ATK-DLMP157-MIPI 调试手册

正点原子公司名称 ：广州市星翼电子科技有限公司
原子哥在线教学平台：www.yuanzige.com
开源电子网 / 论坛 ：http://www.openedv.com/forum.php
正点原子淘宝店铺 ：https://openedv.taobao.com
正点原子官方网站 ：www.alientek.com
正点原子 B 站视频 ：https://space.bilibili.com/394620890

电话：020-38271790 传真：020-36773971

请关注正点原子公众号，资料发布更新我们会通知。
请下载原子哥 APP，数千讲视频免费学习，更快更流畅。

扫码关注正点原子公众号

扫码下载“原子哥”APP

文档更新说明

版本	版本更新说明	负责人	校审	发布日期
V1.0	初稿：	正点原子 linux 团队	正点原子 linux 团队	

目录

前言

本文档主要是教大家如何，在 DLMP157 调试正点原子 5.5 寸的 720 屏。

免责声明

本文档所提及的产品规格和使用说明仅供参考，如有内容更新，恕不另行通知；除非有特殊约定，本文档仅作为产品指导，所作陈述均不构成任何形式的担保。

为了得到最新版本的产品信息，请用户定时访问正点原子资料下载中心或者与淘宝正点原子旗舰店客服联系索取。感谢您的包容与支持。

第一章 在 Uboot 点亮屏幕

1.1 设备树的修改

打开“arch/arm/dts/stm32mp157d-atk.dts”文件，

```
&dsi {
    #address-cells = <1>;
    #size-cells = <0>;
    status = "okay";

    ports {
        #address-cells = <1>;
        #size-cells = <0>;

        port@0 {
            reg = <0>;
            dsi_in: endpoint {
                remote-endpoint = <&ltdc_ep0_out>;
            };
        };

        port@1 {
            reg = <1>;
            dsi_out: endpoint {
                remote-endpoint = <&dsi_panel_in>;
            };
        };
    };

    panel_dsi: panel-dsi@0 {
        compatible = "himax,hx8394";
        reg = <0>;
        reset-gpios = <&gpioa 15 GPIO_ACTIVE_LOW>;
        backlight = <&panel_backlight>;
        power-supply = <&v3v3>;
        status = "okay";

        port {
            dsi_panel_in: endpoint {
                remote-endpoint = <&dsi_out>;
            };
        };
    };
```

```
};
```

Uboot 下的 dsi 节点是没有屏幕的相关的参数，因为已经写到驱动里面去了。dsi 节点不能和 panel_rgb 节点共存，所以如果使用了 MIPI 就需要屏蔽 panel_rgb 节点。

```
/*
    panel_rgb: panel-rgb {
        compatible = "simple-panel";
        backlight = <&panel_backlight>;
        status = "okay";

        port {
            panel_in_rgb: endpoint {
                remote-endpoint = <&ltdc_ep0_out>;
            };
        };
    };
*/
};
```

图 1.1.1 屏蔽 panel_rgb 节点

接着我们需要编写 ltdc 节点，此节点是控制 CPU 的显示数据，输出到 RGB 或者 MIPI 节点。所以我们需要修改 ltdc 节点指向 MIPI 节点。

```
&ltdc {
    status = "okay";
    pinctrl-names = "default", "sleep";
    pinctrl-0 = <&ltdc_pins_b>;
    pinctrl-1 = <&ltdc_pins_sleep_b>;

    port {
        #address-cells = <1>;
        #size-cells = <0>;

        ltdc_ep0_out: endpoint@0 {
            reg = <0>;
            remote-endpoint = <&dsi_in>;
        };
    };
};
```

