密级状态：绝密（　　）　秘密（　　）　内部（ ）　公开（√ ）

RK3399_VR 分体机_软件开发指南
V1.0_20160903

（技术部，系统产品二部）

文件状态： [ ] 正在修改 [√] 正式发布	当前版本：	V1.1
	作　　者：	张文平/王剑辉
	完成日期：	2017-01-04
	审　　核：	
	完成日期：	

福州瑞芯微电子股份有限公司

Fuzhou Rockchips Semiconductor Co . , Ltd

更新记录

版本	修改人	修改日期	修改说明	备注
V1.0	张文平	2016.09.03	初版	
V1.1	张文平/王剑辉	2017.01.04	1.根据代码变更，更新配置说明 2.添加许多配置说明，供客户参考	

目 录

1 概述 2

2 RK3399 端配置说明 3

2.1 ANDROID 配置 3

2.2 KERNEL 配置 3

2.2.1 DTS 设备树选择 3

2.2.2 Typec 口配置 3

2.2.3 显示相关配置 6

2.2.4 Sensor 相关配置 14

2.3 编译 20

2.3.1 ANDROID 端编译说明 20

2.3.2 NANOC 端编译说明 21

3 NANOC 端各个模块说明 21

3.1 按键修改说明 21

3.2 LCD 显示修改说明 21

3.3 SENSOR 修改说明文档 21

3.4 NANOC 端工具说明文档 22

1 概述

本文档只对 VR 分体机相关的配置和修改进行说明，其余公共部分文档请参考《Rockchip RK3399 软件开发指南 V1.00-20160901.pdf》。

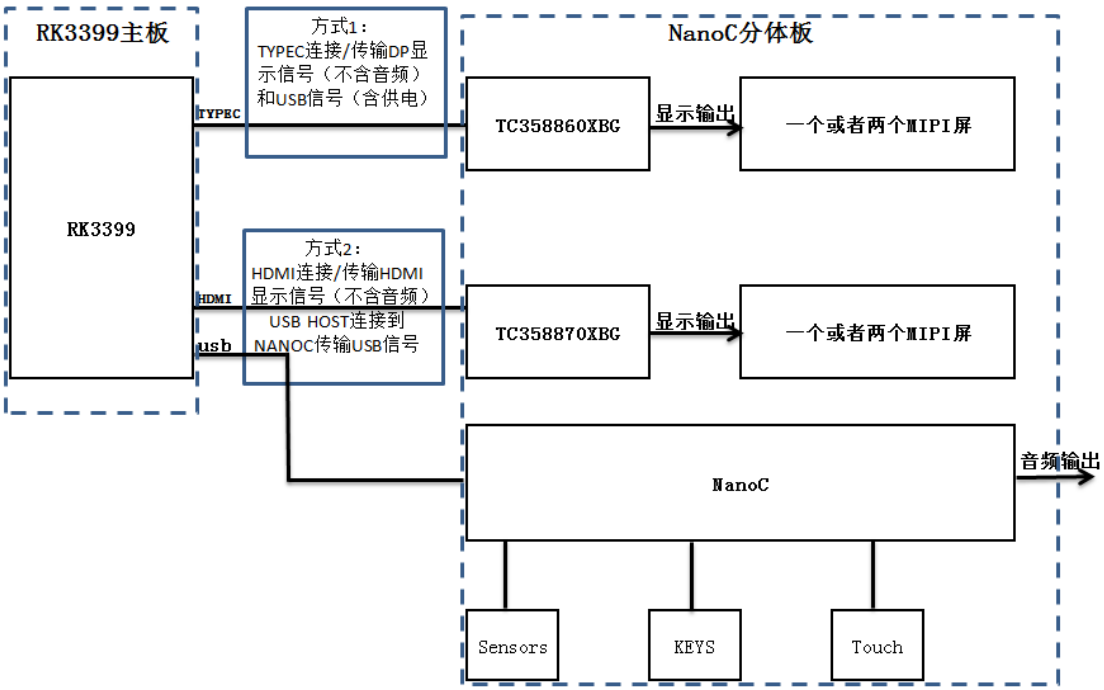


图 1 VR 分体机框图

图 1 描述的是分体机的整体框图，RK3399 和分体设备相连有两种方式：

- Typec 接口

  Typec 口支持同时传输 dp 信号和 usb 信号，dp 信号经过 nanoc 的 TC358860 芯片转换成 mipi 信号输出给屏。USB 信号负责将 NanoC 端的 sensor/key/touch 数据传输给 rk3399 端，并且将 rk3399 端的音频数据传输给 NanoC 端输出给耳机或者喇叭。

- Hdmi + USB

  HDMI 接口负责传输 hdmi 信号给 Nanoc 端的 TC358870，并且由 TC358870 将 hdmi 信号转换为 mipi 信号输出给屏。RK3399 端的 usb host 口则负责传输 usb 信号，usb 信号传输的内容包括 sensor/key/touch/audio。

