Rockchip RK628 MCU 开发指南

文件标识：RK-YH-YF-287

发布版本：V1.2.0

日期：2024-06-28

文件密级：□绝密 □秘密 □内部资料 ■公开

瑞芯微电子股份有限公司

Rockchip Electronics Co., Ltd.

地址： 福建省福州市铜盘路软件园 A 区 18 号

网址： www.rock-chips.com

客户服务电话： +86-4007-700-590

客户服务传真： +86-591-83951833

客户服务邮箱： fae@rock-chips.com

前言

本文档主要描述 RK628 的软件配置、固件编译、固件下载以及调试手段。目前 RK628 系列芯片分为四个型号：RK628D、RK628E、RK628F、RK628H，支持第三方 MCU / 主控通过 I2C 接口进行控制。另外，RK628E / F / H 内部集成了一个 RISC-V MCU，因此还可以在不依赖第三方 MCU / 主控控制的情况下，通过内部的 MCU 来独立编程控制。目前 RK628 MCU 模式支持以 **HDMI** 输入，**DSI / CSI / GVI / LVDS** 输出。

读者对象

本文档（本指南）主要适用于以下工程师：

技术支持工程师

软件开发工程师

修订记录

版本号	作者	修改日期	修改说明
V1.0.0	黄国椿	2021-04-06	初始发布
V1.1.0	陈顺庆	2021-05-28	增加HDMI输入
V1.2.0	黄智斌	2024-01-18	补充对 RK628F/H 驱动的支持

1. 目录

Rockchip RK628 MCU 开发指南

1. 目录
2. 前言
   - 2.1 RK628 芯片框图
   - 2.2 RK628 主要规格
     - 2.2.1 RK628D 主要规格
     - 2.2.2 RK628E 主要规格
     - 2.2.3 RK628F 主要规格
     - 2.2.4 RK628H 主要规格
3. 软件结构
4. 软件配置
   - 4.1 核心配置
     - 4.1.1 内部 MCU 模式
     - 4.1.2 第三方 MCU 访问模式
       - 4.1.2.1 平台移植
   - 4.2 输入输出模块选择
   - 4.3 Panel 端配置
     - 4.3.1 Timing 配置
     - 4.3.2 Panel 上电控制时序实现
   - 4.4 输入模块配置
     - 4.4.1 HDMI 输入
       - 4.4.1.1 配置 HDMI 输入
       - 4.4.1.2 HDMI 检测脚配置
       - 4.4.1.3 注意事项
   - 4.5 输出模块配置
     - 4.5.1 DSI 输出
       - 4.5.1.1 单 DSI 输出
       - 4.5.1.2 双 DSI 输出
       - 4.5.1.3 DSI panel 初始化序列配置
     - 4.5.2 CSI 输出
     - 4.5.3 LVDS 输出
       - 4.5.3.1 单 LVDS 输出
       - 4.5.3.2 双 LVDS 输出
       - 4.5.3.3 双 LVDS 左右屏
     - 4.5.4 GVI 输出
5. 固件编译
6. 固件下载
7. 调试方法

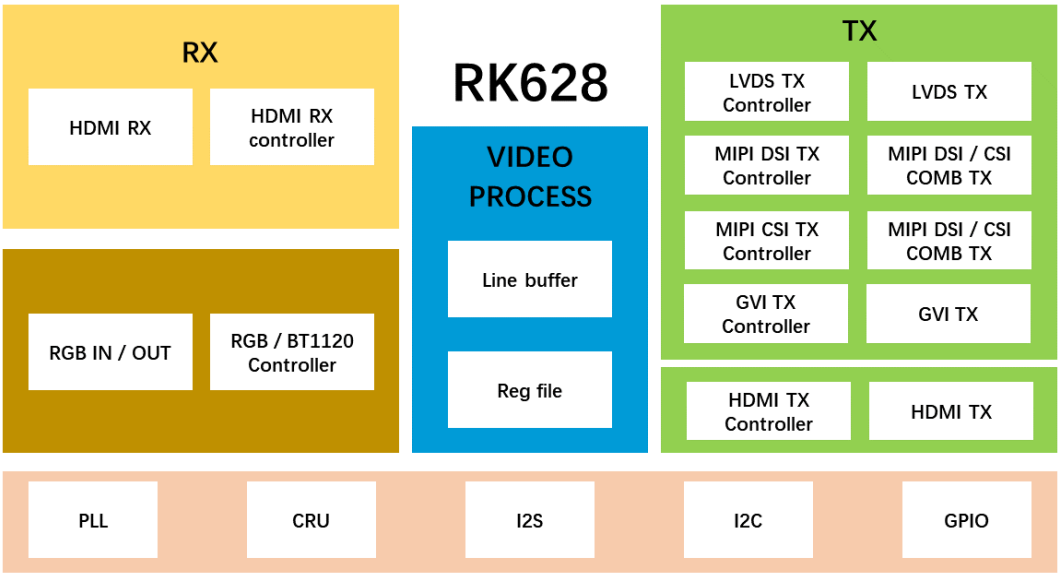
2. 前言

本文档主要描述 RK628 的软件配置、固件编译、固件下载以及调试手段，目前 RK628 系列芯片分为四个型号：RK628D、RK628E、RK628F、RK628H，支持第三方 MCU / 主控通过 I2C 接口进行控制。