上面的加粗的那一行，是指定到 dsi 节点里面。

1.2 驱动的编写。

1.2.1 驱动代码编写

开发板光盘 A-基础资料→01、程序源码→17、MIPI 屏幕→ himax-hx8394.c，拷贝文件到在 Uboot 的源码目录下“drivers/video/”下，也可以在此目录下创建 himax-hx8394.c 文件，输入以下示例代码：

```
#include <common.h>
#include <backlight.h>
#include <dm.h>
#include <mipi_dsi.h>
#include <panel.h>
#include <asm/gpio.h>
#include <power/regulator.h>

struct hx8394_panel_priv {
    struct udevice *reg;
    struct udevice *backlight;
    struct gpio_desc reset;
};

static const struct display_timing default_timing = {
    .pixelclock.typ     = 65000000,
    .hactive.typ        = 720,
    .hfront_porch.typ   = 48,
    .hback_porch.typ    = 52,
    .hsync_len.typ      = 8,
    .vactive.typ        = 1280,
    .vfront_porch.typ   = 16,
    .vback_porch.typ    = 15,
    .vsync_len.typ      = 5,
};

static void hx8394_dcs_write_buf(struct udevice *dev, const void *data,
                 size_t len)
{
    struct mipi_dsi_panel_plat *plat = dev_get_platdata(dev);
    struct mipi_dsi_device *device = plat->device;
    int err;

    err = mipi_dsi_dcs_write_buffer(device, data, len);
    if (err < 0)
        dev_err(dev, "MIPI DSI DCS write buffer failed: %d\n", err);
```

```
}


#define dcs_write_seq(ctx, seq...)            \
({                          \
    static const u8 d[] = { seq };            \
                            \
    hx8394_dcs_write_buf(ctx, d, ARRAY_SIZE(d));       \
})

static void hx8394_init_sequence(struct udevice *dev)
{
    dcs_write_seq(dev, 0XB9, 0xFF, 0x83, 0x94);
    dcs_write_seq(dev, 0X36, 0x1);
    dcs_write_seq(dev, 0XBA, 0X61, 0X03, 0X68, 0X6B, 0XB2, 0XC0);
    dcs_write_seq(dev, 0XB1, 0x48, 0x12, 0x72, 0x09, 0x32, 0x54,
                  0x71, 0x71, 0x57, 0x47);
    dcs_write_seq(dev, 0XB2, 0x00, 0x80, 0x64, 0x0C, 0x0D, 0x2F);
    dcs_write_seq(dev, 0XB4, 0x73, 0x74, 0x73, 0x74, 0x73, 0x74, 0x01,
                  0x0C, 0x86, 0x75, 0x00, 0x3F, 0x73, 0x74, 0x73,
0x74, 0x73, 0x74, 0x01, 0x0C, 0x86);
    dcs_write_seq(dev, 0XB6, 0x6E, 0x6E);
    dcs_write_seq(dev, 0XD3, 0x00, 0x00, 0x07, 0x07, 0x40, 0x07, 0x0C,
                  0x00, 0x08, 0x10, 0x08, 0x00, 0x08, 0x54, 0x15,
0x0A, 0x05, 0x0A, 0x02, 0x15, 0x06,
                  0x05, 0x06, 0x47, 0x44, 0x0A, 0x0A, 0x4B, 0x10,
0x07, 0x07, 0x0C, 0x40);
    dcs_write_seq(dev, 0XD5, 0x1C, 0x1C, 0x1D, 0x1D, 0x00, 0x01, 0x02,
                  0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A,
0x0B, 0x24, 0x25, 0x18, 0x18, 0x26,
                  0x27, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18,
0x18, 0x18, 0x18, 0x18, 0x18, 0x18,
                  0x18, 0x18, 0x18, 0x20, 0x21, 0x18, 0x18, 0x18,
0x18);
    dcs_write_seq(dev, 0XD6, 0x1C, 0x1C, 0x1D, 0x1D, 0x07, 0x06, 0x05,
                  0x04, 0x03, 0x02, 0x01, 0x00, 0x0B, 0x0A, 0x09,
0x08, 0x21, 0x20, 0x18, 0x18, 0x27,
                  0x26, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18,
0x18, 0x18, 0x18, 0x18, 0x18, 0x18,
                  0x18, 0x18, 0x18, 0x25, 0x24, 0x18, 0x18, 0x18,
0x18);
    dcs_write_seq(dev, 0XE0, 0x00, 0x0A, 0x15, 0x1B, 0x1E, 0x21, 0x24,
```

```
                    0x22, 0x47, 0x56, 0x65, 0x66, 0x6E, 0x82, 0x88,
0x8B, 0x9A, 0x9D, 0x98, 0xA8, 0xB9,
                    0x5D, 0x5C, 0x61, 0x66, 0x6A, 0x6F, 0x7F, 0x7F,
0x00, 0x0A, 0x15, 0x1B, 0x1E, 0x21,
                    0x24, 0x22, 0x47, 0x56, 0x65, 0x65, 0x6E, 0x81,
0x87, 0x8B, 0x98, 0x9D, 0x99, 0xA8,
                    0xBA, 0x5D, 0x5D, 0x62, 0x67, 0x6B, 0x72, 0x7F,
0x7F);
    dcs_write_seq(dev, 0xC0, 0x1F, 0x31);
    dcs_write_seq(dev, 0XCC, 0x03);
    dcs_write_seq(dev, 0xD4, 0x02);
    dcs_write_seq(dev, 0XBD, 0x02);
    dcs_write_seq(dev, 0xD8, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
0xFF, 0xFF, 0xFF, 0xFF, 0xFF);
    dcs_write_seq(dev, 0XBD, 0x00);
    dcs_write_seq(dev, 0XBD, 0x01);
    dcs_write_seq(dev, 0XB1, 0x00);
    dcs_write_seq(dev, 0XBD, 0x00);
    dcs_write_seq(dev, 0xBF, 0x40, 0x81, 0x50, 0x00, 0x1A, 0xFC, 0x01);
    dcs_write_seq(dev, 0xC6, 0xED);
}

static int hx8394_panel_enable_backlight(struct udevice *dev)
{
    struct mipi_dsi_panel_plat *plat = dev_get_platdata(dev);
    struct mipi_dsi_device *device = plat->device;
    struct hx8394_panel_priv *priv = dev_get_priv(dev);
    int ret;

    ret = mipi_dsi_attach(device);
    if (ret < 0)
        return ret;

    hx8394_init_sequence(dev);
    mdelay(120);
    ret = mipi_dsi_dcs_exit_sleep_mode(device);
    if (ret)
        return ret;

    ret = mipi_dsi_dcs_set_display_on(device);
    if (ret)
        return ret;
```

```
    mdelay(120);

    ret = backlight_enable(priv->backlight);
    if (ret)
        return ret;

    return 0;
}

static int hx8394_panel_get_display_timing(struct udevice *dev,
                    struct display_timing *timings)
{
    memcpy(timings, &default_timing, sizeof(*timings));

    return 0;
}

static int hx8394_panel_ofdata_to_platdata(struct udevice *dev)
{
    struct hx8394_panel_priv *priv = dev_get_priv(dev);
    int ret;

    if (IS_ENABLED(CONFIG_DM_REGULATOR)) {
        ret =  device_get_supply_regulator(dev, "power-supply",
                        &priv->reg);
        if (ret && ret != -ENOENT) {
            dev_err(dev, "Warning: cannot get power supply\n");
            return ret;
        }
    }

    ret = gpio_request_by_name(dev, "reset-gpios", 0, &priv->reset,
                GPIOD_IS_OUT);
    if (ret) {
        dev_err(dev, "Warning: cannot get reset GPIO\n");
        if (ret != -ENOENT)
            return ret;
    }

    ret = uclass_get_device_by_phandle(UCLASS_PANEL_BACKLIGHT, dev,
                    "backlight", &priv->backlight);
    if (ret) {
        dev_err(dev, "Cannot get backlight: ret=%d\n", ret);
```

```
        return ret;
    }

    return 0;
}

static int hx8394_panel_probe(struct udevice *dev)
{
    struct hx8394_panel_priv *priv = dev_get_priv(dev);
    struct mipi_dsi_panel_plat *plat = dev_get_platdata(dev);
    int ret;

    if (IS_ENABLED(CONFIG_DM_REGULATOR) && priv->reg) {
        ret = regulator_set_enable(priv->reg, true);
        if (ret)
            return ret;
    }

    /* fill characteristics of DSI data link */
    plat->lanes = 2;
    plat->format = MIPI_DSI_FMT_RGB888;
    plat->mode_flags = MIPI_DSI_MODE_VIDEO |
              MIPI_DSI_MODE_VIDEO_BURST |
              MIPI_DSI_MODE_LPM;

    return 0;
}

static const struct panel_ops hx8394_panel_ops = {
    .enable_backlight = hx8394_panel_enable_backlight,
    .get_display_timing = hx8394_panel_get_display_timing,
};

static const struct udevice_id hx8394_panel_ids[] = {
    { .compatible = "himax,hx8394" },
    { }
};

U_BOOT_DRIVER(hx8394_panel) = {
    .name           = "hx8394_panel",
    .id         = UCLASS_PANEL,
    .of_match       = hx8394_panel_ids,
    .ops            = &hx8394_panel_ops,
```