2 RK3399 端配置说明

2.1 Android 配置

针对分体式 VR，Android 端需要配置的参数主要位于 device/rockchip/rk3399/rk3399_disvr.mk 文件中，请在配置时查看这个文件，并且参考文档《RK3399 VR Android 参数配置和调试说明》进行配置。

【注】如果根据下一节的 kernel 端配置正确后能够显示，但是显示方向不对，请参考根据本节所列文档进行调整。

2.2 Kernel 配置

2.2.1 DTS 设备树选择

目前有两种版本的硬件，主要差别在于 PMU 是使用 RK818 还是 RK808，因为两种硬件使用的 dts 配置文件不同，dts 选择的原则如下：

1. 基于 RK3399 VR 一体机或者平板的参考电路设计的硬件，请参考使用下述 dts：

   arch/arm64/boot/dts/rockchip/rk3399-disvr-android.dts

   该设计参考使用的 pmu 为 RK818，请在此 dts 基础上修改具体的外设节点。

2. 基于 RK3399 Box 的参考电路设计的硬件，请参考使用下述 dts：

   arch/arm64/boot/dts/rockchip/rk3399-box-rev1-disvr.dts

   该设计参考使用的 pmu 为 RK808，请在此 dts 基础上修改具体的外设节点。

2.2.2 Typec 口配置

由于硬件和模具的设计不同，可能客户需要两个 typec 端口或者只需要一个 typec 端口，根据不同的要求，需要做一些特别的配置，请根据下述说明在项目的 dts 文件中确认相关节点是否添加：

2.2.2.1 typec0 和 typec1 口均可输出 dp 显示信号

1） 在对应的 i2c 口上定义 fusb302 节点，因为有两个 typec 口，所以需要两个 fusb302，如下所示（注：下面只列出了 fusb0 的对应写法，fusb1 类似）：

```
&i2c4 {
    status = "okay";
    i2c-scl-rising-time-ns = <345>;
    i2c-scl-falling-time-ns = <11>;
    clock-frequency = <400000>;

    fusb0: fusb30x@22 {
        compatible = "fairchild,fusb302";
        reg = <0x22>;
        pinctrl-names = "default";
        pinctrl-0 = <&fusb0_int>;
        int-n-gpios = <&gpio1 2 GPIO_ACTIVE_HIGH>;
        status = "okay";
    };
};
```

2） 使能 typec phy0 和 phy1，如下所示：

```
&tcphy0 {
    status = "okay";
    extcon = <&fusb0>;
};
&tcphy1 {
    status = "okay";
    extcon = <&fusb1>;
};
```

3） 使能 cdn_dp 节点，如下所示：

```
&cdn_dp_fb {
    status = "okay";
    extcon = <&fusb0>, <&fusb1>;
    dp_vop_sel = <DISPLAY_SOURCE_LCDC0>;
    dp_defaultmode = <0>;
};
```

2.2.2.2 只从 typec0 口输出 dp 显示信号

1) 在对应的 i2c 口上定义 fusb302 节点，如下所示：

```
&i2c4 {
    status = "okay";
    i2c-scl-rising-time-ns = <345>;
    i2c-scl-falling-time-ns = <11>;
    clock-frequency = <400000>;

    fusb0: fusb30x@22 {
        compatible = "fairchild,fusb302";
        reg = <0x22>;
        pinctrl-names = "default";
        pinctrl-0 = <&fusb0_int>;
        int-n-gpios = <&gpio1 2 GPIO_ACTIVE_HIGH>;
        status = "okay";
    };
};
```

2) 使能 typec phy0，如下所示：

```
&tcphy0 {
    status = "okay";
    extcon = <&fusb0>;
};
```

3) 使能 cdn_dp 节点，如下所示：

```
&cdn_dp_fb {
    status = "okay";
    extcon = <&fusb0>;
    dp_vop_sel = <DISPLAY_SOURCE_LCDC0>;
    dp_defaultmode = <0>;
    phys = <&tcphy0_dp>;
};
```

2.2.2.3 只从 typec1 口输出 dp 显示信号

1) 在对应的 i2c 口上定义 fusb302 节点，如下所示：

```
&i2c4 {
```

```
    status = "okay";
    i2c-scl-rising-time-ns = <345>;
    i2c-scl-falling-time-ns = <11>;
    clock-frequency = <400000>;

    fusb1: fusb30x@22 {
        compatible = "fairchild,fusb302";
        reg = <0x22>;
        pinctrl-names = "default";
        pinctrl-0 = <&fusb0_int>;
        int-n-gpios = <&gpio1 2 GPIO_ACTIVE_HIGH>;
        status = "okay";
    };
};
```

2) 使能 typec phy1，如下所示：

```
&tcphy1 {
    status = "okay";
    extcon = <&fusb1>;
};
```

