HiKey970

DRM Development Guide

Issue	01
Date	2018-03-11

Trademarks and Permissions

, HISILICON, and other HiSilicon icons are trademarks of HiSilicon Technologies Co., Ltd.

All other trademarks and trade names mentioned in this document are the property of their respective holders.

Notice

The purchased products, services and features are stipulated by the contract made between HiSilicon and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either express or implied.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

HiSilicon Technologies Co., Ltd.

Address: Huawei Industrial Base
Bantian, Longgang
Shenzhen 518129
People's Republic of China

Website: http://www.hisilicon.com

Email: support@hisilicon.com

Change History

Changes between document issues are cumulative. The latest document issue contains all the changes made in earlier issues.

Issue 01 (2018-03-11)

The first version.

Contents

1 Description

1.1 DISPLAY

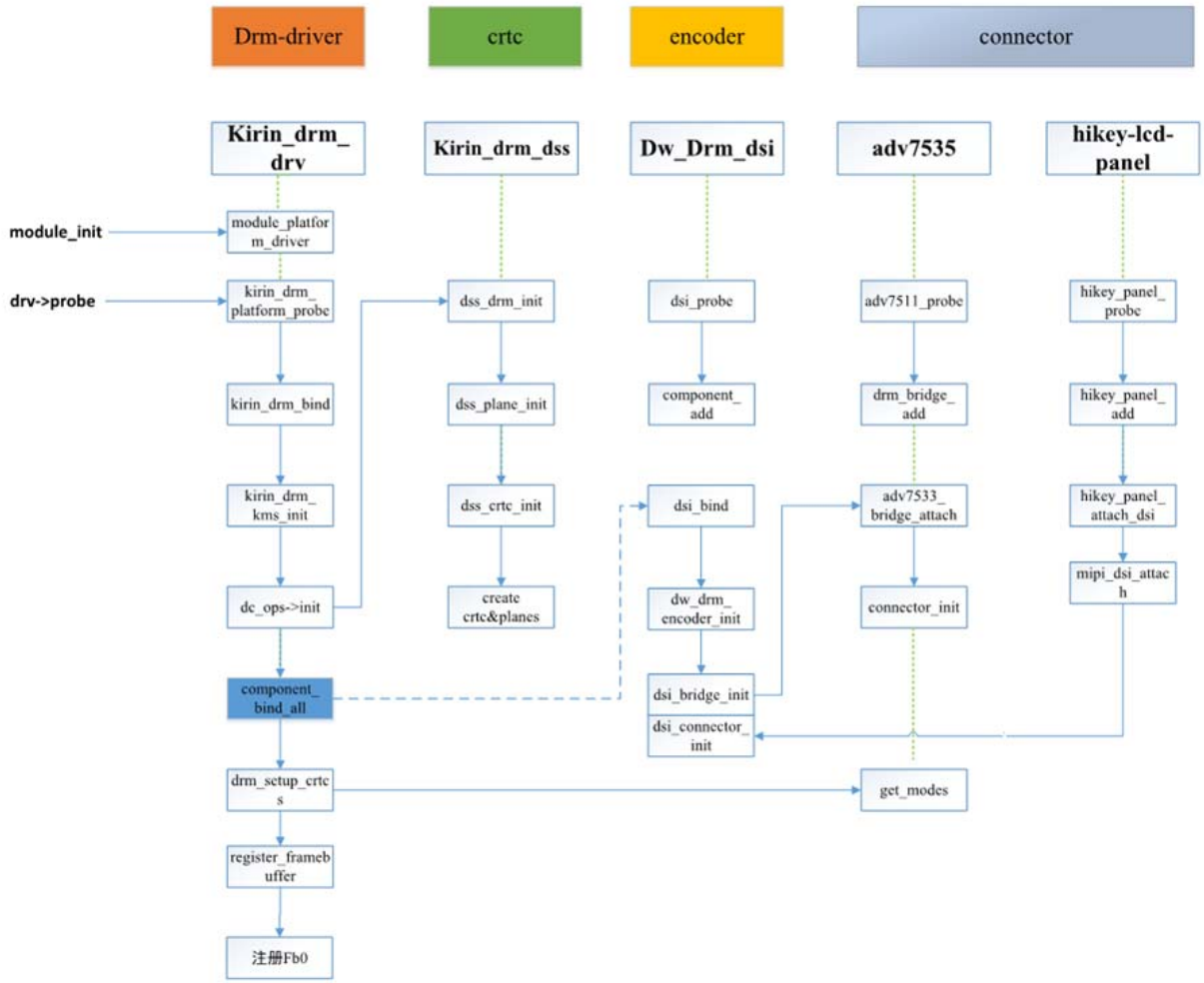
1.1.1 General description

The Display module complies with the standard DRM framework and implements the HDMI port and LCD display capabilities. It can accomplish the final display of the data through outputting MIPI signal via the DSI0 and switch all the way directly to the LCD display via the switch control, and the other way through the adv7535 MIPI signal into HDMI signal.

1.1.2 Features

Display characteristic:
1. Support open source DRM framework;
2. Support HDMI and LCD display;
3. Support HDMI multi-resolution switching;

1.2 DRM overall framework



Correspond with the Drm framework and the display path of each module; Initialize the crtc, encoder, and connector in turn according to the definition of the drm framework.

1.3 The implementation of DRM display driver design

1.3.1 Power on initialization through I2C

[Design Ideas]

When the I2C device is loaded, the system calls adv7535_probe. At this time, the structures such as adv7535 and dsi_lanes are initialized and obtained from the DTS node. The ADV7535 is powered through LDO1 and LDO3. The ICvdd and v1p2 are powered by the adv7535_init_regulators function and are respectively 1.8V and 1.2V. Regmap is initialized after the power supply is completed, and the IC is initialized by regmap;

[Design and Implementation]

Figure 1: ADV7535 functional block diagram

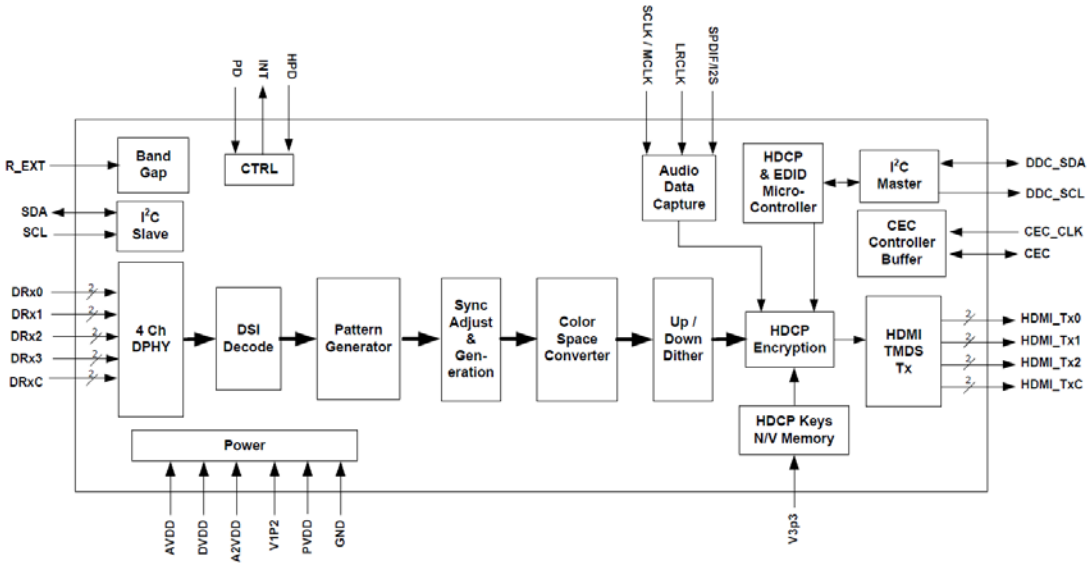


Figure 2 : The Configuration Description of Power Supply Parameter

Parameter	Test Conditions / Comments	Min	Typ	Max	Unit
POWER SUPPLIES					
AVDD	HDMI Tx analog power supply	1.71	1.8	1.89	V
A2VDD	MIPI/D-PHY analog power supply	1.71	1.8	1.89	V
DVDD	Digital and I/O power supply	1.71	1.8	1.89	V
PVDD	PLL power supply	1.71	1.8	1.89	V
V1P8	HDMI/DSI digital core power supply	1.71	1.8	1.89	V
V3P3	HDMI 3.3 V supply	3.14	3.3	3.46	V