另外，RK628E / F / H 内部集成了一个 RISC-V MCU，因此还可以在不依赖第三方 MCU / 主控控制的情况下，通过内部的 MCU 来独立编程控制。

目前 RK628 MCU 模式支持以 **HDMI** 输入，**DSI / CSI / GVI / LVDS** 输出。具体功能描述参考 datasheet。

2.1 RK628 芯片框图



2.2 RK628 主要规格

2.2.1 RK628D 主要规格

Video Iutput Interface	**Typical Resolution**	**Typical Format**
HDMI IN	4K@30	YUV420(4K@60) / YUV422 / YUV444 / RGB888
VIDEO Output Interface	**Typical Resolution**	**Typical Format**
GVI OUT	4K@60	RGB888
MIPI DSI	单通道 1080P@60 / 双通道 2.5K@60	RGB888
MIPI CSI	4k@30	YUV422
LVDS OUT	单通道 720P@60 / 双通道 1080P@60	RGB888

RK628D 仅支持第三方 MCU 通过 I2C 访问模式

2.2.2 RK628E 主要规格

Video Iutput Interface	**Typical Resolution**	**Typical Format**
HDMI IN	4K@30	YUV420(4K@60) / YUV422 / YUV444 / RGB888
VIDEO Output Interface	**Typical Resolution**	**Typical Format**
GVI OUT	4K@60	RGB888
MIPI DSI	单通道 1080P@60 / 双通道 2.5K@60	RGB888
MIPI CSI	4k@30	YUV422
LVDS OUT	单通道 720P@60 / 双通道 1080P@60	RGB888

2.2.3 RK628F 主要规格

Video Iutput Interface	Typical Resolution	Typical Format
HDMI IN	4K@60	RGB888 / YUV422 8bit / YUV422 10bit / YUV420 8bit / YUV420 10bit
VIDEO Output Interface	**Typical Resolution**	**Typical Format**
GVI OUT	4K@60	RGB888
MIPI DSI	单通道 1080P@60 / 双通道 2.5K@60	RGB888
MIPI CSI	单通道 4k@30 / 双通道 4k@60	YUV422
LVDS OUT	单通道 720P@60 / 双通道 1080P@60	RGB888

2.2.4 RK628H 主要规格

对比 **RK628F，RK628H** 没有 **GVI** 接口功能。

Video Iutput Interface	Typical Resolution	Typical Format
HDMI IN	4K@60	RGB888 / YUV422 8bit / YUV422 10bit / YUV420 8bit / YUV420 10bit
VIDEO Output Interface	**Typical Resolution**	**Typical Format**
MIPI DSI	单通道 1080P@60 / 双通道 2.5K@60	RGB888
MIPI CSI	单通道 4k@30 / 双通道 4k@60	YUV422
LVDS OUT	单通道 720P@60 / 双通道 1080P@60	RGB888

3. 软件结构

以下为 RK628 软件结构：

```
.
├── common # RISC-V 核心文件，部分接口需要根据第三方 MCU 重新封装
│   ├── arch.h
│   ├── csr.h
│   ├── platform_config.h
│   ├── printf.c
│   ├── rk628_ram.ld
```

```
│   ├── rk628_xip.ld
│   ├── stringify.h
│   ├── uart.c
│   └── uart.h
├── firmware_merger # 打包固件工具、固件路径及下载工具
│   ├── config.json
│   ├── Image
│   │   ├── Boot2_FPGA.bin
│   │   ├── Hornet_loader.bin
│   │   └── Hornet_upgrade.bin
│   ├── mkimage.sh
│   ├── setting.ini
│   └── tools
│       ├── firmware_merger
│       ├── resource_header_tool
│       └── UpgradeComTool_v1.22.rar
├── Makefile # 编译配置文件
└── src # 核心源文件目录
    ├── combrxphy.c
    ├── combrxphy.h
    ├── combtxphy.c
    ├── combtxphy.h
    ├── config.c
    ├── config.h
    ├── cru.c
    ├── cru.h
    ├── csi.c
    ├── csi.h
    ├── dsi.c
    ├── dsi.h
    ├── gvi.c
    ├── gvi.h
    ├── hdmirx_audio.c
    ├── hdmirx_audio.h
    ├── hdmirx.c
    ├── hdmirx.h
    ├── lvds.c
    ├── lvds.h
    ├── panel.c
    ├── panel.h
    ├── post_process.c
    ├── post_process.h
    ├── rk628.c
    ├── rk628-gpio.h
    ├── rk628-grf.h
    ├── rk628.h
    ├── rk628-pinctrl.c
    ├── rk628-pinctrl.h
    ├── startup.S # RK628 内部 MCU 启动文件
    └── trap.c
```

