开发Hi3516第一个驱动程序示例

本节指导开发者在单板上运行第一个驱动程序，其中包括驱动程序介绍、编译、烧写、运行等步骤。

获取源码

参考“开发Hi3516第一个应用程序示例”获取源码。

驱动程序介绍

下面基于HDF框架，提供一个简单的UART（Universal Asynchronous Receiver/Transmitter）平台驱动开发样例，包含配置文件的添加，驱动代码的实现以及用户态程序和驱动交互的流程。驱动程序源码位于vendor/huawei/hdf/sample目录。

1. 添加配置。

   在HDF框架的驱动配置文件（例如vendor/hisi/hi35xx/hi3516dv300/config/uart/uart_config.hcs）中添加该驱动的配置信息，如下所示：

```
root {
    platform {
        uart_sample {
            num = 5;             // UART设备编号
            base = 0x120a0000;   // UART 寄存器基地址
            irqNum = 38;
            baudrate = 115200;
            uartClk = 24000000;
            wlen = 0x60;
            parity = 0;
            stopBit = 0;
            match_attr = "sample_uart_5";
        }
    }
}
```

   在HDF框架的设备配置文件（例如vendor/hisi/hi35xx/hi3516dv300/config/device_info/device_info.hcs）中添加该驱动的设备节点信息，如下所示：

```
root {
    device_info {
        platform :: host {
            hostName = "platform_host";
            priority = 50;
            device_uart :: device {
                device5 :: deviceNode {
```

```
                    policy = 2;
                    priority = 10;
                    permission = 0660;
                    moduleName = "UART_SAMPLE";
                    serviceName = "HDF_PLATFORM_UART_5";
                    deviceMatchAttr = "sample_uart_5";
                }
            }
        }
    }
}
```

