     }
25 }
```

- **设备驱动程序实例**

  利用GPIO点亮LED灯

  1. 按下RST键复位模组，可发现LED在周期性闪烁。

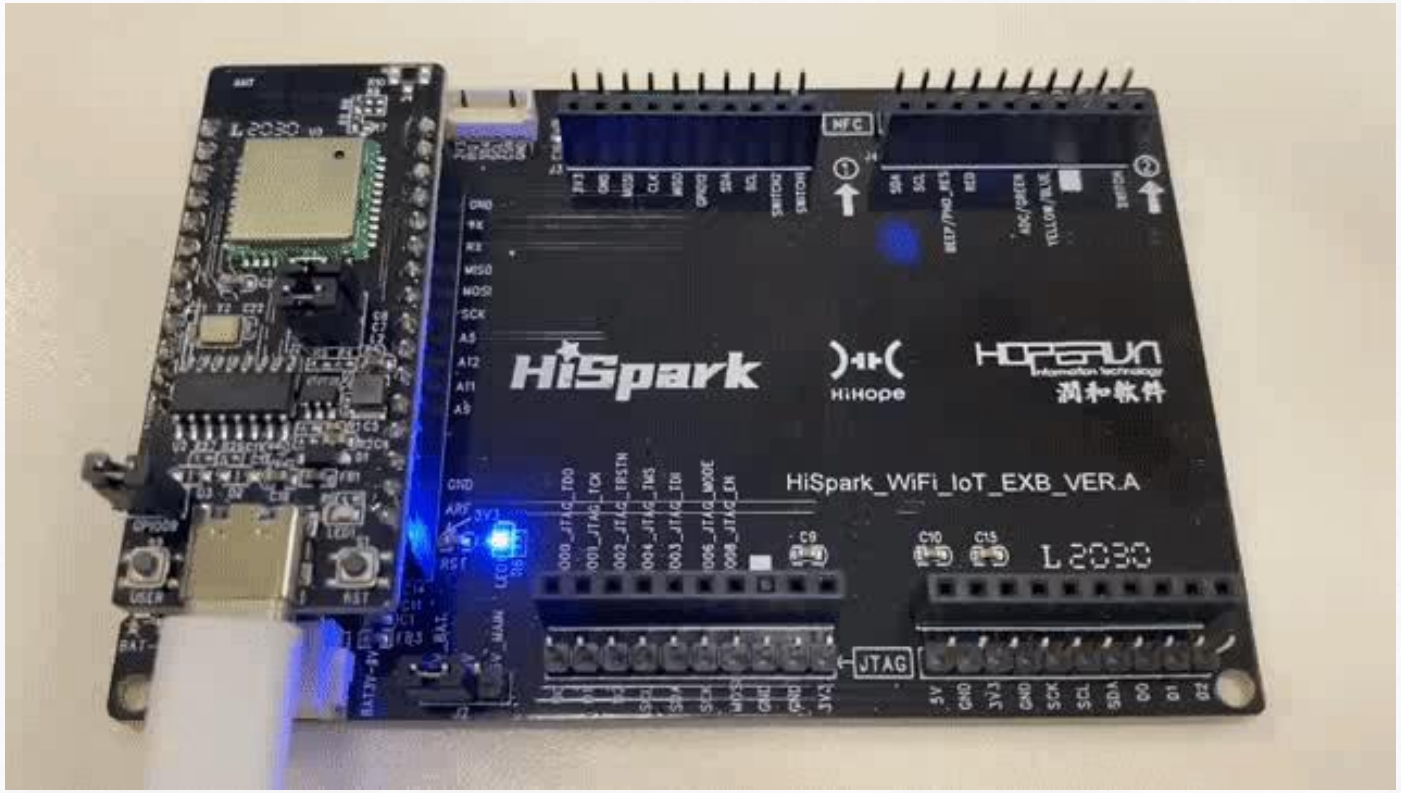


图9-6 GPIO点亮LED灯效果图

PWM

1. PWM（Pulse Width Modulation）即脉冲宽度调制，是一种对模拟信号电平进行数字编码并将其转换为脉冲的技术，广泛应用在从测量、通信到功率控制与变换的许多领域中。通常情况下，在使用马达控制、背光亮度调节时会用到PWM模块。

2. 在HDF框架中，PWM接口适配模式采用独立服务模式。在这种模式下，每一个设备对象会独立发布一个设备服务来处理外部访问，设备管理器收到API的访问请求之后，通过提取该请求的参数，达到调用实际设备对象的相应内部方法的目的。

- **PWM模块各分层作用**

1. 接口层：提供打开PWM设备、设置PWM设备参数、获取PWM设备参数、使能PWM设备、禁止PWM设备、关闭PWM设备的接口。