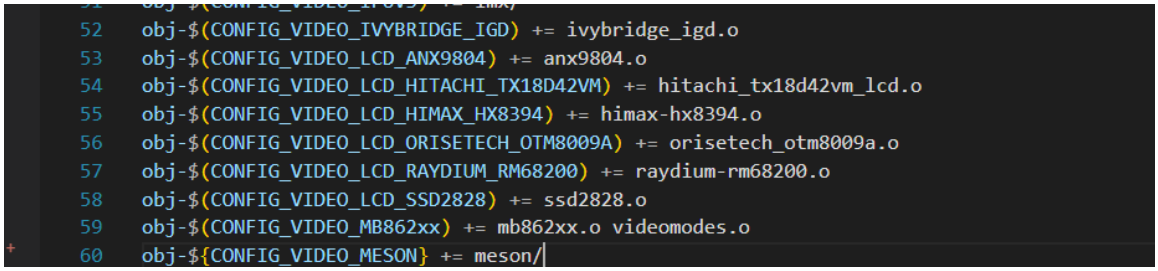
```
    .ofdata_to_platdata   = hx8394_panel_ofdata_to_platdata,
    .probe            = hx8394_panel_probe,
    .platdata_auto_alloc_size = sizeof(struct mipi_dsi_panel_plat),
    .priv_auto_alloc_size   = sizeof(struct hx8394_panel_priv),
};
```