Seen from the figure above, there are three external power supplies required for the normal operation of the adv7535. The 3.3V LDO has a long hardware supply.1.2V and 1.8V LDOs are required in the software.

```
v1p2-supply = <&ldo1>;
vdd-supply = <&ldo3>;
int adv7535_init_regulators(struct adi_hdmi *adv7535)
{
    struct device *dev = &adv7535->i2c_main->dev;
    adv7535->vdd = devm_regulator_get(dev, "vdd");
    adv7535->v1p2 = devm_regulator_get(dev, "v1p2");
 regulator_set_voltage(adv7535->vdd, 1800000, 1800000);
    regulator_set_voltage(adv7535->v1p2, 1200000, 1200000);
    /* keep the regulators always on */
    regulator_enable(adv7535->vdd);
regulator_enable(adv7535->v1p2);
}
```

ADV7535 initialization:
According to the requirements of adv7535_programming_guid, it needs to be initialized after the IC power supply is normal. It can be done directly in the software through regmap_register_patch.

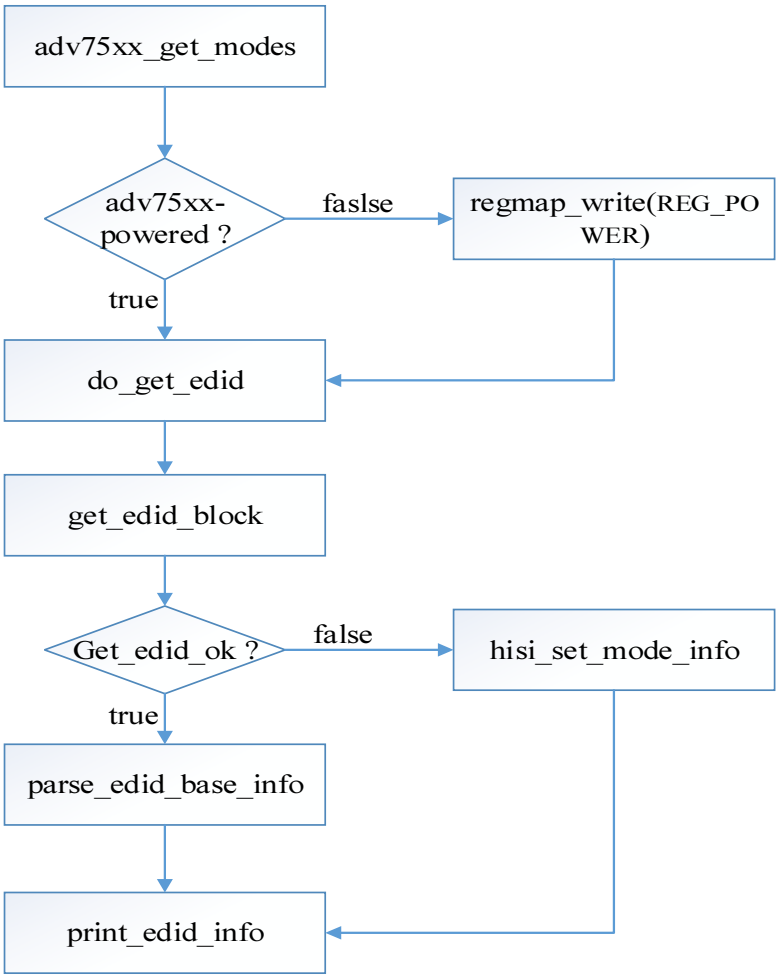
```
/* ADI recommended values for proper operation. */
static const struct reg_default adv7535_fixed_registers[] = {
    { 0x16, 0x20 },
    { 0x9a, 0xe0 },
    { 0xba, 0x70 },
    { 0xde, 0x82 },
    { 0xe4, 0x40 },
    { 0xe5, 0x80 },
};
```

```
regmap_register_patch(adv7535->regmap, adv7535_fixed_registers,
            ARRAY_SIZE(adv7535_fixed_registers));
```