4. 软件配置

4.1 核心配置

4.1.1 内部 MCU 模式

RK628E/F/H 作为 MCU 运行内部 MCU 模式的硬件前提是需要外挂 Flash，并通过烧写工具 UpgradeComTool 将镜像烧入 flash 中，烧写方式详见“固件下载”章节。

4.1.2 第三方 MCU 访问模式

RK628E/F/H 在硬件上外挂 Flash，会自动进入内部 MCU 模式，所以通过第三方 MCU 访问时，需要将硬件上外挂的 Flash 去掉。

4.1.2.1 平台移植

RK628 的地址和数据都是32bit，根据下图 RK628 I2C操作流程封装出访问接口：

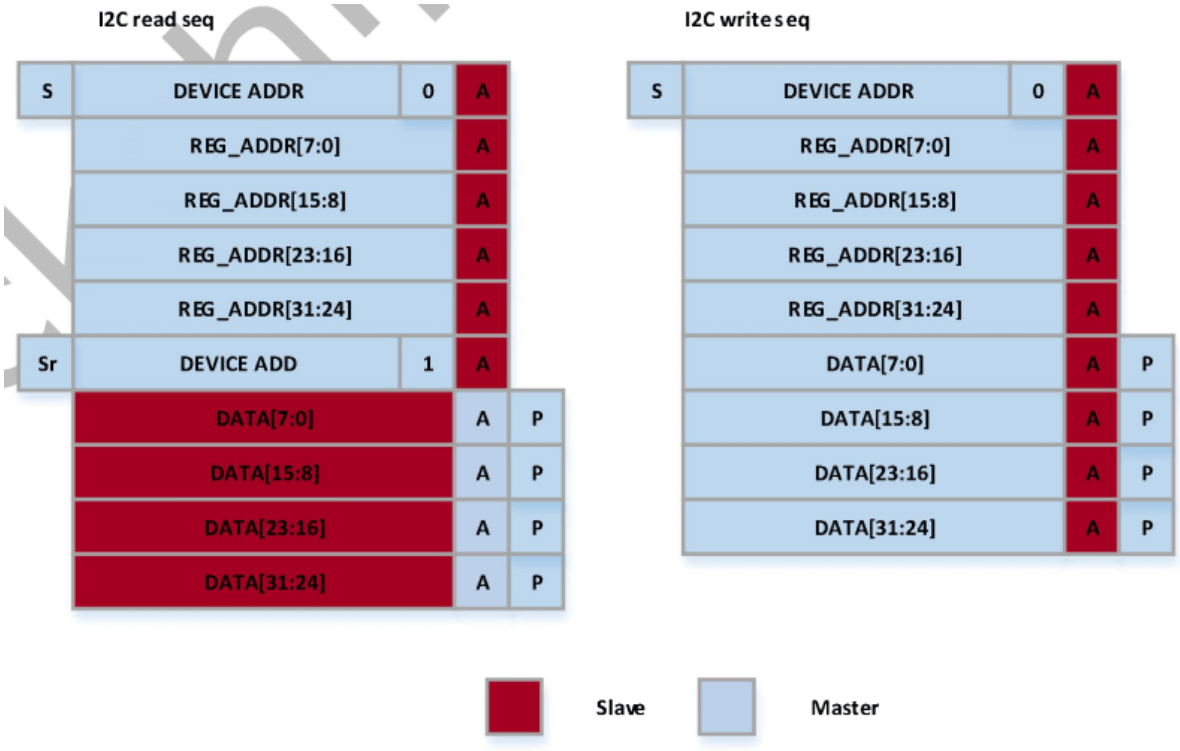


Fig. 5- 4 i2c2apb single read and write

第三方 MCU 通过 I2C 访问 RK628 寄存器，使用第三方 MCU 访问时，需要改写 rk628.h 的 writel 和 readl 的实现，补丁参考如下：

```
diff --git a/src/rk628.h b/src/rk628.h
index d10469b..320fc7f 100644
--- a/src/rk628.h
+++ b/src/rk628.h
@@ -272,12 +272,12 @@ inline bool is_rk628f()

 static inline void writel(uintptr_t reg, uint32_t val)
 {
-       *(volatile uint32_t *)reg = val;
+       i2c_write(reg, val);
```

```
 }

 static inline uint32_t readl(uintptr_t reg)
 {
-       return *(volatile uint32_t *)reg;
+       return i2c_read(reg);
 }

 static inline void modb(uintptr_t reg, unsigned mask, unsigned val)
```