2. 核心层：主要提供PWM控制器的添加、移除以及管理的能力，通过钩子函数与适配层交互。
3. 适配层：主要是将钩子函数的功能实例化，实现具体的功能。

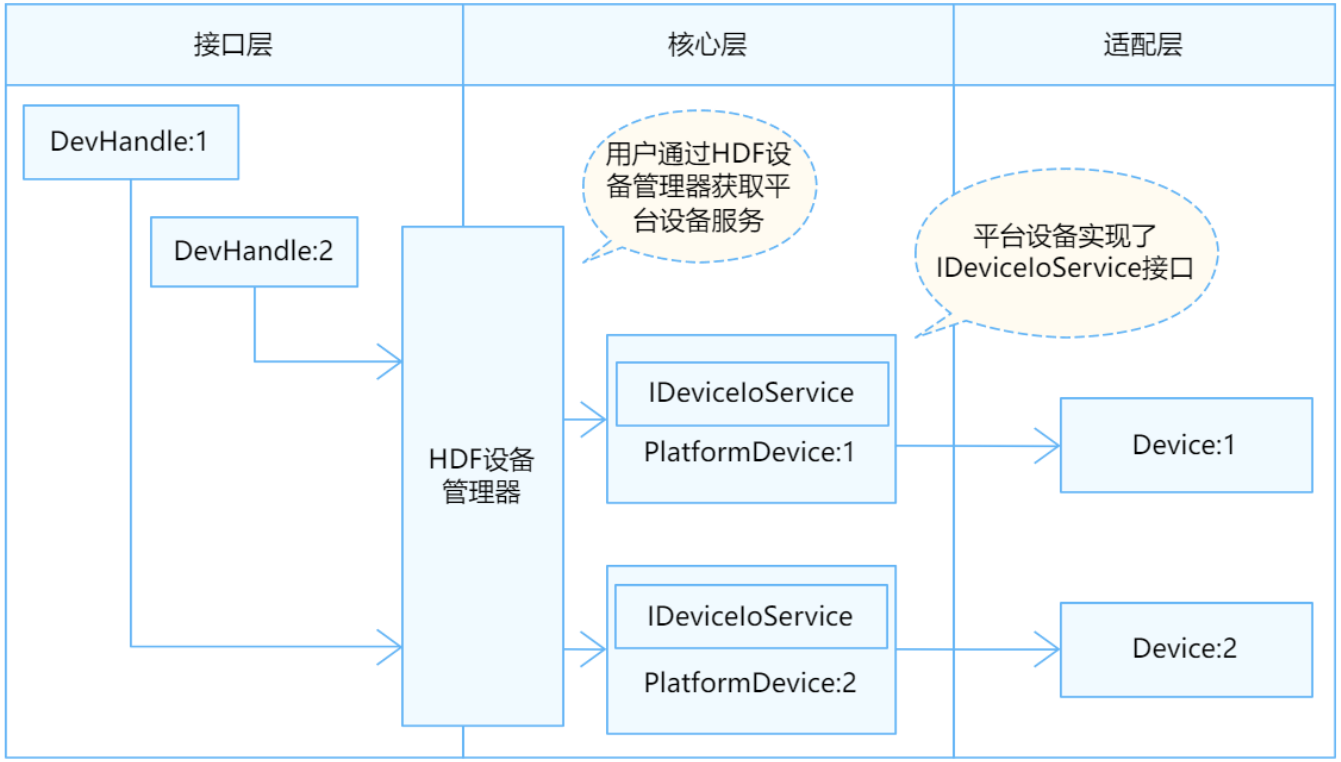


图9-7 PWM独立服务模式结构图

HDMI

1. HDMI（High Definition Multimedia Interface），即高清多媒体接口，主要用于DVD、机顶盒等音视频Source到TV、显示器等Sink设备的传输。
2. HDMI以主从方式工作，通常有一个Source端和一个Sink端。
3. HDMI接口定义了完成HDMI传输的通用方法集合。
   - HDMI控制器管理：打开或关闭HDMI控制器
   - HDMI启动/停止传输：启动或停止HDMI传输
   - HDMI控制器设置：设置音频、视频及HDR属性，设置色彩深度、声音图像消隐等
   - HDMI读取EDID：读取Sink端原始的EDID数据
   - HDMI热插拔：注册/注销热插拔回调函数