1.2.2 把代码添加到 uboot 里面

在 uboot 的源码目录下打开“drivers/video/Makefile”在 54 行下，添加以下代码：

```
obj-$(CONFIG_VIDEO_LCD_HIMAX_HX8394) += himax-hx8394.o
```

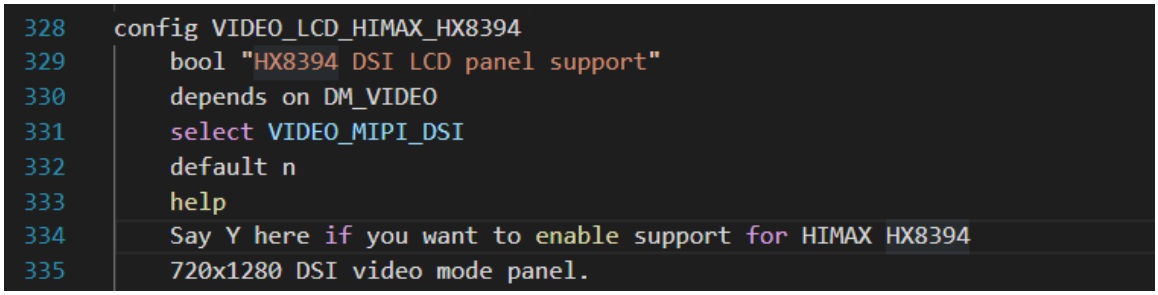
添加如下图所示：



接着去修改 Kconfig，打开“drivers/video/Kconfig”，在第 328 行，添加以下示例代码：

```
config VIDEO_LCD_HIMAX_HX8394
    bool "HX8394 DSI LCD panel support"
    depends on DM_VIDEO
    select VIDEO_MIPI_DSI
    default n
    help
    Say Y here if you want to enable support for HIMAX HX8394
    720x1280 DSI video mode panel.
```

添加结果如下图所示：



1.3 使能 HX8394 驱动

我们首先生成“.config”文件，运行命令如下所示：

```
make stm32mp157d_atk_defconfig
```

接着我们进入图形配置界面，按照以下路径配置 HX8394 把这个驱动编译进 uboot 里。

```
Device Drivers
    → Graphics support
        [*] HX8394 DSI LCD panel support
```

配置如下图所示：

```
[ ]   Support rotated displays
[ ]   Support a console that uses TrueType fonts
[ ] Display console as white on a black background
[ ] Skip framebuffer clear
    TrueType Fonts  ----
[ ] Use 'vidconsole' when 'lcd' is seen in stdout
[ ] Enable VESA video driver support
[ ] ANX9804 bridge chip
[*] HX8394 DSI LCD panel support
[*] OTM8009A DSI LCD panel support
[*] RM68200 DSI LCD panel support
[ ] SSD2828 bridge chip
[ ] Enable Amlogic Meson video support
[ ] Armada XP LCD controller
[ ] Enable EDID library
[ ] Enable Display support
[ ] Enable ATMEL video support using HLCDC
[ ] Enable Freescale Display Control Unit
[ ] Enable Rockchip Video Support  ----
[ ] Enable Arm Mali Display Processor support
```

