整理：电子发烧友 - 绿波电龙

作者：疯壳

注：文档和视频中所有的图片及代码截图皆为示意图，具体以 HarmonyOS 官网发布内容为准。

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。

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 = 0644;
                    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 HdfUartSampleBind(struct HdfDeviceObject *device)
{
    if (device == NULL) {
        return HDF_ERR_INVALID_OBJECT;
    }
    HDF_LOGI("Enter %s:", __func__);
    return (UartHostCreate(device) == NULL) ? HDF_FAILURE : HDF_SUCCESS;
}
```

```
// 从 UART 驱动的 HCS 中获取配置信息
static uint32_t UartDeviceGetResource(
    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 SampleAttach(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 = UartDeviceGetResource(uartDevice, device->property);
    if (ret != HDF_SUCCESS) {
        (void) OsalMemFree(uartDevice);
        return HDF_FAILURE;
    }
    host->num = uartDevice->resource.num;
    host->priv = uartDevice;
    UartSampleAddDev(host); // 添加用户态 UART 设备节点，具体实现见源码 uart_dev_sample
    return UartDeviceInit(uartDevice); // 初始化 UART PL011，具体实现见源码 uart_pl011_sample
}

// 初始化 UART 驱动
static int32_t HdfUartSampleInit(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 = SampleAttach(host, device);
    if (ret != HDF_SUCCESS) {
        HDF_LOGE("%s: attach error", __func__);
        return HDF_FAILURE;
    }
    host->method = &g_uartSampleHostMethod;
    return ret;
}

static void UartDeviceDeinit(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 SampleDetach(struct UartHost *host)
{
    struct UartDevice *uartDevice = NULL;

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

// 释放 UART 驱动
static void HdfUartSampleRelease(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) {
        SampleDetach(host);
    }
    UartHostDestroy(host);
}

struct HdfDriverEntry g_hdfUartSample = {
    .moduleVersion = 1,
    .moduleName = "UART_SAMPLE",
    .Bind = HdfUartSampleBind,
    .Init = HdfUartSampleInit,
    .Release = HdfUartSampleRelease,
};

HDF_INIT(g_hdfUartSample);
```

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：

```
1. static int32_t SampleInit(struct UartHost *host)
2. {
3.     HDF_LOGI("%s: Enter", __func__);
4.     if (host == NULL) {
```

```
5.          HDF_LOGE("%s: invalid parameter", __func__);
6.          return HDF_ERR_INVALID_PARAM;
7.      }
8.      return HDF_SUCCESS;
9. }
10.
11.   static int32_t SampleDeinit(struct UartHost *host)
12.   {
13.       HDF_LOGI("%s: Enter", __func__);
14.       if (host == NULL) {
15.           HDF_LOGE("%s: invalid parameter",
   __func__);
16.           return HDF_ERR_INVALID_PARAM;
17.       }
18.       return HDF_SUCCESS;
19.   }
20.
21.   // 向 UART 中写入数据
22.   static int32_t SampleWrite(struct UartHost *host,
   uint8_t *data, uint32_t size)
23.   {
24.       HDF_LOGI("%s: Enter", __func__);
25.       uint32_t idx;
26.       struct UartRegisterMap *regMap = NULL;
27.       struct UartDevice *device = NULL;
28.
29.       if (host == NULL || data == NULL || size == 0)
   {
30.           HDF_LOGE("%s: invalid parameter",
   __func__);
31.           return HDF_ERR_INVALID_PARAM;
32.       }
33.       device = (struct UartDevice *) host->priv;
34.       if (device == NULL) {
35.           HDF_LOGE("%s: device is NULL", __func__);
36.           return HDF_ERR_INVALID_PARAM;
37.       }
```

```
38.       regMap = (struct UartRegisterMap *)
  device->resource.physBase;
39.       for (idx = 0; idx < size; idx++) {
40.           while (UartPl011IsBusy(regMap));
41.           UartPl011Write(regMap, data[idx]);
42.       }
43.       return HDF_SUCCESS;
44.   }
45.
46.   // 设置 UART 的波特率
47.   static int32_t SampleSetBaud(struct UartHost *host,
  uint32_t baudRate)
48.   {
49.       HDF_LOGI("%s: Enter", __func__);
50.       struct UartDevice *device = NULL;
51.       struct UartRegisterMap *regMap = NULL;
52.       UartPl011Error err;
53.
54.       if (host == NULL) {
55.           HDF_LOGE("%s: invalid parameter",
  __func__);
56.           return HDF_ERR_INVALID_PARAM;
57.       }
58.       device = (struct UartDevice *) host->priv;
59.       if (device == NULL) {
60.           HDF_LOGE("%s: device is NULL", __func__);
61.           return HDF_ERR_INVALID_PARAM;
62.       }
63.       regMap = (struct UartRegisterMap *)
  device->resource.physBase;
64.       if (device->state != UART_DEVICE_INITIALIZED) {
65.           return UART_PL011_ERR_NOT_INIT;
66.       }
67.       if (baudRate == 0) {
68.           return UART_PL011_ERR_INVALID_BAUD;
69.       }
70.       err = UartPl011SetBaudrate(regMap,
  device->uartClk, baudRate);
```