3) 使能 cdn_dp 节点，如下所示：

```
&cdn_dp_fb {
    status = "okay";
    extcon = <&fusb1>;
    dp_vop_sel = <DISPLAY_SOURCE_LCDC0>;
    dp_defaultmode = <0>;
    phys = <&tcphy1_dp>;
};
```

2.2.3 显示相关配置

2.2.3.1 VOP 接口选择

关于显示，由于 RK3399 支持 HDMI 和 TypeC 口两种接口的分体 VR 显示形式，RK3399 有两路 VOP，分别是 Vop Big（vopb）和 Vop Little（vopl），为了确保头盔的显示效果，我们要求头盔必须接 vopb，下面以两种接口形式分别讨论：

1） HDMI 接口

如果头盔通过 HDMI 接口输出，则 hdmi 接口通过 Vop Big（LCDC0）输出，请确认项目 dts 中的 hdmi_rk_fb 节点的 rockchip,hdmi_video_source 项按照如下配置：

```
&hdmi_rk_fb {
        status = "okay";
        rockchip,hdmi_video_source = <DISPLAY_SOURCE_LCDC0>;
};
```

否则，hdmi 从 Vop Little（LCDC1）输出，按照如下方式配置：

```
&hdmi_rk_fb {
        status = "okay";
        rockchip,hdmi_video_source = <DISPLAY_SOURCE_LCDC1>;
};
```

2） TypeC（DP）接口

如果头盔通过 TypeC（DP）接口输出，则 DP 配置为从 Vop Big（LCDC0）输出，请确认项目 dts 中的 cdn_dp_fb 节点的 dp_vop_sel 项按照如下配置：

```
&cdn_dp_fb {
        status = "okay";
        extcon = <&fusb0>;
        dp_vop_sel = <DISPLAY_SOURCE_LCDC0>;
        dp_defaultmode = <0>;
};
```

否则，配置 DP 从 Vop Little（LCDC1）输出，按照如下方式配置：

```
&cdn_dp_fb {
        status = "okay";
        extcon = <&fusb0>;
        dp_vop_sel = <DISPLAY_SOURCE_LCDC1>;
        dp_defaultmode = <0>;
};
```

2.2.3.2 Lcd 屏幕配置

1. 虚拟屏幕配置

目前提供的两个 dts 参考，默认 rk3399 主板端不带 lcd 屏，屏的显示可以通过插 typec 口或者 hdmi 口接头盔或者显示器输出，因此，dts 中配置的默认屏幕均为虚拟屏幕，如下所示：

```
&rk_screen {
```

```
        #include <dt-bindings/display/screen-timing/lcd-box.dtsi>
};

&fb {
        rockchip,uboot-logo-on = <0>;
        rockchip,disp-policy = <DISPLAY_POLICY_BOX>;
};
```

上述代码中，lcd-box.dtsi 文件为虚拟的屏幕配置文件，不是实际物理的屏。这个文件里面定义了几个常用的 timing，比如 702p、1080p 等，项目 dts 中默认使用的是 1080p 的 timing（也就是下面指定的 timing1），如下所示：

```
&disp_timings {
    native-mode = <&timing1>;
    timing1 {
        screen-width = <68>;
        screen-hight = <120>;
    };
};
```

screen-width 和 screen-hight 表示 VR 头盔中的 lcd 屏的实际尺寸，请根据 lcd 手册配置该值。

由于系统初始化时会根据上面的 timing1 中的分辨率定义 frame buffer 的大小，如果头盔的分辨率大于该 timing（1920x1080）中定义的分辨率，则系统实际输出到头盔显示屏的图像是经过缩放的，也就是说在 frame buffer 大小的基础上进行缩放，所以我们要求将头盔的 timing 填入虚拟屏幕的 dts 中：

```
include/dt-bindings/display/screen-timing/lcd-box.dtsi
```

在上面这个文件里面加上和 VR 头盔 lcd 匹配的 timing，如下所示：

```
diff --git a/include/dt-bindings/display/screen-timing/lcd-box.dtsi
b/include/dt-bindings/display/screen-timing/lcd-box.dtsi
index 2109a89..b9cc321 100644
--- a/include/dt-bindings/display/screen-timing/lcd-box.dtsi
+++ b/include/dt-bindings/display/screen-timing/lcd-box.dtsi
@@ -95,4 +95,25 @@ disp_timings: display-timings {
                swap-rg = <0>;
                swap-gb = <0>;
        };
+       timing3: timing3 {
+               screen-type = <SCREEN_RGB>;
```

```
+                out-face    = <OUT_P888>;
+                color-mode = <COLOR_YCBCR>;
+                clock-frequency = <340000000>;
+                hactive = <2880>;
+                vactive = <1440>;
+                hback-porch = <100>;
+                hfront-porch = <50>;
+                vback-porch = <8>;
+                vfront-porch = <6>;
+                hsync-len = <50>;
+                vsync-len = <1>;
+                hsync-active = <1>;
+                vsync-active = <1>;
+                de-active = <0>;
+                pixelclk-active = <0>;
+                swap-rb = <0>;
+                swap-rg = <0>;
+                swap-gb = <0>;
+        };
 };
```