配置完成后就我们就可以进行编译源码，这个时候我们可以看出 hx8394 编译进 uboot。

1.4 烧录测试

替换 u-boot.stm32 后，进入 uboot 命令终端，运行以下命令进行测试：

```
ext4load mmc 1:2 c4300000    alientek_480x272.bmp
bmp display c4300000
```

这里我的是出厂系统，所以在 emmc 里面第二个分区里面有 bmp 图片，选择 alientek_480x272.bmp 进行测试。

第二章 在 Linux 点亮屏幕

2.1 设备树的修改

dsi 节点和 uboot 节点是一样的，所以拷贝 1.1 小节的设备树。接着我们就可以修改 ltdc 节点。修改代码如下示例代码所示：

```
&ltdc {
    port {
        #address-cells = <1>;
        #size-cells = <0>;

        ltdc_ep1_out: endpoint@1 {
            reg = <1>;
            remote-endpoint = <&dsi_in>;
        };
```

```
    };
};
```

2.2 驱动的编写

2.2.1 驱动代码编译

发板光盘 A-基础资料→01、程序源码→17、MIPI 屏幕→ himax-hx8394.c，拷贝文件到 kernel 的源码目录下“drivers/gpu/drm/panel/”，也可以在此路径创建 panel-himax-hx8394.c 文件，panel-himax-hx8394.c 把以下代码拷贝到此文件里：

```
// SPDX-License-Identifier: GPL-2.0
/*
 * Authors: Wencong Liang <liangwc21@126.com>
 *
 */
#include <linux/backlight.h>
#include <linux/delay.h>
#include <linux/gpio/consumer.h>
#include <linux/module.h>
#include <linux/regulator/consumer.h>
#include <video/mipi_display.h>
#include <drm/drm_mipi_dsi.h>
#include <drm/drm_modes.h>
#include <drm/drm_panel.h>
#include <drm/drm_print.h>

struct hx8394 {
    struct device *dev;
    struct drm_panel panel;
    struct gpio_desc *reset_gpio;
    struct regulator *supply;
    struct backlight_device *backlight;
    bool prepared;
    bool enabled;
};

static const struct drm_display_mode default_mode = {
    .clock = 65000,
    .hdisplay = 720,
    .hsync_start =  720+ 52,
    .hsync_end = 720 + 52 + 8 ,
    .htotal = 720 + 52 + 8 +48 ,
    .vdisplay = 1280,
    .vsync_start = 1280 + 15,
```

```
    .vsync_end = 1280 + 15 + 6,
    .vtotal = 1280 + 15 + 6 + 16,
    .vrefresh = 60,
    .flags = DRM_MODE_FLAG_NHSYNC | DRM_MODE_FLAG_NVSYNC,
    .width_mm = 74,
    .height_mm = 150,
};


static inline struct hx8394 *panel_to_hx8394(struct drm_panel *panel)
{
    return container_of(panel, struct hx8394, panel);
}


static void hx8394_dcs_write_buf(struct hx8394 *ctx, const void *data,
                 size_t len)
{
    struct mipi_dsi_device *dsi = to_mipi_dsi_device(ctx->dev);
    int err;

    err = mipi_dsi_dcs_write_buffer(dsi, data, len);
    if (err < 0)
        DRM_ERROR_RATELIMITED("MIPI DSI DCS write buffer failed: %d\n",
                      err);
}

#define dcs_write_seq(ctx, seq...)            \
({                            \
    static const u8 d[] = { seq };            \
                            \
    hx8394_dcs_write_buf(ctx, d, ARRAY_SIZE(d));    \
})

static int hx8394_init_sequence(struct hx8394 *ctx)
{
    dcs_write_seq(ctx, 0XB9, 0xFF, 0x83, 0x94);
    dcs_write_seq(ctx, 0X36, 0x1);
    dcs_write_seq(ctx, 0XBA, 0X61, 0X03, 0X68, 0X6B, 0XB2, 0XC0);
    dcs_write_seq(ctx, 0XB1, 0x48, 0x12, 0x72, 0x09, 0x32, 0x54,
                  0x71, 0x71, 0x57, 0x47);
    dcs_write_seq(ctx, 0XB2, 0x00, 0x80, 0x64, 0x0C, 0x0D, 0x2F);
    dcs_write_seq(ctx, 0XB4, 0x73, 0x74, 0x73, 0x74, 0x73, 0x74, 0x01,
```

```
                    0x0C, 0x86, 0x75, 0x00, 0x3F, 0x73, 0x74, 0x73,
0x74, 0x73, 0x74, 0x01, 0x0C, 0x86);
    dcs_write_seq(ctx, 0XB6, 0x6E, 0x6E);
    dcs_write_seq(ctx, 0XD3, 0x00, 0x00, 0x07, 0x07, 0x40, 0x07, 0x0C,
                    0x00, 0x08, 0x10, 0x08, 0x00, 0x08, 0x54, 0x15,
0x0A, 0x05, 0x0A, 0x02, 0x15, 0x06,