```
71.       if (err == UART_PL011_ERR_NONE) {
72.           device->baudrate = baudRate;
73.       }
74.       return err;
75.   }
76.
77.   // 获取 UART 的波特率
78.   static int32_t SampleGetBaud(struct UartHost *host,
  uint32_t *baudRate)
79.   {
80.       HDF_LOGI("%s: Enter", __func__);
81.       struct UartDevice *device = NULL;
82.
83.       if (host == NULL) {
84.           HDF_LOGE("%s: invalid parameter",
  __func__);
85.           return HDF_ERR_INVALID_PARAM;
86.       }
87.       device = (struct UartDevice *) host->priv;
88.       if (device == NULL) {
89.           HDF_LOGE("%s: device is NULL", __func__);
90.           return HDF_ERR_INVALID_PARAM;
91.       }
92.       *baudRate = device->baudrate;
93.       return HDF_SUCCESS;
94.   }
95.
96.   // 在 HdfUartSampleInit 方法中绑定
97.   struct UartHostMethod g_uartSampleHostMethod = {
98.       .Init = SampleInit,
99.       .Deinit = SampleDeinit,
100.      .Read = NULL,
101.      .Write = SampleWrite,
102.      .SetBaud = SampleSetBaud,
103.      .GetBaud = SampleGetBaud,
104.      .SetAttribute = NULL,
105.      .GetAttribute = NULL,
106.      .SetTransMode = NULL,
```

```
107.  };
```

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：

```
1. LITEOS_BASELIB += -lhdf_uart_sample
2. 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：

```
1. #include <stdlib.h>
2. #include <unistd.h>
3. #include <fcntl.h>
4. #include "hdf_log.h"
5.
6. #define HDF_LOG_TAG "hello_uart"
7. #define INFO_SIZE 16
8.
9. int main(void)
10.   {
11.       int ret;
12.       int fd;
13.       const char info[INFO_SIZE] = {" HELLO UART! "};
14.
15.       fd = open("/dev/uartdev-5", O_RDWR);
16.       if (fd < 0) {
17.           HDF_LOGE("hello_uart uartdev-5 open
   failed %d", fd);
18.           return -1;
19.       }
20.       ret = write(fd, info, INFO_SIZE);
21.       if (ret != 0) {
22.           HDF_LOGE("hello_uart write uartdev-5 ret
   is %d", ret);
23.       }
24.       ret = close(fd);
25.       if (ret != 0) {
```

```
26.          HDF_LOGE("hello_uart uartdev-5 close
   failed %d", fd);
27.          return -1;
28.      }
29.      return ret;
30.  }
```

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：

```
  {

2.  "subsystem": [
3.    {
4.      "name": "hdf",
5.      "component": [
6.        { "name": "hdf_sample", "dir":
   "//vendor/huawei/hdf/sample/platform/uart:hello_uart
   _sample", "features":[] }
7.      ]
8.    }
9.  ]

  }
```

如上代码均为示例代码，完整代码可以在 vendor/huawei/hdf/sample 查看。

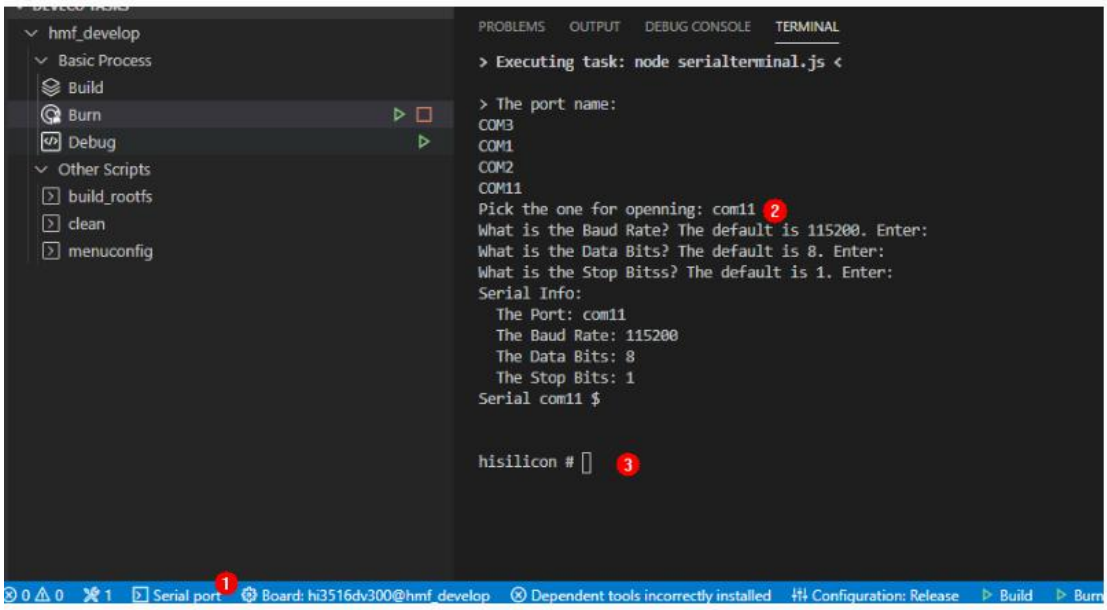
示例代码默认不参与编译，需要手动添加到编译脚本中。

编译和烧写

镜像运行

1. 连接串口。

图 1 连接串口图



1. 单击 **Serial port** 打开串口。
2. 输入"com11"串口编号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。

（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。
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启动参数

执行命令	**setenv bootcmd "sf probe 0;mmc read 0x0 0x80000000 0x800 0x4800; go 0x80000000";** **setenv bootargs "console=ttyAMA0,115200n8 root=emmc fstype=vfat rootaddr=10M rootsize=15M rw";** **saveenv** **reset**
命令解释	**setenv bootcmd "mmc read 0x0 0x80000000 0x800 0x4800;go 0x80000000";** 表示选择FLASH器件0，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。 **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 **[可选]“go 0x80000000”** 默认配置已将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。

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。

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

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