然后在对应的项目 dts 里面（rk3399-disvr-android.dts 或者 rk3399-box-rev1-disvr.dts），将 timing 改为上面新加的 timing3：

```
+&disp_timings {
+       native-mode = <&timing3>;
+       timing3 {
+               screen-width = <104>;
+               screen-hight = <52>;
        };
 };
```

2. 头盔屏幕配置

如果需要通过 rk3399 点亮一个新的头盔，请在之前所述步骤的基础上添加该头盔的 timing 信息到下述文件中，添加了 timing 以后，插入的头盔能够自动识别 edid 并且输出对应的显示信号：

```
diff --git a/drivers/video/rockchip/hdmi/rockchip-hdmi-lcdc.c
b/drivers/video/rockchip/hdmi/rockchip-hdmi-lcdc.c
index db6d0a4..957d68d 100644
--- a/drivers/video/rockchip/hdmi/rockchip-hdmi-lcdc.c
```

```
+++ b/drivers/video/rockchip/hdmi/rockchip-hdmi-lcdc.c
@@ -771,6 +771,50 @@ static const struct hdmi_video_timing hdmi_mode[]
= {
                .pixelrepeat = 1,
                .interface = OUT_P888,
        },
+       {
+               .mode = {
+                       .name = "1440x1280@60Hz",
+                       .refresh = 60,
+                       .xres = 1440,
+                       .yres = 1280,
+                       .pixclock = 148500000,
+                       .left_margin = 84,
+                       .right_margin = 360,
+                       .upper_margin = 8,
+                       .lower_margin = 10,
+                       .hsync_len = 20,
+                       .vsync_len = 2,
+                       .sync = 0,
+                       .vmode = 0,
+                       .flag = 0,
+               },
+               .vic = HDMI_VIDEO_DISCRETE_VR | 1,
+               .vic_2nd = 0,
+               .pixelrepeat = 1,
+               .interface = OUT_P888,
+       },
};
```

其中上述配置中的.vic 表示这个 timing 对应的 id，系统会通过 edid 匹配到对应的 vic，然后输出对应的显示信号。.vic 的值我们要求新添加的 id 必须符合下面的格式：

```
.vic = HDMI_VIDEO_DISCRETE_VR | xxx,
```

其中 xxx 为客户指定的 id 值，只要不和原有的 id 重复即可。

上述配置以后，插入头盔，会在 kernel 的 log 中有如下打印：

✧ DP 接口头盔

```
rk322x-lcdc vop0: lcdc0: dclk: 148500000>>fps:60
cdn-dp-fb fec00000.dp-fb: rate:10, lanes:2
```

✧ HDMI 头盔

```
rk322x-lcdc vop0: lcdc0: dclk: 148500000>>fps:60
```

其中 log 中的 dclk: 148500000 表示输出的 dclock 的实际值，这个值必须和头盔对应的 timing 匹配，即上面所加 timing 中的.pixclock = 148500000，如果不匹配，则说明没有读取到正确的 edid，请确认头盔端的 edid 是否配置正确。如果始终无法读取到正确的 edid，可以尝试在代码中将 dp 或者 hdmi 头盔需要输出的 timing 写死，如下所示：

✧ DP 头盔

假设客户新添加的 timing 对应的 vic 为.vic = HDMI_VIDEO_DISCRETE_VR | 5：

```
diff --git a/drivers/video/rockchip/dp/rockchip_dp.c
b/drivers/video/rockchip/dp/rockchip_dp.c
index 6570937..c7ad8fc 100644
--- a/drivers/video/rockchip/dp/rockchip_dp.c
+++ b/drivers/video/rockchip/dp/rockchip_dp.c
@@ -230,9 +230,9 @@ int cdn_dp_fb_register(struct platform_device
*pdev, void *dp)
        rk_cdn_dp_prop->videosrc = dp_dev->disp_info.vop_sel;
        rk_cdn_dp_prop->display = DISPLAY_MAIN;
        if (!of_property_read_u32(np, "dp_defaultmode", &val))
-               rk_cdn_dp_prop->defaultmode = val;
+               rk_cdn_dp_prop->defaultmode =
HDMI_VIDEO_DISCRETE_VR | 5;
        else
-               rk_cdn_dp_prop->defaultmode =
HDMI_VIDEO_DEFAULT_MODE;
+               rk_cdn_dp_prop->defaultmode =
HDMI_VIDEO_DISCRETE_VR | 5;
        rk_cdn_dp_prop->name = (char *)pdev->name;
        rk_cdn_dp_prop->priv = dp_dev;
        rk_cdn_dp_prop->feature |=
```