                    0x05, 0x06, 0x47, 0x44, 0x0A, 0x0A, 0x4B, 0x10,
0x07, 0x07, 0x0C, 0x40);
    dcs_write_seq(ctx, 0XD5, 0x1C, 0x1C, 0x1D, 0x1D, 0x00, 0x01, 0x02,
                    0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A,
0x0B, 0x24, 0x25, 0x18, 0x18, 0x26,
                    0x27, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18,
0x18, 0x18, 0x18, 0x18, 0x18, 0x18,
                    0x18, 0x18, 0x18, 0x20, 0x21, 0x18, 0x18, 0x18,
0x18);
    dcs_write_seq(ctx, 0XD6, 0x1C, 0x1C, 0x1D, 0x1D, 0x07, 0x06, 0x05,
                    0x04, 0x03, 0x02, 0x01, 0x00, 0x0B, 0x0A, 0x09,
0x08, 0x21, 0x20, 0x18, 0x18, 0x27,
                    0x26, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18,
0x18, 0x18, 0x18, 0x18, 0x18, 0x18,
                    0x18, 0x18, 0x18, 0x25, 0x24, 0x18, 0x18, 0x18,
0x18);
    dcs_write_seq(ctx, 0XE0, 0x00, 0x0A, 0x15, 0x1B, 0x1E, 0x21, 0x24,
                    0x22, 0x47, 0x56, 0x65, 0x66, 0x6E, 0x82, 0x88,
0x8B, 0x9A, 0x9D, 0x98, 0xA8, 0xB9,
                    0x5D, 0x5C, 0x61, 0x66, 0x6A, 0x6F, 0x7F, 0x7F,
0x00, 0x0A, 0x15, 0x1B, 0x1E, 0x21,
                    0x24, 0x22, 0x47, 0x56, 0x65, 0x65, 0x6E, 0x81,
0x87, 0x8B, 0x98, 0x9D, 0x99, 0xA8,
                    0xBA, 0x5D, 0x5D, 0x62, 0x67, 0x6B, 0x72, 0x7F,
0x7F);
    dcs_write_seq(ctx, 0xC0, 0x1F, 0x31);
    dcs_write_seq(ctx, 0XCC, 0x03);
    dcs_write_seq(ctx, 0xD4, 0x02);
    dcs_write_seq(ctx, 0XBD, 0x02);
    dcs_write_seq(ctx, 0xD8, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
0xFF, 0xFF, 0xFF, 0xFF, 0xFF);
    dcs_write_seq(ctx, 0XBD, 0x00);
    dcs_write_seq(ctx, 0XBD, 0x01);
    dcs_write_seq(ctx, 0XB1, 0x00);
    dcs_write_seq(ctx, 0XBD, 0x00);
    dcs_write_seq(ctx, 0xBF, 0x40, 0x81, 0x50, 0x00, 0x1A, 0xFC, 0x01);
    dcs_write_seq(ctx, 0xC6, 0xED);
```

```
    return 0;
}


static int hx8394_disable(struct drm_panel *panel)
{
    struct hx8394 *ctx = panel_to_hx8394(panel);

    if (!ctx->enabled)
        return 0;

    backlight_disable(ctx->backlight);

    ctx->enabled = false;

    return 0;
}

static int hx8394_unprepare(struct drm_panel *panel)
{
    struct hx8394 *ctx = panel_to_hx8394(panel);
    struct mipi_dsi_device *dsi = to_mipi_dsi_device(ctx->dev);
    int ret;

    if (!ctx->prepared)
        return 0;

    ret = mipi_dsi_dcs_set_display_off(dsi);
    if (ret)
        DRM_WARN("failed to set display off: %d\n", ret);

    ret = mipi_dsi_dcs_enter_sleep_mode(dsi);
    if (ret)
        DRM_WARN("failed to enter sleep mode: %d\n", ret);

    msleep(120);

    regulator_disable(ctx->supply);

    ctx->prepared = false;

    return 0;
```

```
}

static int hx8394_enable(struct drm_panel *panel)
{
    struct hx8394 *ctx = panel_to_hx8394(panel);

  if (ctx->enabled)
        return 0;

    ctx->prepared = true;

    return 0;
}

static int hx8394_prepare(struct drm_panel *panel)
{
    struct hx8394 *ctx = panel_to_hx8394(panel);
    struct mipi_dsi_device *dsi = to_mipi_dsi_device(ctx->dev);
    int ret;

    if (ctx->enabled)
        return 0;

    ret = regulator_enable(ctx->supply);
    if (ret < 0) {
        DRM_ERROR("failed to enable supply: %d\n", ret);
        return ret;
    }

    hx8394_init_sequence(ctx);

    msleep(120);

    ret = mipi_dsi_dcs_exit_sleep_mode(dsi);