可以通过 dump RK628 的寄存器来判断是否通信成功：

```
diff --git a/src/rk628.c b/src/rk628.c
index 6ad0b21..efbd986 100644
--- a/src/rk628.c
+++ b/src/rk628.c
@@ -246,6 +246,8 @@ int main(void)
 {
        const struct rk628_cmd *prev_cmd = 0;
        void *prev_addr = 0;
+       uintptr_t i;
+       uint32_t val;

        cru_init();

@@ -275,6 +277,11 @@ int main(void)
        usage();

        rk628_get_version();
+       for (i = 0x0; i <= 0x300; i += 0x4) {
+               val = readl(i);
+               printf("reg: 0x%x, data: 0x%x\n",i, val);
+       }
+
        rk628_display_enable();

        while (1) {
```

内部 MCU 寄存器基地址为 0x80000000，而 I2C 通信访问这些寄存器的基地址为 0x0，所以需要将 rk628.h 中 RK628_REG_BASE 宏定义设置为0x0，补丁参考如下：

```
diff --git a/src/rk628.h b/src/rk628.h
index a5af352..db06c01 100644
--- a/src/rk628.h
+++ b/src/rk628.h
@@ -24,7 +24,7 @@
 #define DIV_ROUND_UP(n,d) (((n) + (d) - 1) / (d))
 #define DIV_ROUND_CLOSEST_ULL(x, divisor) ((x) + (divisor) / 2) /divisor

-#define RK628_REG_BASE                 0x80000000
+#define RK628_REG_BASE                 0x0

 #define RK628_RXPHY_BASE               (RK628_REG_BASE + 0x10000)
 #define RK628_HDMIRX_BASE              (RK628_REG_BASE + 0x30000)
```

4.2 输入输出模块选择

在 rk628.h 中通过宏定义选择输入输出模块：

```
#define RK628_HDMI_IN

// #define  RK628_DSI_OUT
// #define RK628_CSI_OUT
// #define  RK628_GVI_OUT
#define RK628_LVDS_OUT
```

目前输入有 HDMI IN，输出有 DSI / CSI / GVI / LVDS，所以需要配置对应的输入输出，如上配置的是 HDMI IN + LVDS OUT。DSI / CSI / GVI / LVDS 的 OUT 选项是不可同时打开，只能根据使用场景选择其中一个输出模块。各个输入输出模块配置详见《输入模块配置》及《输出模块配置》章节。

4.3 Panel 端配置

4.3.1 Timing 配置

在 config.c 中修改配置：

```
static struct drm_display_mode src_mode = {
    /* .clock = */ 148500,
    /* .hdisplay = */ 1920,
    /* .hsync_start = */ 1920 + 88,
    /* .hsync_end = */ 1920 + 88 + 44,
    /* .htotal = */ 1920 + 88 + 44 + 148,
    /* .vdisplay = */ 1080,
    /* .vsync_start = */ 1080 + 4,
    /* .vsync_end = */ 1080 + 4 + 5,
    /* .vtotal = */ 1080 + 4 + 5 + 36,
    /* .flags = */ DRM_MODE_FLAG_PHSYNC | DRM_MODE_FLAG_PVSYNC,
};

const static struct drm_display_mode dst_mode = {
    /* .clock = */ 148500,
    /* .hdisplay = */ 1920,
    /* .hsync_start = */ 1920 + 88,
    /* .hsync_end = */ 1920 + 88 + 44,
    /* .htotal = */ 1920 + 88 + 44 + 148,
    /* .vdisplay = */ 1080,
    /* .vsync_start = */ 1080 + 4,
    /* .vsync_end = */ 1080 + 4 + 5,
    /* .vtotal = */ 1080 + 4 + 5 + 36,
    /* .flags = */ DRM_MODE_FLAG_PHSYNC | DRM_MODE_FLAG_PVSYNC,
};
```

如果 RK628 输入端与输出端的分辨率采用缩放方式，则将输入端的 timing 配置到 src_mode，将输出端 timing 配置到 dst_mode。
缩放前后 timing 需要保证显示部分与消隐部分缩放比例一致：

```
src_mode.hdisplay / dst_mode.hdisplay = src_mode.htotal / dst_mode.htotal
src_mode.vdisplay / dst_mode.vdisplay = src_mode.vtotal / dst_mode.vtotal
```

且缩放前后帧率需要保持一致

```
src_mode.clock / (src_mode.htotal * src_mode.vtotal) = dst_mode.clock /
(dst_mode.htotal * dst_mode.vtotal)
```

如果输入端与输出端分辨率一致，则 src-timing 和 dst-timing 配置相同的目标 timing。

如果用到两个同分辨率的 DSI 单屏或两个同分辨率的 LVDS 单屏，需要对 src_mode 和 dst_mode 中的 clock、hdisplay、hsync_start、hsync_end 和 htotal 在原有单屏配置基础上乘以 2。

4.3.2 Panel 上电控制时序实现

在 panel.c 修改实现

```
void panel_pre_enable(void)
{
    /* operation panel rst/enable/power-supply */

#ifdef RK628_DSI_OUT
    panel_simple_xfer_dsi_cmd_seq();
#endif
}

void panel_enable(void)
{
    /* operation panel backlight */
}

void panel_disable(void)
{
    /* reverse operation panel_pre_enable/panel_enable */
}
```