✧ HDMI 头盔

```
diff --git a/drivers/video/rockchip/hdmi/rockchip-hdmi.h
b/drivers/video/rockchip/hdmi/rockchip-hdmi.h
index a5e8dd5..4bd9a6c 100644
--- a/drivers/video/rockchip/hdmi/rockchip-hdmi.h
+++ b/drivers/video/rockchip/hdmi/rockchip-hdmi.h
@@ -487,7 +487,7 @@ struct hdmi {
```

```
 #define HDMI_AUTO_CONFIG                    false

 /* HDMI default vide mode */
-#define HDMI_VIDEO_DEFAULT_MODE
HDMI_1280X720P_60HZ
+#define HDMI_VIDEO_DEFAULT_MODE
(HDMI_VIDEO_DISCRETE_VR | 5)

/*HDMI_1920X1080P_60HZ*/
 #define HDMI_VIDEO_DEFAULT_COLORMODE
HDMI_COLOR_AUTO
 #define HDMI_VIDEO_DEFAULT_COLORDEPTH            8
```

2.2.3.3 头盔和 hdmi 同时显示

分体机 VR 支持 TypeC VR 头盔和 HDMI 同时显示功能，但是目前默认的代码可能在适配所有的 HDMI 电视上有些问题，如果要支持 Typec 头盔和 hdmi 电视同时显示（typec 头盔接 vopb，hdmi 电视接 vopl），请更新代码到 2017 年 1 月份以后，并且需要手动打上下面的补丁（以 rk3399-box-rev1-disvr.dts 为例）：

- Uboot

  需要在 include/configs /rk33plat.h 中加上如下配置：

```
diff --git a/include/configs/rk33plat.h b/include/configs/rk33plat.h
index bff9399..5ef0abe 100755
--- a/include/configs/rk33plat.h
+++ b/include/configs/rk33plat.h
@@ -312,7 +312,7 @@
 #define CONFIG_DIRECT_LOGO
 #define CONFIG_OF_BOARD_SETUP
 #undef CONFIG_RK_TVE
-#undef CONFIG_RK_VOP_DUAL_ANY_FREQ_PLL
+#define CONFIG_RK_VOP_DUAL_ANY_FREQ_PLL
 #endif

 #ifdef CONFIG_ROCKCHIP_DISPLAY
```

- Kernel

```
diff --git a/arch/arm64/boot/dts/rockchip/rk3399-box-rev1-disvr.dts
```

```
b/arch/arm64/boot/dts/rockchip/rk3399-box-rev1-disvr.dts
index 052ea17..9c82a74 100644
--- a/arch/arm64/boot/dts/rockchip/rk3399-box-rev1-disvr.dts
+++ b/arch/arm64/boot/dts/rockchip/rk3399-box-rev1-disvr.dts
@@ -167,10 +167,14 @@

 &vopb_rk_fb {
        status = "okay";
+        assigned-clocks = <&cru DCLK_VOP0_DIV>;
+        assigned-clock-parents = <&cru PLL_CPLL>;
 };

 &vopl_rk_fb {
        status = "okay";
+         assigned-clocks = <&cru DCLK_VOP1_DIV>;
+         assigned-clock-parents = <&cru PLL_VPLL>;
 };

 &disp_timings {
diff --git a/include/dt-bindings/clock/rk3399-cru.h
b/include/dt-bindings/clock/rk3399-cru.h
index 0fc9e7a..d32ce01 100644
--- a/include/dt-bindings/clock/rk3399-cru.h
+++ b/include/dt-bindings/clock/rk3399-cru.h
@@ -16,7 +16,7 @@
 #ifndef _DT_BINDINGS_CLK_ROCKCHIP_RK3399_H
 #define _DT_BINDINGS_CLK_ROCKCHIP_RK3399_H

-/* #define RK3399_TWO_PLL_FOR_VOP */
+#define RK3399_TWO_PLL_FOR_VOP

 /* core clocks */
 #define PLL_APLLL                       1
```

2.2.3.4 分体 VR 单双屏支持

分体机头盔支持单屏和双屏两种配置，系统默认为单屏模式，如果需要修改为双屏模式，请在对应的项目 dts 里面修改如下配置（以 rk3399-box-rev1-disvr.dts 为例）：

