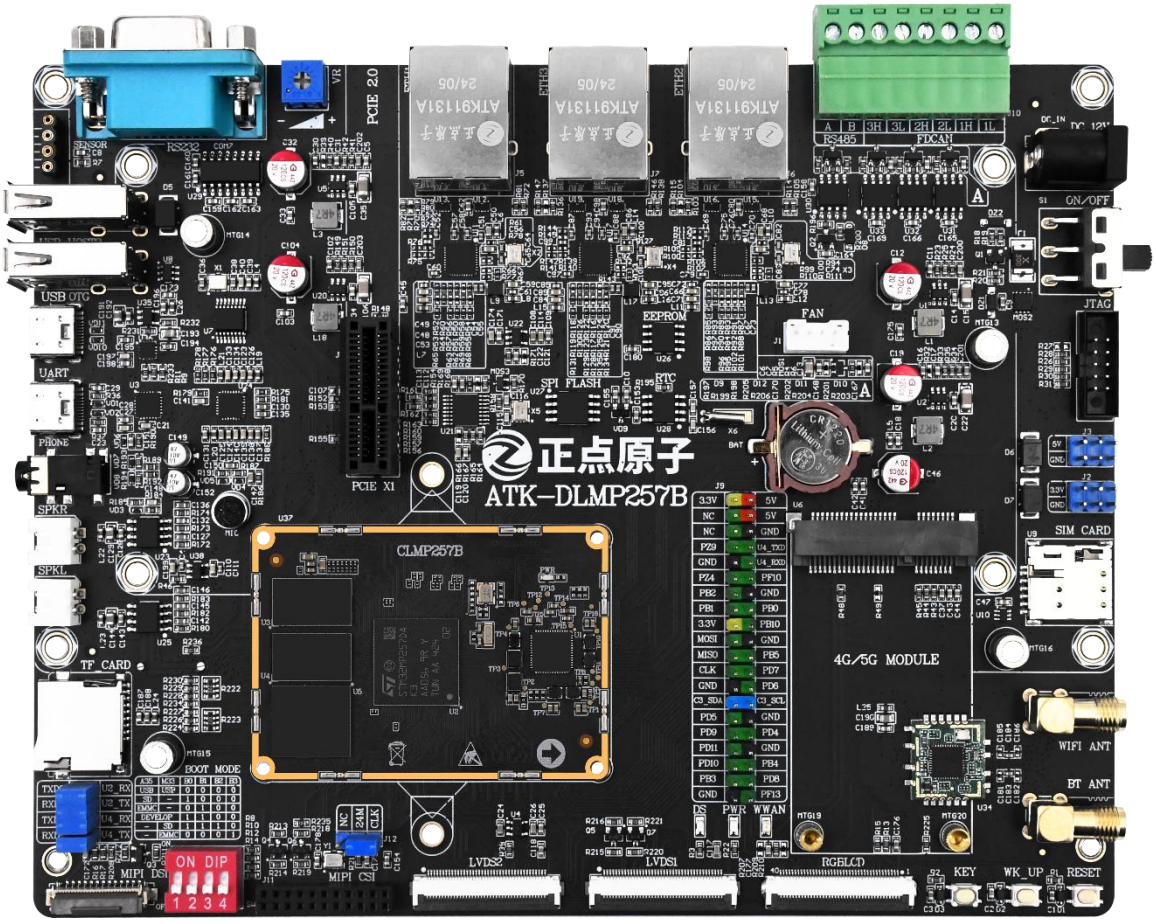
ATK-DLMP257B

Factory system screen ID identification Function Introduction

V1.1



ALIENTEK

1. Shopping:

TMALL: https://zhengdianyuanzi.tmall.com

TAOBAO: https://openedv.taobao.com

2. Download

Address: http://www.openedv.com/docs/index.html

3. FAE

Website : www.alientek.com

Forum : http://www.openedv.com/forum.php

Videos : www.yuanzige.com

Fax : +86 - 20 - 36773971

Phone : +86 - 20 - 38271790

Disclaimer

The product specifications and instructions mentioned in this document are for reference only and subject to update without prior notice; Unless otherwise agreed, this document is intended as a product guide only, and none of the representations made herein constitutes a warranty of any kind.

In order to get the latest version of product information, please regularly visit the download center or contact the customer service of Taobao ALIENTEK flagship store. Thank you for your tolerance and support.

Revision History:

Version	Version Update Notes	Responsible person	Proofreading	Date
V1.0	release officially	ALIENTEK	ALIENTEK	2025.04.01
V1.1	New touch chip FT5426\CST340 description	ALIENTEK	ALIENTEK	2025.4.30

Catalogue

Introduction....1
Chapter 1. RGB screen ID recognition function implementation....2
1.1 RGB screen ID setpoints....2
1.2 RGB screen ID identification source code implementation....3
1.2.1 The uboot source code....3
1.2.2 kernel source code....10
Chapter 2. MIPI screen ID recognition function realization....14
2.1 Setting value of MIPI screen ID....14
2.2 MIPI screen ID identification source code implementation....16
2.2.1 The uboot source code....16
2.2.2 kernel source code....21
Chapter 3. Summary....25

Introduction

There are a variety of RGB and MIPI interface LCD screens with different resolutions. When accessing the development board, different screen timing parameters need to be configured to display normally. For the convenience of users, there is no need to manually load different RGB and MIPI screen timing parameters. The ATK-DLMP257 development board adds the screen ID recognition function in the factory system. When users access the RGB screen or MIPI screen of different specifications produced by the ALIENTEK, the factory system can automatically obtain the screen ID, and then choose to load different screen timing parameters to realize normal display.