> 📖 **说明**：
> 配置文件与UART驱动示例的源码在同一个路径，需要手动添加到Hi3516DV300单板路径下。

2. 注册UART驱动入口。

   基于HDF框架注册UART驱动的入口HdfDriverEntry，代码如下：

```
// 绑定UART驱动接口到HDF框架
static int32_t SampleUartDriverBind(struct HdfDeviceObject *device)
{
    struct UartHost *uartHost = NULL;

    if (device == NULL) {
        return HDF_ERR_INVALID_OBJECT;
    }
    HDF_LOGI("Enter %s:", __func__);

    uartHost = UartHostCreate(device);
    if (uartHost == NULL) {
        HDF_LOGE("%s: UartHostCreate failed", __func__);
        return HDF_FAILURE;
    }
    uartHost->service.Dispatch = SampleDispatch;
    return HDF_SUCCESS;
}

// 从UART驱动的HCS中获取配置信息
static uint32_t GetUartDeviceResource(
    struct UartDevice *device, const struct DeviceResourceNode
*resourceNode)
{
    struct UartResource *resource = &device->resource;
    struct DeviceResourceIface *dri = NULL;
    dri = DeviceResourceGetIfaceInstance(HDF_CONFIG_SOURCE);
    if (dri == NULL || dri->GetUint32 == NULL) {
        HDF_LOGE("DeviceResourceIface is invalid");
        return HDF_FAILURE;
    }

    if (dri->GetUint32(resourceNode, "num", &resource->num, 0) !=
HDF_SUCCESS) {
        HDF_LOGE("uart config read num fail");
        return HDF_FAILURE;
    }
```

```
    if (dri->GetUint32(resourceNode, "base", &resource->base, 0) !=
HDF_SUCCESS) {
        HDF_LOGE("uart config read base fail");
        return HDF_FAILURE;
    }
    resource->physBase = (unsigned long)OsalIoRemap(resource->base, 0x48);
    if (resource->physBase == 0) {
        HDF_LOGE("uart config fail to remap physBase");
        return HDF_FAILURE;
    }
    if (dri->GetUint32(resourceNode, "irqNum", &resource->irqNum, 0) !=
HDF_SUCCESS) {
        HDF_LOGE("uart config read irqNum fail");
        return HDF_FAILURE;
    }
    if (dri->GetUint32(resourceNode, "baudrate", &resource->baudrate, 0) !=
HDF_SUCCESS) {
        HDF_LOGE("uart config read baudrate fail");
        return HDF_FAILURE;
    }
    if (dri->GetUint32(resourceNode, "wlen", &resource->wlen, 0) !=
HDF_SUCCESS) {
        HDF_LOGE("uart config read wlen fail");
        return HDF_FAILURE;
    }
    if (dri->GetUint32(resourceNode, "parity", &resource->parity, 0) !=
HDF_SUCCESS) {
        HDF_LOGE("uart config read parity fail");
        return HDF_FAILURE;
    }
    if (dri->GetUint32(resourceNode, "stopBit", &resource->stopBit, 0) !=
HDF_SUCCESS) {
        HDF_LOGE("uart config read stopBit fail");
        return HDF_FAILURE;
    }
    if (dri->GetUint32(resourceNode, "uartClk", &resource->uartClk, 0) !=
HDF_SUCCESS) {
        HDF_LOGE("uart config read uartClk fail");
        return HDF_FAILURE;
    }
    return HDF_SUCCESS;
}

// 将UART驱动的配置和接口附加到HDF驱动框架
static int32_t AttachUartDevice(struct UartHost *host, struct
HdfDeviceObject *device)
{
    int32_t ret;
    struct UartDevice *uartDevice = NULL;
    if (device->property == NULL) {
        HDF_LOGE("%s: property is NULL", __func__);
        return HDF_FAILURE;
    }
    uartDevice = (struct UartDevice *)OsalMemCalloc(sizeof(struct
UartDevice));
    if (uartDevice == NULL) {
        HDF_LOGE("%s: OsalMemCalloc uartDevice error", __func__);
        return HDF_ERR_MALLOC_FAIL;
```

```
    }
    ret = GetUartDeviceResource(uartDevice, device->property);
    if (ret != HDF_SUCCESS) {
        (void)OsalMemFree(uartDevice);
        return HDF_FAILURE;
    }
    host->num = uartDevice->resource.num;
    host->priv = uartDevice;
    AddUartDevice(host);
    return InitUartDevice(uartDevice);
}

// 初始化UART驱动
static int32_t SampleUartDriverInit(struct HdfDeviceObject *device)
{
    int32_t ret;
    struct UartHost *host = NULL;

    if (device == NULL) {
        HDF_LOGE("%s: device is NULL", __func__);
        return HDF_ERR_INVALID_OBJECT;
    }
    HDF_LOGI("Enter %s:", __func__);
    host = UartHostFromDevice(device);
    if (host == NULL) {
        HDF_LOGE("%s: host is NULL", __func__);
        return HDF_FAILURE;
    }
    ret = AttachUartDevice(host, device);
    if (ret != HDF_SUCCESS) {
        HDF_LOGE("%s: attach error", __func__);
        return HDF_FAILURE;
    }
    host->method = &g_sampleUartHostMethod;
    return ret;
}

static void DeinitUartDevice(struct UartDevice *device)
{
    struct UartRegisterMap *regMap = (struct UartRegisterMap *)device-
>resource.physBase;
    /* wait for uart enter idle. */
    while (UartPl011IsBusy(regMap));
    UartPl011ResetRegisters(regMap);
    uart_clk_cfg(0, false);
    OsalIoUnmap((void *)device->resource.physBase);
    device->state = UART_DEVICE_UNINITIALIZED;
}

// 解绑并释放UART驱动
static void DetachUartDevice(struct UartHost *host)
{
    struct UartDevice *uartDevice = NULL;

    if (host->priv == NULL) {
        HDF_LOGE("%s: invalid parameter", __func__);
        return;
    }
```

```
    uartDevice = host->priv;
    DeinitUartDevice(uartDevice);
    (void)OsalMemFree(uartDevice);
    host->priv = NULL;
}

// 释放UART驱动
static void SampleUartDriverRelease(struct HdfDeviceObject *device)
{
    struct UartHost *host = NULL;
    HDF_LOGI("Enter %s:", __func__);

    if (device == NULL) {
        HDF_LOGE("%s: device is NULL", __func__);
        return;
    }
    host = UartHostFromDevice(device);
    if (host == NULL) {
        HDF_LOGE("%s: host is NULL", __func__);
        return;
    }
    if (host->priv != NULL) {
        DetachUartDevice(host);
    }
    UartHostDestroy(host);
}

struct HdfDriverEntry g_sampleUartDriverEntry = {
    .moduleVersion = 1,
    .moduleName = "UART_SAMPLE",
    .Bind = SampleUartDriverBind,
    .Init = SampleUartDriverInit,
    .Release = SampleUartDriverRelease,
};

HDF_INIT(g_sampleUartDriverEntry);
```