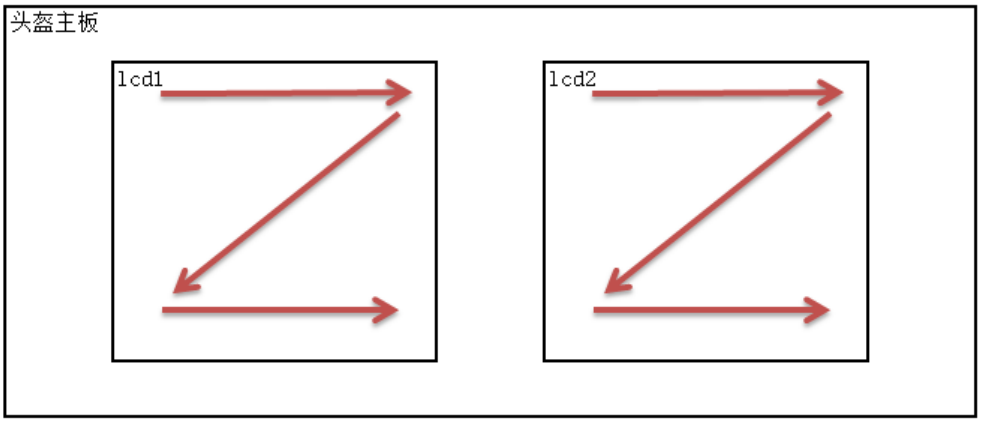
```
diff --git a/arch/arm64/boot/dts/rockchip/rk3399-box-rev1-disvr.dts
b/arch/arm64/boot/dts/rockchip/rk3399-box-rev1-disvr.dts
index 052ea17..005f85d 100644
--- a/arch/arm64/boot/dts/rockchip/rk3399-box-rev1-disvr.dts
+++ b/arch/arm64/boot/dts/rockchip/rk3399-box-rev1-disvr.dts
@@ -167,6 +167,7 @@

 &vopb_rk_fb {
        status = "okay";
+        rockchip,dsp_mode = <ONE_VOP_DUAL_MIPI_HOR_SCAN>;
 };

 &vopl_rk_fb {
```

【注】这里我们假设头盔是接在 vopb 上面，也就是默认配置。

另外，需要注意，双屏的硬件结构上，头盔端 lcd 屏有默认的安装方向，我们系统默认只支持竖屏扫描，如下图所示：



上图所画的扫描方向可以是从上往下，或者从下往上，但是不能从左往右或者从右往左。

2.2.4 Sensor 相关配置

分体机 VR，我们提供的头盔端的硬件参考设计使用 NanoC 作为头盔端控制器，我们默认的代码只支持 RK 自己的头盔设计，如果客户需要使用他们自己原有的头盔，则需要另外修改。因此，这里详细描述两种头盔所需的 kernel 端修改和配置方法。

首先不论是那种头盔，都必须确认项目 dts 中下述节点是否存在（如果参考我们默认的分体 VR 的 dts，是存在这个配置的）：

```
        mpu6500_hid {
                status = "okay";
                compatible = "inv-hid,mpu6500";
        };
```

2.2.4.1 NanoC 头盔 sensor 相关配置

SDK 默认的代码不需要进行任何修改即可适配，而 sensor 的方向调整，请参考 3.3 节内容。

2.2.4.2 非 NanoC 头盔 sensor 相关配置

结合实例，非 NanoC 头盔 Sensor 配置步骤如下。

- ✧ 定义 VID 和 PID

```
    --- a/drivers/hid/hid-ids.h
    +++ b/drivers/hid/hid-ids.h
    @@ -828,6 +828,8 @@
     #ifdef CONFIG_HID_RKVR
     #define USB_VENDOR_ID_ROCKCHIP      0x071b
     #define USB_DEVICE_ID_NANOC      0x3205
    +#define USB_DEVICE_ID_XXX            0x0001
    +#define USB_VENDOR_ID_XXX        0x2B1C
     #endif
```

- ✧ 为 HID 设备驱动添加 VID 和 PID

```
    --- a/drivers/hid/hid-core.c
    +++ b/drivers/hid/hid-core.c
    @@ -1997,6 +1997,7 @@ static const struct hid_device_id
hid_have_special_driver[] = {
     #endif
     #ifdef CONFIG_HID_RKVR
        { HID_USB_DEVICE(USB_VENDOR_ID_ROCKCHIP,
USB_DEVICE_ID_NANOC) },
    +   { HID_USB_DEVICE(USB_VENDOR_ID_XXX,
USB_DEVICE_ID_XXX) },
     #endif
        { HID_USB_DEVICE(USB_VENDOR_ID_SAMSUNG,
USB_DEVICE_ID_SAMSUNG_IR_REMOTE) },
    --- a/drivers/hid/hid-rkvr.c
```

```
+++ b/drivers/hid/hid-rkvr.c
@@ -1406,6 +1415,7 @@ static int rkvr_raw_event(struct hid_device
*hdev, struct hid_report *report, u8

 static const struct hid_device_id rkvr_devices[] = {
    { HID_USB_DEVICE(USB_VENDOR_ID_ROCKCHIP,
USB_DEVICE_ID_NANOC) },
+   { HID_USB_DEVICE(USB_VENDOR_ID_XXX,
USB_DEVICE_ID_XXX) },
    { }
 };
```