4.4 输入模块配置

4.4.1 HDMI 输入

4.4.1.1 配置 HDMI 输入

在 rk628.h 中配置

```
#define RK628_HDMI_IN
```

4.4.1.2 HDMI 检测脚配置

在 hdmirx.c 中配置，硬件需设计 HDP 为高电平有效

```
#define HPD_DETECT_GPIO            GPIO0_A7
```

4.4.1.3 注意事项

1. RK628F/H 理论支持任意频点，但RK628D/E HDMIRX对时钟率点有要求，目前只支持下面这些率点：

   ```
   27M, 64M, 74.25M, 148.5M, 297M, 594M
   ```

2. 因为HDMI输入的分辨率、颜色格式等是可能被切换的，因为率点参数计数比较耗时，所以切分辨率过程时间可能会比较长，大概需要几秒钟。

4.5 输出模块配置

4.5.1 DSI 输出

1. 在 rk628.h 中配置

```
#define RK628_DSI_OUT
```

2. 在 config.c 中配置 dsi host 输出类型：

```
/* option config dsi0 */
const static struct rk628_dsi rk628_dsi0 = {
        /* .bpp = */ 24,
        /* .bus_format */ MIPI_DSI_FMT_RGB888,
        /* .slave = */ FALSE,
        /* .master = */ FALSE,
        /* .channel = */ 0,
        /* .reg_base*/ RK628_DSI0_BASE,
        /* .lanes */ 4,
        /* .id */ 0,
        /* .mode_flags */ MIPI_DSI_MODE_VIDEO |
                MIPI_DSI_MODE_VIDEO_BURST |
                MIPI_DSI_MODE_LPM |
                MIPI_DSI_MODE_EOT_PACKET,
};

/* option config dsi1 for dual dsi */
const static struct rk628_dsi rk628_dsi1 = {
        /* .bpp = */ 24,
        /* .bus_format */ MIPI_DSI_FMT_RGB888,
        /* .slave = */ FALSE,
        /* .master = */ FALSE,
        /* .channel = */ 0,
        /* .reg_base*/ RK628_DSI1_BASE,
        /* .lanes */ 4,
        /* .id */ 1,
        /* .mode_flags */ MIPI_DSI_MODE_VIDEO |
```

```
                MIPI_DSI_MODE_VIDEO_BURST |
                MIPI_DSI_MODE_LPM |
                MIPI_DSI_MODE_EOT_PACKET,
};
```

属性说明

Property	Description	Option Value
bpp	每个像素 bits 数	16 / 18 / 24
bus_format	DSI 数据格式	MIPI_DSI_FMT_RGB888 / MIPI_DSI_FMT_RGB666 / MIPI_DSI_FMT_RGB666_PACKED / MIPI_DSI_FMT_RGB565
lanes	DSI lanes	1 / 2 / 4
slave	双通道 DSI	TRUE/FALSE
master	双通道 DSI	TRUE/FALSE
flags	DSI 模式选择	MIPI_DSI_MODE_VIDEO | MIPI_DSI_MODE_VIDEO_BURST | MIPI_DSI_MODE_LPM | MIPI_DSI_MODE_EOT_PACKET

注：除了表格中描述的属性可以根据 option value 做修改外，其他属性不建议修改。

4.5.1.1 单 DSI 输出

rk628_dsi0 和 rk628_dsi1 的 slave 和 master 均为 FALSE：

```
/* option config dsi0 */
const static struct rk628_dsi rk628_dsi0 = {
        /* .bpp = */ 24,
        /* .bus_format */ MIPI_DSI_FMT_RGB888,
        /* .slave = */ FALSE,
        /* .master = */ FALSE,
        /* .channel = */ 0,
        /* .reg_base*/ RK628_DSI0_BASE,
        /* .lanes */ 4,
        /* .id */ 0,
        /* .mode_flags */ MIPI_DSI_MODE_VIDEO |
                MIPI_DSI_MODE_VIDEO_BURST |
                MIPI_DSI_MODE_LPM |
                MIPI_DSI_MODE_EOT_PACKET,
};

/* option config dsi1 for dual dsi */
const static struct rk628_dsi rk628_dsi1 = {
        /* .bpp = */ 24,
        /* .bus_format */ MIPI_DSI_FMT_RGB888,
        /* .slave = */ FALSE,
        /* .master = */ FALSE,
        /* .channel = */ 0,
        /* .reg_base*/ RK628_DSI1_BASE,
        /* .lanes */ 4,
```

```
        /* .id */ 1,
        /* .mode_flags */ MIPI_DSI_MODE_VIDEO |
                  MIPI_DSI_MODE_VIDEO_BURST |
                  MIPI_DSI_MODE_LPM |
                  MIPI_DSI_MODE_EOT_PACKET,
};
```