3. 注册UART驱动接口。

   HDF框架提供了UART驱动接口的模板方法UartHostMethod，实现UART驱动接口的代码如下：

```
static int32_t SampleUartHostInit(struct UartHost *host)
{
    HDF_LOGI("%s: Enter", __func__);
    if (host == NULL) {
        HDF_LOGE("%s: invalid parameter", __func__);
        return HDF_ERR_INVALID_PARAM;
    }
    return HDF_SUCCESS;
}

static int32_t SampleUartHostDeinit(struct UartHost *host)
{
    HDF_LOGI("%s: Enter", __func__);
    if (host == NULL) {
        HDF_LOGE("%s: invalid parameter", __func__);
        return HDF_ERR_INVALID_PARAM;
```

```
    }
    return HDF_SUCCESS;
}

// 向UART中写入数据
static int32_t SampleUartHostWrite(struct UartHost *host, uint8_t *data,
uint32_t size)
{
    HDF_LOGI("%s: Enter", __func__);
    uint32_t idx;
    struct UartRegisterMap *regMap = NULL;
    struct UartDevice *device = NULL;

    if (host == NULL || data == NULL || size == 0) {
        HDF_LOGE("%s: invalid parameter", __func__);
        return HDF_ERR_INVALID_PARAM;
    }
    device = (struct UartDevice *)host->priv;
    if (device == NULL) {
        HDF_LOGE("%s: device is NULL", __func__);
        return HDF_ERR_INVALID_PARAM;
    }
    regMap = (struct UartRegisterMap *)device->resource.physBase;
    for (idx = 0; idx < size; idx++) {
        UartPl011Write(regMap, data[idx]);
    }
    return HDF_SUCCESS;
}

// 设置UART的波特率
static int32_t SampleUartHostSetBaud(struct UartHost *host, uint32_t
baudRate)
{
    HDF_LOGI("%s: Enter", __func__);
    struct UartDevice *device = NULL;
    struct UartRegisterMap *regMap = NULL;
    UartPl011Error err;

    if (host == NULL) {
        HDF_LOGE("%s: invalid parameter", __func__);
        return HDF_ERR_INVALID_PARAM;
    }
    device = (struct UartDevice *)host->priv;
    if (device == NULL) {
        HDF_LOGE("%s: device is NULL", __func__);
        return HDF_ERR_INVALID_PARAM;
    }
    regMap = (struct UartRegisterMap *)device->resource.physBase;
    if (device->state != UART_DEVICE_INITIALIZED) {
        return UART_PL011_ERR_NOT_INIT;
    }
    if (baudRate == 0) {
        return UART_PL011_ERR_INVALID_BAUD;
    }
    err = UartPl011SetBaudrate(regMap, device->uartClk, baudRate);
    if (err == UART_PL011_ERR_NONE) {
        device->baudrate = baudRate;
    }
```

```
    return err;
}

// 获取UART的波特率
static int32_t SampleUartHostGetBaud(struct UartHost *host, uint32_t
*baudRate)
{
    HDF_LOGI("%s: Enter", __func__);
    struct UartDevice *device = NULL;

    if (host == NULL) {
        HDF_LOGE("%s: invalid parameter", __func__);
        return HDF_ERR_INVALID_PARAM;
    }
    device = (struct UartDevice *)host->priv;
    if (device == NULL) {
        HDF_LOGE("%s: device is NULL", __func__);
        return HDF_ERR_INVALID_PARAM;
    }
    *baudRate = device->baudrate;
    return HDF_SUCCESS;
}

// 在HdfUartSampleInit方法中绑定
struct UartHostMethod g_sampleUartHostMethod = {
    .Init = SampleUartHostInit,
    .Deinit = SampleUartHostDeinit,
    .Read = NULL,
    .Write = SampleUartHostWrite,
    .SetBaud = SampleUartHostSetBaud,
    .GetBaud = SampleUartHostGetBaud,
    .SetAttribute = NULL,
    .GetAttribute = NULL,
    .SetTransMode = NULL,
};
```

在vendor/huawei/hdf/hdf_vendor.mk编译脚本中增加示例UART驱动模块，代码如下：

```
LITEOS_BASELIB += -lhdf_uart_sample
LIB_SUBDIRS    += $(VENDOR_HDF_DRIVERS_ROOT)/sample/platform/uart
```