✧ 修改 sensor 的 usb interface。如果头盔没有物理按键和触摸板，需要把上报按键事件的代码用宏注释掉。修改如下：

```
--- a/drivers/hid/hid-rkvr.c
+++ b/drivers/hid/hid-rkvr.c
@@ -26,10 +26,16 @@
 #include "hid-rkvr.h"
 #include "hid-ids.h"

+#define XXX_VR
+
 #define USB_TRACKER_INTERFACE_PROTOCOL    0
 /* define rkvr interface number */
 #define RKVR_INTERFACE_USB_AUDIO_ID 1
+#ifdef XXX_VR
+#define RKVR_INTERFACE_USB_SENSOR_ID 0
+#else
 #define RKVR_INTERFACE_USB_SENSOR_ID 2
+#endif
 #define RKVR_INTERFACE_USB_AUDIO_KEY_ID 1
 /* number of reports to buffer */
 #define RKVR_HIDRAW_BUFFER_SIZE 64
@@ -507,6 +513,7 @@ static int rkvr_hidraw_release(struct inode
*inode, struct file *file)
    return 0;
 }

+#ifndef XXX_VR
 static void rkvr_send_key_event(struct input_dev *input, int key_value,
int state)
```

```
     {
        if (!input) {
    @@ -585,7 +592,7 @@ static int rkvr_keys_event(struct hid_device
*hdev, void *data, unsigned long le

        return 0;
     }
    +#endif
     static int rkvr_report_event(struct hid_device *hid, u8 *data, int len)
     {
        struct hidraw *dev = hid->hidraw;
    @@ -596,9 +603,11 @@ static int rkvr_report_event(struct hid_device
*hid, u8 *data, int len)
        struct sensor_hid_data *pdata = hid_get_drvdata(hid);

        spin_lock_irqsave(&dev->list_lock, flags);
    +#ifndef XXX_VR
        if (hid->hiddev) {
            rkvr_keys_event(hid, data, len);
        }
    +#endif
        if (pdata && pdata->priv && pdata->send_event) {
            pdata->send_event(rkvr_data->buf, len, pdata->priv);
            spin_unlock_irqrestore(&dev->list_lock, flags);
```

✧ 确认头盔用到的 sensor 量程，修改 3399 端 sensor 量程。比如加速度是 4g，陀螺仪是 1000DPS，服务器上 3399 端默认量程是 2g，2000DPS

```
    --- a/drivers/staging/iio/imu/inv_mpu/inv_mpu_core.c
    +++ b/drivers/staging/iio/imu/inv_mpu/inv_mpu_core.c
    @@ -281,11 +281,11 @@ static int inv_init_config(struct iio_dev
*indio_dev)
                    return result;
     #endif
            result = inv_plat_single_write(st, reg->gyro_config,
    -               INV_FSR_2000DPS << GYRO_CONFIG_FSR_SHIFT);
    +               INV_FSR_1000DPS << GYRO_CONFIG_FSR_SHIFT);
            if (result)
                    return result;

    -       st->chip_config.fsr = INV_FSR_2000DPS;
    +       st->chip_config.fsr = INV_FSR_1000DPS;
```

```
            result = inv_plat_single_write(st, reg->lpf, INV_FILTER_42HZ);
            if (result)
    @@ -304,9 +304,9 @@ static int inv_init_config(struct iio_dev
*indio_dev)
            st->self_test.samples = INIT_ST_SAMPLES;
            st->self_test.threshold = INIT_ST_THRESHOLD;
            if (INV_ITG3500 != st->chip_type) {
    -               st->chip_config.accl_fs = INV_FS_02G;
    +                st->chip_config.accl_fs = INV_FS_04G;
                    result = inv_plat_single_write(st, reg->accl_config,
    -                       (INV_FS_02G << ACCL_CONFIG_FSR_SHIFT));
    +                        (INV_FS_04G << ACCL_CONFIG_FSR_SHIFT));
                    if (result)
                            return result;
                    st->tap.time = INIT_TAP_TIME;
```