4. HDMI的传输过程遵循TMDS（Transition Minimized Differential Signaling）协议。TMDS即过渡调制差分信号，也被称为最小化传输差分信号，用于发送音频、视频及各种辅助数据。

HDMI模块各分层作用

1. 接口层：提供打开HDMI设备、启动HDMI传输、停止HDMI传输、声音图像消隐设置、设置色彩深度、获取色彩深度、设置视频属性、获取视频属性、设置HDR属性、读取Sink端原始EDID数据、注册HDMI热插拔检测回调函数、注销HDMI热插拔检测回调函数、关闭HDMI设备的接口。核心层：主要提供PWM控制器的添加、移除以及管理的能力，通过钩子函数与适配层交互。
2. 核心层：主要提供HDMI控制器的打开、关闭及管理的能力，通过钩子函数与适配层交互。
3. 适配层：主要是将钩子函数的功能实例化，实现具体的功能。

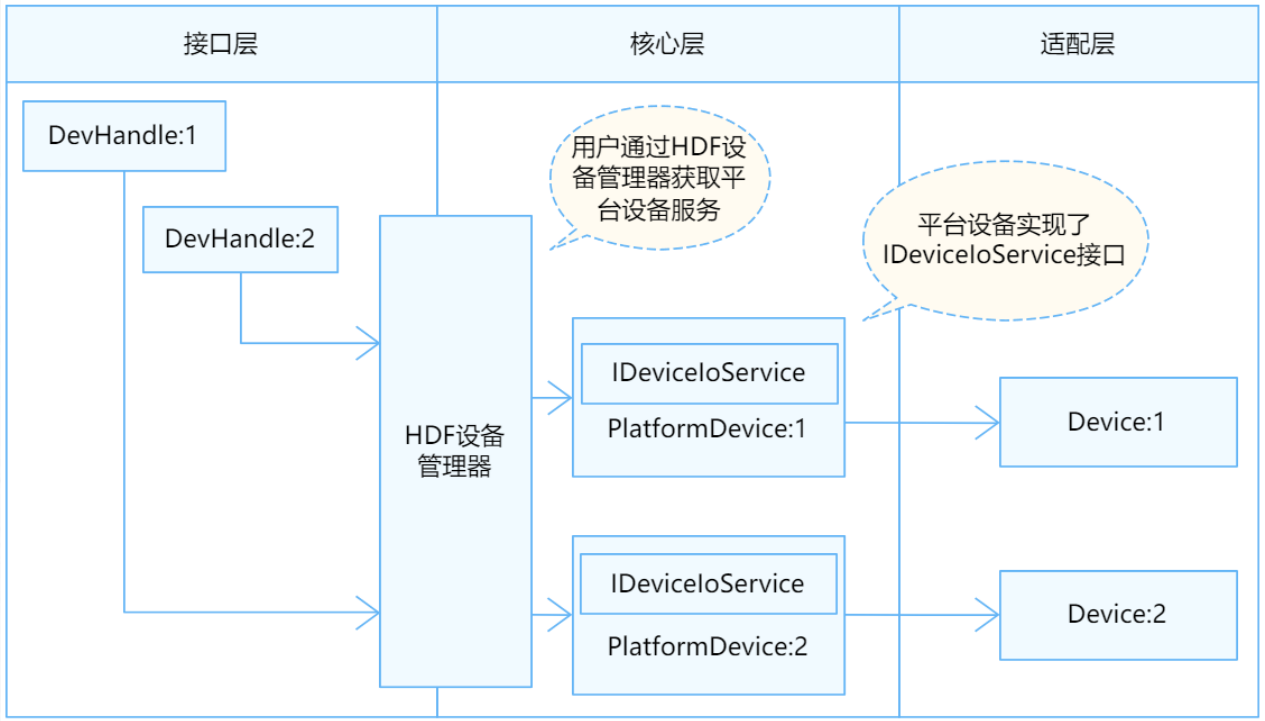


图9-7 HDMI独立服务模式结构图

中山大學
SUN YAT-SEN UNIVERSITY

谢谢观看

SUN YAT-SEN UNIVERSITY