4. 用户程序和驱动交互代码。

   UART驱动成功初始化后，会创建/dev/uartdev-5设备节点，通过设备节点与UART驱动交互的代码如下：

```
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include "hdf_log.h"

#define HDF_LOG_TAG "hello_uart"
#define INFO_SIZE 16

int main(void)
{
    int ret;
```

```
    int fd;
    const char info[INFO_SIZE] = {" HELLO UART! "};

    fd = open("/dev/uartdev-5", O_RDWR);
    if (fd < 0) {
        HDF_LOGE("hello_uart uartdev-5 open failed %d", fd);
        return -1;
    }
    ret = write(fd, info, INFO_SIZE);
    if (ret != 0) {
        HDF_LOGE("hello_uart write uartdev-5 ret is %d", ret);
    }
    ret = close(fd);
    if (ret != 0) {
        HDF_LOGE("hello_uart uartdev-5 close failed %d", fd);
        return -1;
    }
    return ret;
}
```

在build/lite/product/ipcamera_hi3516dv300.json产品配置的hdf子系统下增加hello_uart_sample组件，代码如下：

```
{
  "subsystem": [
    {
      "name": "hdf",
      "component": [
        { "name": "hdf_sample", "dir":
"//vendor/huawei/hdf/sample/platform/uart:hello_uart_sample", "features":[]
}
      ]
    }
  ]
}
```

> 📖 **说明：**
> 如上代码均为示例代码，完整代码可以在vendor/huawei/hdf/sample查看。
> 示例代码默认不参与编译，需要手动添加到编译脚本中。

编译和烧写

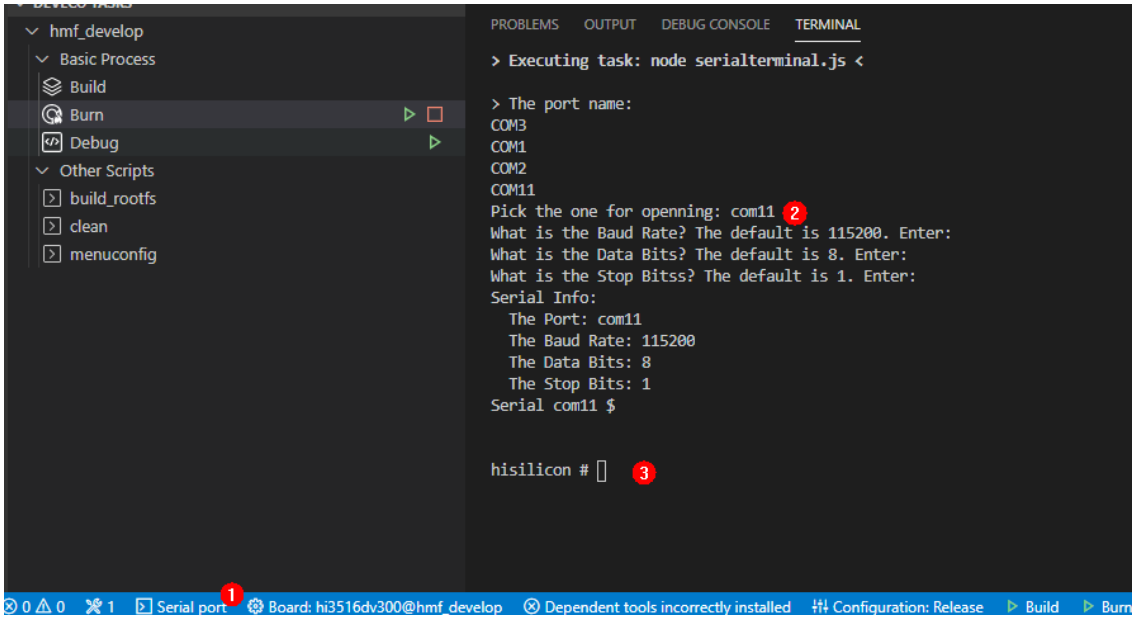
参考示例1进行编译和烧写：编译、烧录。

镜像运行

1. 连接串口。

> ⚠ **须知：**
> 若无法连接串口，请参考常见问题进行排查。

图 1 连接串口图



1. 单击**Serial port**打开串口。
2. 输入串口编号(按照烧录步骤中查询的串口号，此处以com11举例)，并连续输入回车直到串口显示"hisillicon"。
3. 单板初次启动或修改启动参数，请进入步骤2，否则进入步骤3。