✧ 解析头盔上报的 sensor 数据，由于 3399 端默认是 16bit 的数据，有些头盔是 21bit 的数据，需要加 21bit 转 16bit 算法，这个转换算法由头盔厂商给出（头盔端 sensor 寄存器里面数据一般是 16bit，通过算法转换成 21bit）。下面给出一个解包 sensor 数据例子

```
    --- a/drivers/staging/iio/imu/inv_mpu/inv_mpu_ring.c
    +++ b/drivers/staging/iio/imu/inv_mpu/inv_mpu_ring.c
    @@ -1197,6 +1197,24 @@ static const struct iio_buffer_setup_ops
inv_mpu_ring_setup_ops = {
        .predisable = &inv_predisable,
     };

    +#define XXX_VR
    +#ifdef XXX_VR
    +static void UnpackSensorPackage(const u8* buffer, short* x, short* y,
short* z)
    +{
    +   struct { int n : 21; } s;
    +   int X,Y,Z;
    +
    +   X = s.n = (buffer[0] << 13) | (buffer[1] << 5) | ((buffer[2] &
0xF8) >> 3);
    +   Y = s.n = ((buffer[2] & 0x07) << 18) | (buffer[3] << 10) | (buffer[4]
<< 2) |
    +    ((buffer[5] & 0xC0) >> 6);
    +   Z = s.n = ((buffer[5] & 0x3F) << 15) | (buffer[6] << 7) |
```

```
(buffer[7] >> 1);
    +
    +   *y = -((X >> 5)*74/100)*7;
    +   *x = ((Y >> 5)*74/100)*7;
    +   *z= ((Z >> 5)*74/100)*7;
    +}
    +#endif
    +
     #ifdef CONFIG_HID_RKVR
     /* Callback handler to send event */
     static int inv_proc_event(char *raw_data, size_t raw_len, void *priv)
    @@ -1207,7 +1225,9 @@ static int inv_proc_event(char *raw_data,
size_t raw_len, void *priv)
        int ind;
        u8 buf[64];
        u64 *tmp;
    +#ifndef XXX_VR
        int i;
    +#endif
        char *p;
        struct inv_chip_config_s *conf;

    @@ -1240,19 +1260,27 @@ static int inv_proc_event(char *raw_data,
size_t raw_len, void *priv)
            ind += BYTES_FOR_DMP;

        if (conf->accl_fifo_enable) {
    +#ifdef XXX_VR
    +       UnpackSensorPackage(raw_data+24, a, a+1, a+2);
    +#else
            for (i = 0; i < ARRAY_SIZE(a); i++) {
                a[i] = (p[1] << 8) | p[0];
                p += 2;
            }
    +#endif
            memcpy(&buf[ind], a, sizeof(a));
            ind += BYTES_PER_SENSOR;
        }

        if (conf->gyro_fifo_enable) {
```

```
+#ifdef XXX_VR
+       UnpackSensorPackage(raw_data+32, g, g+1, g+2);
+#else
        for (i = 0; i < ARRAY_SIZE(g); i++) {
            g[i] = (p[1] << 8) | p[0];
            p += 2;
        }
+#endif
        memcpy(&buf[ind], g, sizeof(g));
        ind += BYTES_PER_SENSOR;
    }
```

2.3 编译

2.3.1 Android 端编译说明

- ✧ uboot 编译:

  如果客户拿到的是 VR Station SDK，编译命令如下：

```
cd u-boot
make rk3399_box_defconfig
make ARCHV=aarch64
```

  如果拿到的是 RK3399 VR 和平板 SDK，则为：

```
cd u-boot
make rk3399_defconfig
make ARCHV=aarch64
```

- ✧ kernel 编译:

  如上一章所述，分体 VR 提供两个参考 dts 配置，

  arch/arm64/boot/dts/rockchip/rk3399-disvr-android.dts

  arch/arm64/boot/dts/rockchip/rk3399-box-rev1-disvr.dts

  针对这两个 dts，对应的编译命令如下：

```
cd kernel
make ARCH=arm64 rockchip_defconfig -j8
make ARCH=arm64 rk3399-disvr-android.img -j12
或者
make ARCH=arm64 rk3399-box-rev1-disvr.img -j12
```

- ✧ android 编译:

  如果客户拿到的是 VR Station SDK，编译命令如下：

```
source build/envsetup.sh
```

```
lunch rk3399_stbvr-userdebug
make -j12
./mkimages.sh
```

如果拿到的是 RK3399 VR 和平板 SDK，则为：

```
source build/envsetup.sh
lunch rk3399_disvr-userdebug
make -j12
./mkimages.sh
```

2.3.2 NanoC 端编译说明

请参考代码仓库中，“RKDocs”目录下的文档：

《RK3399_VR 分体机_NANOC_编译和烧写说明文档.pdf》

3 NanoC 端各个模块说明

3.1 按键修改说明

请参考代码仓库中，“RKDocs/”目录下的文档：

《RK3399_VR 分体机_KEY_修改说明文档.pdf》

3.2 LCD 显示修改说明

请参考代码仓库中，“RKDocs/”目录下的文档：

《RK3399_VR 分体机_NanoC_显示屏参数修改说明文档.pdf》

3.3 Sensor 修改说明文档

请参考代码仓库中，“RKDocs/”目录下的文档：

《RK3399_VR 分体机_NANOC_sensor 方向配置.pdf》

3.4 NanoC 端工具说明文档

请参考代码仓库中，“RKDocs/”目录下的文档：

《RK3399_VR 分体机_NANOC_编译工具安装说明文档.pdf》