1.3.2 Reading EDID via I2C

[Design Ideas]

Figure 3 Reading the EDID flow chart



Note: The above figure shows the edid reading process. When the edid can't be read correctly, you can use the hiti_set_mode_info function to manually assign a value to the display mode and use the general 1080*1920@60Hz to assign the corresponding parameter to ensure the most basic HDMI signal output. .

[Design Implementation]

Function: Reading EDID via i2c
Parameters: @ *data point to the structure adv7535
@ *buf points to the data space block that holds the EDID
@ block indicates that the EDID data segment is to be read (the base segment is 0; the extending segment is 1;)
@ len EDID data segment length EDID_LENGTH = 128 bytes

```
static int adv7535_get_edid_block(void *data, u8 *buf, unsigned int block,
                  size_t len)
{
    struct adi_hdmi *adv7535 = data;

        ret = regmap_read(adv7535->regmap, ADV7535_REG_DDC_STATUS,
                  &status);
```

```
        if (status != IDLE) {
            adv7535->edid_read = false;
            regmap_write(adv7535->regmap, ADV7535_REG_EDID_SEGMENT,
                    block);
            ret = adv7535_wait_for_edid(adv7535, 200);
        }

        /* Break this apart, hopefully more I2C controllers will
         * support 64 byte transfers than 256 byte transfers
         */
        xfer[0].addr = adv7535->i2c_edid->addr;
        xfer[0].flags = 0;
        xfer[0].len = 1;
        xfer[0].buf = &offset;
        xfer[1].addr = adv7535->i2c_edid->addr;
        xfer[1].flags = I2C_M_RD;
        xfer[1].len = 64;
        xfer[1].buf = edid_buf;

        offset = 0;
        for (i = 0; i < 4; ++i) {
            ret = i2c_transfer(adv7535->i2c_edid->adapter, xfer,
                       ARRAY_SIZE(xfer));
            if (ret < 0)
                return ret;
            else if (ret != 2)
                return -EIO;

            xfer[1].buf += 64;
            offset += 64;
        }

        adv7535->current_edid_segment = block;
    }

    if (block % 2 == 0)
        memcpy(buf, edid_buf, len);
    else
        memcpy(buf, edid_buf + EDID_LENGTH, len);

    return 0;
}
```

1.3.3 LDI/MIPI/HDMI Initialization of Adaptation Platform Based on EDID

[Design Ideas]

Each display device contains a specific EDID, which contains the resolution, pixel clock, blanking parameters, refresh rate, and other necessary information for display. We need to adapt the Soc platform to generate the correct timing based on these parameters;

Design Implementation

```
mode = adv7535->mode;
/* init hdmi display info */
pinfo = g_adi_hdmi_data.panel_info;
pinfo->xres = mode->hdisplay;
pinfo->yres = mode->vdisplay;
pinfo->width = mode->width_mm;
pinfo->height = mode->height_mm;
pinfo->orientation = PORTRAIT;
pinfo->bpp = RGB888;
pinfo->bgr_fmt = RGB;
pinfo->bl_set_type = BL_SET_BY_MIPI;
pinfo->type = PANEL_MIPI_VIDEO;

pinfo->pxl_clk_rate = mode->clock * 1000UL;
pinfo->ldi.h_back_porch = mode->htotal - mode->hsync_end;
pinfo->ldi.h_front_porch = mode->hsync_offset;
pinfo->ldi.h_pulse_width = mode->hsync_pulse_width;
pinfo->ldi.v_back_porch = mode->vtotal - mode->vsync_end;
pinfo->ldi.v_front_porch = mode->vsync_offset;
pinfo->ldi.v_pulse_width = mode->vsync_pulse_width;
```