    if (ret)
        return ret;

    ret = mipi_dsi_dcs_set_display_on(dsi);
    if (ret)
        return ret;

    msleep(120);
```

```
    backlight_enable(ctx->backlight);

    ctx->enabled = true;

    return 0;
}

static int hx8394_get_modes(struct drm_panel *panel)
{
    struct drm_display_mode *mode;

    mode = drm_mode_duplicate(panel->drm, &default_mode);
    if (!mode) {
        DRM_ERROR("failed to add mode %ux%ux@%u\n",
             default_mode.hdisplay, default_mode.vdisplay,
             default_mode.vrefresh);
        return -ENOMEM;
    }
    drm_mode_set_name(mode);

    mode->type = DRM_MODE_TYPE_DRIVER | DRM_MODE_TYPE_PREFERRED;
    drm_mode_probed_add(panel->connector, mode);

    panel->connector->display_info.width_mm = mode->width_mm;
    panel->connector->display_info.height_mm = mode->height_mm;

    return 1;
}

static const struct drm_panel_funcs hx8394_drm_funcs = {
    .disable = hx8394_disable,
    .unprepare = hx8394_unprepare,
    .prepare = hx8394_prepare,
    .enable = hx8394_enable,
    .get_modes = hx8394_get_modes,
};


static int hx8394_probe(struct mipi_dsi_device *dsi)
{
    struct device *dev = &dsi->dev;
    struct hx8394 *ctx;
    int ret;
```

```
    ctx = devm_kzalloc(dev, sizeof(*ctx), GFP_KERNEL);
    if (!ctx)
        return -ENOMEM;

    ctx->reset_gpio = devm_gpiod_get_optional(dev, "reset",
GPIOD_OUT_LOW);
    if (IS_ERR(ctx->reset_gpio)) {
        ret = PTR_ERR(ctx->reset_gpio);
        dev_err(dev, "cannot get reset GPIO: %d\n", ret);
        return ret;
    }

    ctx->supply = devm_regulator_get(dev, "power");
    if (IS_ERR(ctx->supply)) {
        ret = PTR_ERR(ctx->supply);
        if (ret != -EPROBE_DEFER)
            dev_err(dev, "cannot get regulator: %d\n", ret);
        return ret;
    }

    ctx->backlight = devm_of_find_backlight(dev);
    if (IS_ERR(ctx->backlight))
        return PTR_ERR(ctx->backlight);

    mipi_dsi_set_drvdata(dsi, ctx);

    ctx->dev = dev;

    dsi->lanes = 2;
    dsi->format = MIPI_DSI_FMT_RGB888;
     dsi->mode_flags = MIPI_DSI_MODE_VIDEO | MIPI_DSI_MODE_VIDEO_BURST |
            MIPI_DSI_MODE_LPM | MIPI_DSI_CLOCK_NON_CONTINUOUS;

    drm_panel_init(&ctx->panel);
    ctx->panel.dev = dev;
    ctx->panel.funcs = &hx8394_drm_funcs;

    drm_panel_add(&ctx->panel);

    ret = mipi_dsi_attach(dsi);
    if (ret < 0) {
        dev_err(dev, "mipi_dsi_attach() failed: %d\n", ret);
```

```
        drm_panel_remove(&ctx->panel);
        return ret;
    }

    return 0;
}

static int hx8394_remove(struct mipi_dsi_device *dsi)
{
    struct hx8394 *ctx = mipi_dsi_get_drvdata(dsi);

    mipi_dsi_detach(dsi);
    drm_panel_remove(&ctx->panel);

    return 0;
}

static const struct of_device_id himax_hx8394_of_match[] = {
    { .compatible = "himax,hx8394" },
    { }
};
MODULE_DEVICE_TABLE(of, himax_hx8394_of_match);

static struct mipi_dsi_driver himax_hx8394_driver = {
    .probe = hx8394_probe,
    .remove = hx8394_remove,
    .driver = {
        .name = "panel-himax-hx8394",
        .of_match_table = himax_hx8394_of_match,
    },
};
module_mipi_dsi_driver(himax_hx8394_driver);

MODULE_AUTHOR("Wencong Liang <liangwc21@126.com>");
MODULE_DESCRIPTION("DRM Driver for Himax hx8394 MIPI DSI panel");
MODULE_LICENSE("GPL v2");
```