2. （单板初次启动必选）修改U-boot的bootcmd及bootargs内容：该步骤为固化操作，若不修改参数只需执行一次。每次复位单板均会自动进入系统。

> **须知：**
> U-boot引导程序默认会有2秒的等待时间，用户可使用回车打断等待并显示"hisillicon"，通过**reset**命令可再次启动系统。

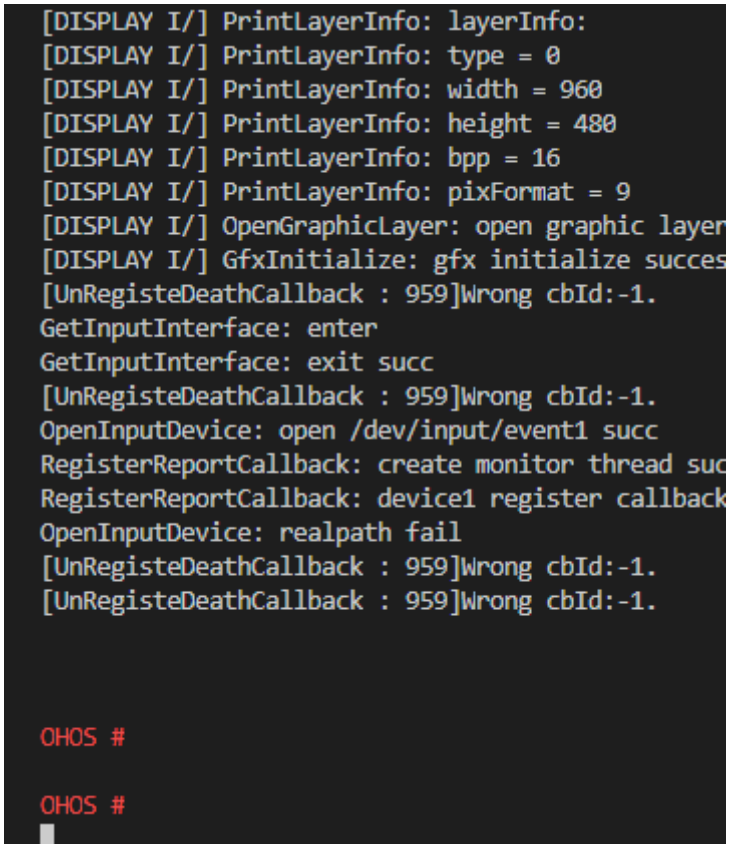
表 1 U-boot修改命令

执行命令	命令解释
setenv bootcmd "mmc read 0x0 0x80000000 0x800 0x4800; go 0x80000000";	读取FLASH起始地址为0x800（单位为512B，即1MB），大小为0x4800（单位为512B，即9MB）的内容到0x80000000的内存地址，该大小（9MB）与IDE中所填写OHOS_Image.bin文件大小**必须相同。**
setenv bootargs "console=ttyAMA0,115200n8 root=emmc fstype=vfat rootaddr=10M rootsize=15M rw";	表示设置启动参数，输出模式为串口输出，波特率为115200，数据位8，rootfs挂载于emmc器件，文件系统类型为vfat， “rootaddr=10M rootsize=15M rw”处对应填入rootfs.img的烧写起始位置与长度，此处与IDE中新增rootfs.img文件时所填大小**必须相同。**
saveenv	表示保存当前配置。
reset	表示复位单板。

> **须知：**
> **“go 0x80000000”**为可选指令，默认配置已将该指令固化在启动参数中，单板复位后可自动启动。若想切换为手动启动，可在U-boot启动倒数阶段使用"回车"打断自动启动。

3. 输入**“reset”**指令并回车，重启单板，启动成功如下图，输入回车串口显示OHOS字样。

图 2 系统启动图



4. 根目录下，在命令行输入指令“**./bin/hello_uart**”执行写入的demo程序，显示成功结果如下所示。

```
OHOS # ./bin/hello_uart
OHOS #  HELLO UART!
```

下一步学习

恭喜，您已完成Hi3516 快速上手！建议您下一步进入带屏摄像头产品开发的学习 。