[HDMI driver detailed design]

The HDMI driver function processing flow has the following sequence:
1. int __init adv7535_init(void); The module_init call is made during system startup and i2c-driven registration is completed.
2, int adv7535_probe (struct i2c_client * i2c, const struct i2c_device_id *id);When i2c device is loading, the system calls adv7535_probe to complete the matching of the device and the driver, the function needs to power on the adv7535, and then initialize the regmap, and initialize the key member variables of the adi_hdmi structure for assignment;

3, struct hisi_display_mode * adv7535_get_modes (struct adi_hdmi * adv7535); This function is used to obtain display related parameters, first obtain edid, then parse related information from edid;
4, int adv7535_mode_valid (struct hisi_display_mode *mode); This function is mainly used to determine whether the resolution of the display mode chip support;
5, void adv7535_mode_set (struct adi_hdmi * adv7535, struct hisi_display_mode * mode); This function is mainly set the minimum refresh rate and synchronization signal polarity based on mode info;

6, void adv7535_dsi_config_tgen (struct adi_hdmi * adv7535); The function is mainly configured timing-related parameters hsw, hfp, hbp, vsw, vfp, vbp;

1.4 HDMI display debug verification design

1.4.1 driver IC output test pattern(colorbar)

The adv7535 can output the test pattern in the test mode, which is used to test the chip configuration and the path between the display device and the device when the mipi input data is abnormal. The function can be opened by the macro definition switch by using in the software.

```
#define TEST_COLORBAR_DISPLAY

#ifdef TEST_COLORBAR_DISPLAY
    /* set pixel clock auto mode */
    regmap_write(adv7535->regmap_cec, ADV7535_REG_CEC_PIXEL_CLOCK_DIV,
            0x00);
#else
    /* set pixel clock divider mode */
    regmap_write(adv7535->regmap_cec, ADV7535_REG_CEC_PIXEL_CLOCK_DIV,
            clock_div_by_lanes[adv7535->num_dsi_lanes - 2] << 3);
#endif

#ifdef TEST_COLORBAR_DISPLAY
        /* reset internal timing generator */
        regmap_write(adv7535->regmap_cec, 0x27, 0xcb);
        regmap_write(adv7535->regmap_cec, 0x27, 0x8b);
        regmap_write(adv7535->regmap_cec, 0x27, 0xcb);
#else
        /* disable internal timing generator */
        regmap_write(adv7535->regmap_cec, 0x27, 0x0b);
#endif
```

1.4.2 DSS output ldi colorbar

You can configure the ldi register to output the colorbar after the device can display the driver IC's test pattern normally, which is used to test the transmission of the mipi signal. It is achieved by the following script command

```
#rem enable Idi signal output
db shell ecall reg_write_u32 0xe867d024 0xec1
#rem 0x00 Vertical stripes colorbar
adb shell ecall reg_write_u32 0xe867d028 0x0
pause
#rem 0x02 Horizontal stipes colorbar
```

```
adb shell ecall reg_write_u32 0xe867d028 0x2
pause
```