2.2.2 把代码添加到 kernel 里面

在 kernel 的源码目录下打开“drivers/gpu/drm/panel/Makefile”在最后添加以下代码：

```
obj-$(CONFIG_DRM_PANEL_HIMAX_HX8394) += panel-himax-hx8394.o
```

添加如下图所示：

```
37  obj-$(CONFIG_DRM_PANEL_TPO_TD028TTEC1) += panel-tpo-td028ttec1.o
38  obj-$(CONFIG_DRM_PANEL_TPO_TD043MTEA1) += panel-tpo-td043mtea1.o
39  obj-$(CONFIG_DRM_PANEL_TPO_TPG110) += panel-tpo-tpg110.o
40  obj-$(CONFIG_DRM_PANEL_TRULY_NT35597_WQXGA) += panel-truly-nt35597.o
41  obj-$(CONFIG_DRM_PANEL_HIMAX_HX8394) += panel-himax-hx8394.o
42
```

接着去修改 Kconfig，打开“drivers/gpu/drm/panel/Kconfig”，在第 328 行，添加以下示例代码：

```
config DRM_PANEL_HIMAX_HX8394
    tristate "HiMax Hx8394 panel driver"
    depends on OF
    depends on DRM_MIPI_DSI
    depends on BACKLIGHT_CLASS_DEVICE
    help
      Say Y here if you want to enable support for the HIMAX
      HX8394 controller for 720X1280 LCD panels with MIPI
          system interfaces.
```

修改结果如下图所示：

```
42  config DRM_PANEL_HIMAX_HX8394
43      tristate "HiMax Hx8394 panel driver"
44      depends on OF
45      depends on DRM_MIPI_DSI
46      depends on BACKLIGHT_CLASS_DEVICE
47      help
48        Say Y here if you want to enable support for the HIMAX
49        HX8394 controller for 720X1280 LCD panels with MIPI
50        system interfaces.
```

2.3 使能 HX8394 驱动

我们首先生成“.config”文件，运行命令如下所示：

```
make stm32mp1_atk_defconfig
```

接着我们进入图形配置界面，按照以下路径配置 HX8394 把这个驱动编译进 kernel。

```
Device Drivers
    →Graphics support
        → Display Panels
            → <*> HiMax Hx8394 panel driver
```

配置结果如下图所示：

```
< > ARM Versatile panel driver
< > Generic LVDS panel driver
<*> support for simple panels
<*> HiMax Hx8394 panel driver
< > Feiyang FY07024DI26A30-D MIPI-DSI LCD panel
< > Ilitek ILI9322 320x240 QVGA panels
< > Ilitek ILI9881C-based panels
< > Innolux P079ZCA panel
< > JDI LT070ME05000 WUXGA DSI panel
< > Kingdisplay kd097d04 panel
<M> Samsung LD9040 RGB/SPI panel
< > LG LB035Q024573 RGB panel
< > LG4573 RGB/SPI panel
< > NEC NL8048HL11 RGB panel
< > Novatek NT39016 RGB/SPI panel
< > Olimex LCD-OLinuXino panel
<*> Orise Technology otm8009a 480x800 dsi 2dl panel
< > OSD OSD101T2587-53TS DSI 1920x1200 video mode panel
< > Panasonic VVX10F034N00 1920x1200 video mode panel
< > Raspberry Pi 7-inch touchscreen panel
```

配置完成后就可以编译测试，替换 uImage 和设备树进行测试。