4.5.1.2 双 DSI 输出

修改 rk628_dsi0 slave 为 TRUE，rk628_dsi1 master 为 TRUE：

```
/* option config dsi0 */
const static struct rk628_dsi rk628_dsi0 = {
        /* .bpp = */ 24,
        /* .bus_format */ MIPI_DSI_FMT_RGB888,
        /* .slave = */ TRUE,
        /* .master = */ FALSE,
        /* .channel = */ 0,
        /* .reg_base*/ RK628_DSI0_BASE,
        /* .lanes */ 4,
        /* .id */ 0,
        /* .mode_flags */ MIPI_DSI_MODE_VIDEO |
                  MIPI_DSI_MODE_VIDEO_BURST |
                  MIPI_DSI_MODE_LPM |
                  MIPI_DSI_MODE_EOT_PACKET,
};

/* option config dsi1 for dual dsi */
const static struct rk628_dsi rk628_dsi1 = {
        /* .bpp = */ 24,
        /* .bus_format */ MIPI_DSI_FMT_RGB888,
        /* .slave = */ FALSE,
        /* .master = */ TRUE,
        /* .channel = */ 0,
        /* .reg_base*/ RK628_DSI1_BASE,
        /* .lanes */ 4,
        /* .id */ 1,
        /* .mode_flags */ MIPI_DSI_MODE_VIDEO |
                  MIPI_DSI_MODE_VIDEO_BURST |
                  MIPI_DSI_MODE_LPM |
                  MIPI_DSI_MODE_EOT_PACKET,
};
```

4.5.1.3 DSI panel 初始化序列配置

根据 panel vendor 提供的初始化序列在 panel.c 中按如下格式配置：

```
const struct panel_cmd_seq {
    unsigned int cmd_cnt;
    unsigned int init_sequence[5][7];
} panel_cmd_init_seq = {
    5,
    {
```

```
        { 0x23, 0x00, 0x02, 0xd1, 0x2e },
        { 0x23, 0x00, 0x02, 0xd2, 0x32 },
        { 0x23, 0x00, 0x02, 0xd3, 0x00 },
        { 0x29, 0x00, 0x04, 0xff, 0x98, 0x81, 0x00 },
        { 0x05, 0x78, 0x01, 0x11 },
        { 0x05, 0x00, 0x01, 0x29 },
    },
};
```

panel_cmd_seq 定义为序列结构体，包含两个成员：cmd_cnt 是 cmd 数量，init_sequence 存放 cmd 数据。
init_sequence 是一个二维数组数组大小，大小为 m*n，m 为 cmd 数量，n 为 最长命令的长度；
init_sequence 第一列表示 data type，第二列表示 mdelays，第三列表示发送每条命令的 payload_lenth，后面几列都是表示每条命令的 payload。

常见数据类型

data type	description	packet size
0x03	Generic Short WRITE, no parameters	short
0x13	Generic Short WRITE, 1 parameters	short
0x23	Generic Short WRITE, 2 parameters	short
0x29	Generic long WRITE,	long
0x05	DCS Short WRITE, no parameters	short
0x15	DCS Short WRITE, 1 parameters	short

4.5.2 CSI 输出

1. 在 rk628.h 中配置

```
#define RK628_CSI_OUT
```

2. 在 config.c 中配置 csi host 输出类型：

```
const struct rk628_csi rk628_csi0 = {
        /* .csi_lanes = */ 4,
        /* .bus_format */ BUS_FMT_YUV422,
        /* .lane_mbps = */ 1300,
        /* .dual_mipi = */ FALSE,
        /* .continues_clk = */ TRUE,
};
```

属性说明

Property	Description	Option Value
dual_mipi	单双通道选择	TRUE / FALSE

注：除了表格中描述的属性可以根据 option value 做修改外，其他属性不建议修改。

4.5.3 LVDS 输出

1. 在 rk628.h 中配置

```
#define RK628_LVDS_OUT
```

2. 在 config.c 中配置 LVDS 输出类型：

```
uint32_t rk628_lvds_get_link_type(void)
{
    /*
     * LVDS_SINGLE_LINK
     * LVDS_DUAL_LINK_ODD_EVEN_PIXELS
     * LVDS_DUAL_LINK_EVEN_ODD_PIXELS
     * LVDS_DUAL_LINK_LEFT_RIGHT_PIXELS
     * LVDS_DUAL_LINK_RIGHT_LEFT_PIXELS
     */
    return LVDS_DUAL_LINK_EVEN_ODD_PIXELS;
}

uint32_t rk628_lvds_get_format(void)
{
    /*
     * LVDS_FORMAT_VESA_24BIT
     * LVDS_FORMAT_JEIDA_24BIT
     * LVDS_FORMAT_JEIDA_18BIT
     * LVDS_FORMAT_VESA_18BIT
     */
    return LVDS_FORMAT_VESA_24BIT;
}
```