1.4.3 HDMI Display Android Interface

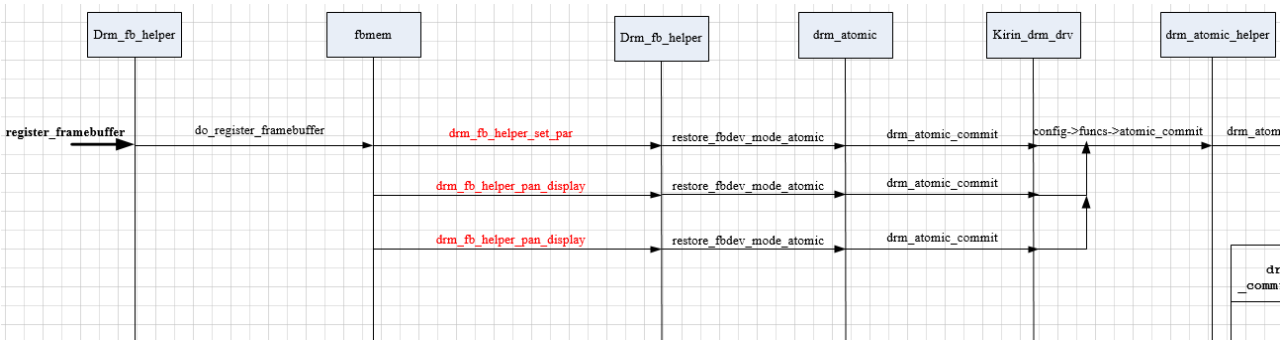
You can debug and display the Android system interface after the test pattern and ldi colorbar can be displayed normally. Mainly to adjust the pixel clock pxl_clk and mipi_dsi clock to ensure that each line of the line time hline_time and blanking time has_time, hbp_time is an integer:
Hsa_time = HSA*(PCLK period/Clk Lane Byte Period)
Hbp_time = HBP*(PCLK period/Clk Lane Byte Period)
Hline_time = (HSA+HBP+HACT+HFP)*(PCLK period/Clk Lane Byte Period)
Due to the platform clock frequency division limit, the software settings of pxl_clk and mipi_dsi_clk may be inconsistent with the actual generated, the clk need to be read out through the clk_get_rate function before the calculation, and then calculate it, which can reduce the error between the software configuration and the actual output of the hardware:

```
    pixel_clk = (uint64_t)clk_get_rate(hisifd->dss_pxl0_clk);

    hsa_time = pinfo->ldi.h_pulse_width * pinfo->dsi_phy_ctrl.lane_byte_clk
/ pixel_clk;
    hbp_time = pinfo->ldi.h_back_porch * pinfo->dsi_phy_ctrl.lane_byte_clk
/ pixel_clk;
    hline_time = (pinfo->ldi.h_pulse_width + pinfo->ldi.h_back_porch +
        rect.w + pinfo->ldi.h_front_porch) *
pinfo->dsi_phy_ctrl.lane_byte_clk / pixel_clk;
```

1.5 DRM frame delivery process

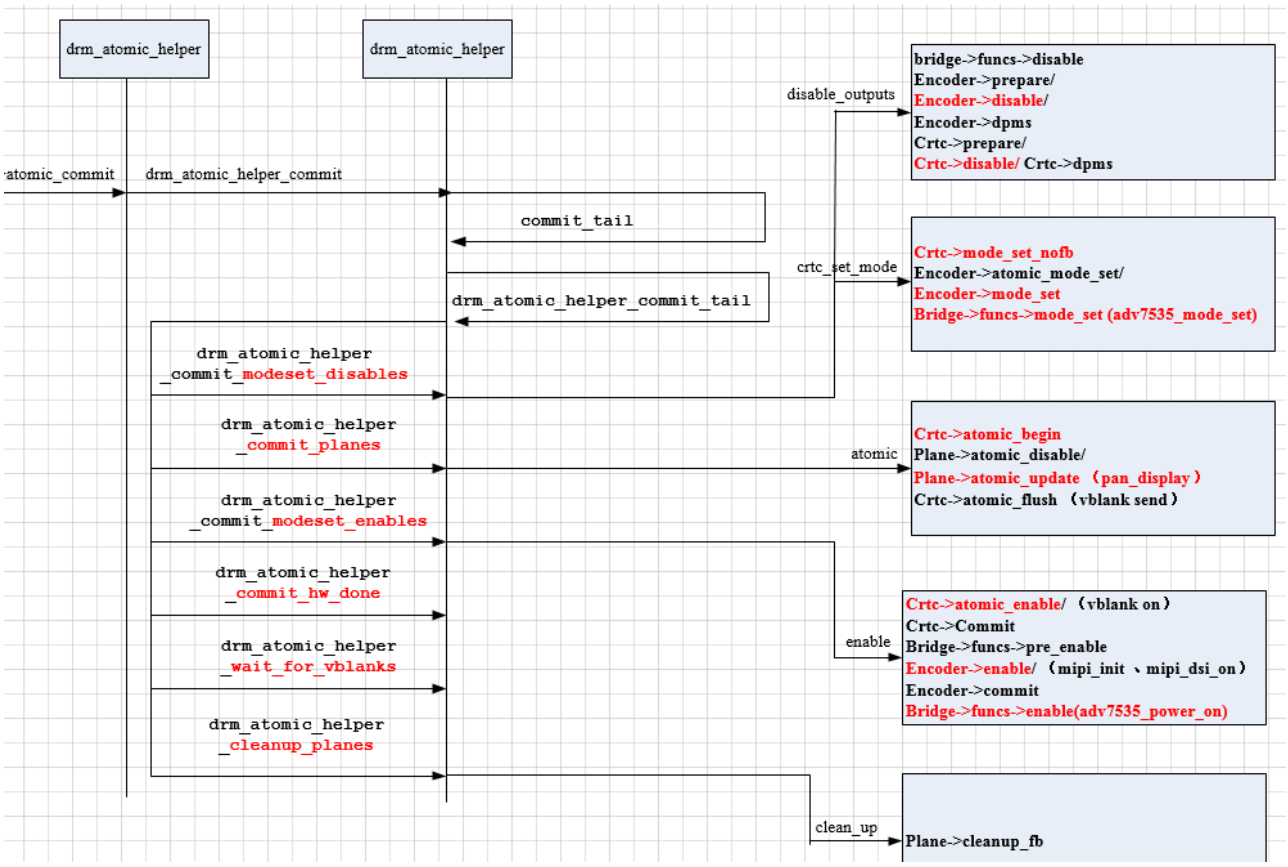
After the drm driver is registered, the kernel will call register_framebuffer to register fb0. After register fb, fb_set_par and fb_pan_display will be called by fbops, as shown in the following figure:




```
 #define DRM_FB_HELPER_DEFAULT_OPS \
.fb_check_var = drm_fb_helper_check_var, \
.fb_set_par = drm_fb_helper_set_par, \
```

```
.fb_setcmap = drm_fb_helper_setcmap, \
.fb_blank = drm_fb_helper_blank, \
.fb_pan_display = drm_fb_helper_pan_display, \
.fb_debug_enter = drm_fb_helper_debug_enter, \
.fb_debug_leave = drm_fb_helper_debug_leave, \
.fb_ioctl = drm_fb_helper_ioctl
```

1, There exists the following call relationship in the first fb_set_par:
(1) crtc_set_mode calls Crtc->mode_set_nofb to power on dss, set up ldi, dpe initialization and other operations;
(2) crtc_set_mode calls Encoder->mode_set (dsi_encoder_mode_set) to copy crtc's mode parameter information to encoder,encoder corresponds to hardware mipi_dsi here;
(3) crtc_set_mode call Bridge->funcs->mode_set (adv7535_mode_set) Set the timing parameters corresponding to adv7535;
(4) commit_planes calls Crtc->atomic_begin (powers up dss if dss is not powered);
(5) commit_planes calls Plane->atomic_update (dss_plane_atomic_update) to execute hisi_fb_pan_display;
(6) commit_planes calls Crtc->atomic_flush (dss_crtc_atomic_flush) to get the reference count of vblank and send vblank event after pageflip;
(7) modeset_enables calls Crtc->atomic_enable (dss_crtc_atomic_enable) to enable vblank event;
(8) modeset_enables calls Encoder->enable (dsi_encoder_enable) to initialize mipi dsi (including mipi dphy clk enable, mipi_init, mipi_dsi_on, etc);
(9) Modeset_enables calls Bridge->funcs->enable (adv7535_power_on) to power on the hdmi IC.



2,Since fb_set_par has already performed the initial setup and power-on for the corresponding module, the two subsequent fb_pan_display calls dss_crtc_atomic_begin --> dss_plane_atomic_update (hisi_fb_pan_display) --> dss_crtc_atomic_flush to refresh the display;