属性说明

函数	return	description
rk628_lvds_get_link_type	LVDS_SINGLE_LINK LVDS_DUAL_LINK_ODD_EVEN_PIXELS LVDS_DUAL_LINK_EVEN_ODD_PIXELS LVDS_DUAL_LINK_LEFT_RIGHT_PIXELS LVDS_DUAL_LINK_RIGHT_LEFT_PIXELS	单通道 LVDS 双通道 LVDS，左右通道为奇偶通道 双通道 LVDS，左右通道为偶奇通道 双通道 LVDS，左右通道为左右屏 双通道 LVDS，左右通道为右左屏
rk628_lvds_get_format	LVDS_FORMAT_VESA_24BIT LVDS_FORMAT_JEIDA_24BIT LVDS_FORMAT_JEIDA_18BIT LVDS_FORMAT_VESA_18BIT	LVDS 数据格式

4.5.3.1 单 LVDS 输出

```
uint32_t rk628_lvds_get_link_type(void)
{
    return LVDS_SINGLE_LINK;
}
```

4.5.3.2 双 LVDS 输出

```
uint32_t rk628_lvds_get_link_type(void)
{
    return LVDS_DUAL_LINK_ODD_EVEN_PIXELS;
}
```

注：根据屏规格可以选择配置双通道 LVDS 的奇偶特性。

4.5.3.3 双 LVDS 左右屏

```
uint32_t rk628_lvds_get_link_type(void)
{
    return LVDS_DUAL_LINK_LEFT_RIGHT_PIXELS;
}
```

注：根据硬件设计可以选择配置双通道 LVDS 的左右输出。

4.5.4 GVI 输出

1. 在 rk628.h 中配置

```
#define RK628_GVI_OUT
```

2. 在 config.c 中配置 GVI 输出类型：

```
struct rk628_gvi gvi = {
    /* .bus_format = */ GVI_MEDIA_BUS_FMT_RGB888_1X24,
    /* .color_depth = */ COLOR_DEPTH_RGB_YUV444_24BIT,
    /* .lanes = */ 8,
    /* .division_mode = */ 0,
    /* .frm_rst = */ 0,
    /* .byte_mode = */ 4,
};
```

属性说明

Property	Description	Option Value
bus_format	GVI 数据格式	GVI_MEDIA_BUS_FMT_RGB666_1X18 GVI_MEDIA_BUS_FMT_RGB888_1X24 GVI_MEDIA_BUS_FMT_YUYV10_1X20 GVI_MEDIA_BUS_FMT_YUYV8_1X16 GVI_MEDIA_BUS_FMT_RGB101010_1X30
color_depth	色彩深度	COLOR_DEPTH_RGB_YUV444_18BIT COLOR_DEPTH_RGB_YUV444_24BIT COLOR_DEPTH_RGB_YUV444_30BIT COLOR_DEPTH_YUV422_16BIT COLOR_DEPTH_YUV422_20BIT
lanes	GVI lanes	1 / 2 / 4 / 8
division_mode	division 选择	1 / 2
byte_mode	byte mode	3 / 4

注：除了表格中描述的属性可以根据 option value 做修改外，其他属性不建议修改。

5. 固件编译

1. 下载 riscv64-unknown-elf-gcc-8.2.0-2019.05.3-x86_64-linux 编译工具
2. 设置环境变量及编译命令

   示例中 `../gcc/riscv64-unknown-elf-gcc-8.2.0-2019.05.3-x86_64-linux` 为编译器相对于工程目录的路径，根据编译器实际路径修改命令

   ```
   export CROSS_PATH=../gcc/riscv64-unknown-elf-gcc-8.2.0-2019.05.3-x86_64-linux/bin
   make clean && make
   ```

3. 编译完成后使用以下命令打包固件：

   ```
   cp build/rk628.bin firmware_merger/Image/
   cd firmware_merger && ./mkimage.sh
   ```

6. 固件下载

1. 固件下载通过 UART 进行的，如下图所示：

2. 进入下载模式

固件的下载是通过 UART_M0 进行的，如果此时板子内不存在固件，连接 UART_M0 (波特率 115200)，上电启动后，串口会不断打印 RKUART：

```
RKUARTRKUARTRKUARTRKUARTRKUARTRKUARTRKUARTRKUARTRKUARTRKUARTRKUARTRKUAR
TRKUARTRKUARTRKUARTRKUARTRKUARTRKUARTRKUARTRKUARTRKUARTRKUARTRKUARTRKUA
RTRKUARTRKUARTRKUARTRKUARTRKUARTRKUARTRKUARTRKUARTRKUARTRKUARTRKUARTRKU
ARTRKUARTRKUARTRKUARTRKUARTRKUARTRKUARTRKUARTRKUARTRKUARTRKUARTRKUARTRK
UARTRKUARTRKUARTRKUARTRKUARTRKUARTRKUARTRKUARTRKUARTRKUARTRKUARTRKUARTR
KUARTRKUARTRKUARTRKUARTRKUARTRKUARTRKUARTRKUARTRKUARTRKUARTRKUARTRKUART
RKUARTRKUARTRKUARTRKUARTRKUARTRKUARTRKUARTRKUARTRKUARTRKUARTRKUARTRKUAR
TRKUARTRKUARTRKUARTRKUARTRKUARTRKUARTRKUARTRKUARTRKUARTRKUARTRKUARTRKUA
RTRKUARTRKUARTRKUARTRKUARTRKUARTRKUARTRKUARTRKUARTRKUARTRKUARTRKUARTRKU
ARTRKUARTRKUARTRKUARTRKUARTRKUARTRKUARTRKUARTRKUARTRKUARTRKUARTRKUARTRK
UARTRKUARTRKUARTRKUARTRKUARTRKUARTRKUARTRKUARTRKUARTRKUARTRKUARTRKUARTR
KUARTRKUARTRKUARTRKUARTRKUARTRKUARTRKUARTRKUARTRKUARTRKUARTRKUARTRKUART
RKUARTRKUARTRKUARTRKUARTRKUARTRKUARTRKUARTRKUARTRKUARTRKUARTRKUARTRKUAR
TRKUARTRKUARTRKUARTRKUARTRKUARTRKUARTRKUARTRKUARTRKUARTRKUARTRKUARTRKUA
```

此时说明 UART_M0 工作正常，可以进行固件下载。

若板子中已经下载过固件了，需要长按板子背后的按键上电，则串口会不断打印 RKUART ，说明芯片已经进入下载模式。

3. 使用下载工具 upgradecomtool 进行固件下载

工具路径为：

```
firmware_merger/tools/UpgradeComTool_v1.22.rar
```

1. Com: 选择 UART_M0 连接的端口；
2. Buadrate: 波特率为 1500000；
3. boot 和 firmware: 分别选择打包固件后生成的 Hornet_upgrade.bin 和 Firmware.img，生成路径为：

   ```
   firmware_merger/Image/Hornet_upgrade.bin
   firmware_merger/Image/Firmware.img
   ```

烧写成功如图所示：

7. 调试方法

RK628 支持通过 UART 进行寄存器调试，通过 UART_M0 进行的，连接波特率为 115200。

1. 支持的命令有 ”d”: 读取寄存器值，“m”: 修改寄存器值，“i”：查看 RK628 信息，在串口出现 “: ” 后可以输入命令：

   ```
   :
   ```

2. 读取寄存器（以读取 CRU 相关寄存器为例，内部 MCU 模式基地址 0x800C0000，第三方 MCU I2C 访问模式基地址 0xC0000）：

   如下所示，输入 “d” 后 出现 “addr:”，即可输入需要读取的寄存器地址，会 dump 出以输入地址为基地址的 (base) ~ (base + 0x1fc) 的寄存器值

```
: d
dump mem
addr: 800c0000
800C0000: 00001063 00001442 00000000 00000007
800C0010: 00007F00 00000000 00000000 00000000
800C0020: 00001028 00000441 00F5C28F 00000007
800C0030: 00007F00 00000000 00000000 00000000
800C0040: 00001028 00000441 00F5C28F 00000007
800C0050: 00007F00 00000000 00000000 00000000
800C0060: 00000015 00000000 00000000 00000000
800C0070: 00000000 00000000 00000000 00000000
800C0080: 00008989 00000003 00000007 00004B80
800C0090: 03355460 00008257 00008004 00008030
800C00A0: 00002C00 00000000 00000000 00000000
800C00B0: 00285F58 00010002 00010002 00000103
800C00C0: 00000300 00000703 0000000B 00000008
800C00D0: 0001000A 00000B07 00000000 00000000
800C00E0: 00000000 00000000 00000000 00000000
800C00F0: 00000000 00000000 00000000 00000000
800C0100: 00000000 00000000 00000000 00000000
800C0110: 00000000 00000000 00000000 00000000
800C0120: 00000000 00000000 00000000 00000000
800C0130: 00000000 00000000 00000000 00000000
800C0140: 00000000 00000000 00000000 00000000
800C0150: 00000000 00000000 00000000 00000000
800C0160: 00000000 00000000 00000000 00000000
800C0170: 00000000 00000000 00000000 00000000
800C0180: 00000000 00000000 00000000 00000000
800C0190: 00000000 00000000 00000000 00000000
800C01A0: 00000000 00000000 00000000 00000000
800C01B0: 00000000 00000000 00000000 00000000
800C01C0: 00000000 00000000 00000000 00000000
800C01D0: 00000000 00000000 00000000 00000000
800C01E0: 00000000 00000000 00000000 00000000
800C01F0: 00000000 00000000 00000000 00000000
```

3. 修改寄存器（以 RK628E/H colorbar为例，内部 MCU 模式地址 0x80000010，第三方 MCU I2C 访问模式基地址 0x10）：

   如下所示，输入 “m” 后 出现 “addr:”，即可输入需要写入的寄存器地址，再输入需要写入寄存器的值。

```
: m
modify mem
addr: 80000010
80000010: 02000200
```