As a technical debugging note, this article will introduce the source code implementation process and source code location of this function. When the user accesses the screen produced by other manufacturers, it is not necessary to use this function, and the corresponding source code can be blocked or removed.

Chapter 1. RGB screen ID recognition function implementation

We produce a variety of RGB LCD capacitive touch screens, supporting different sizes and resolutions, respectively 4.3 inch 800x480, 7 inch 800x480, 7 inch 1024x600, 10.1 inch 1280x800. For specific screen parameters, you can download the corresponding screen information from the ALIENTEK data download center. This section introduces the principle of RGB screen ID recognition and the source code implementation process for reference.

1.1 RGB screen ID setpoints

The ALIENTEK RGB screen uses parallel 24-bit RGB interface as LCD display interface, supports RGB565, RGB888 color format, and communicates with the touch chip through the I2C interface to realize the touch function. Among them, the parallel 24-bit RGB interfaces are 8-bit RED data lines R0 to R7, 8-bit GREEN data lines G0 to G7, and 8-bit BLUE data lines B0 to B7.

ATK-DLMP257B development board uses parallel 16-bit RGB interface as the display interface, that is, R3-R7, G2-G7, B3-B7 a total of 16 signal pins, supporting RGB565 format.

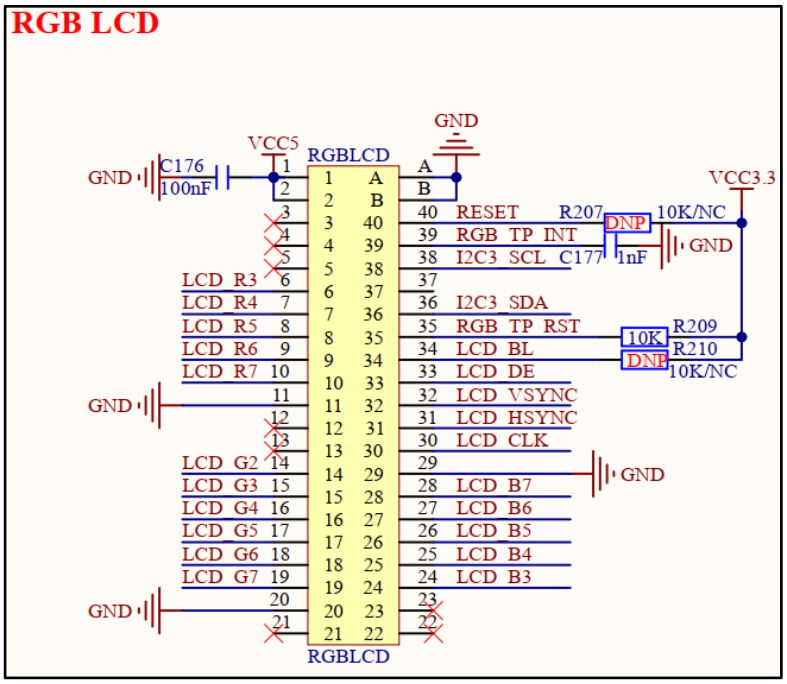


Figure 1.2.1-1 ATK-DLMP257B development board RGB screen interface definition

Many RGB screens of ALIENTEK support obtaining the ID setting value of RGB screen through three signal pins of R7, G7 and B7. The specific ID values are shown in the table below.

Table 1.1 Description Table of ALIENTEK RGB screen ID

B7（M2）	**G7**（M1）	**R7**（M0）	RGB screen model (size, resolution)
1	0	0	ATK-MD0430R-800480（4.3 inch 800x480）
0	0	1	ATK-MD0700R-800480（7 inch 800x480）
0	1	0	ATK-MD0700R-1024600（7 inch 1024x600）
1	0	1	ATK-MD1018R-1280800（10.1 inch 1280x800）

In the table, the value "1" represents the high level and the value "0" represents the low level. Different screen ID values are formed by the combination of B7\G7\R7 (M2|M1|M0) levels. These ID values are realized by the hardware of the ALIENTEK RGB screen by matching the pull-down resistor voltage on the three signal pins.

1.2 RGB screen ID identification source code implementation

In the previous subsection, the hardware ID values of the four RGB screens of the ALIENTEK have been obtained. Then the ID values formed by the combination of the three signal pins B7\G7\R7 of the RGB screen can be obtained in the system source code to realize the RGB screen ID recognition function. The following will introduce the source code implementation part of RGB screen ID recognition function, which is divided into two parts: uboot source code and kernel source code.

1.2.1 The uboot source code

Use vscode software to open the uboot source code, and search the macro definition "**ALIENTEK_MIPI_RGB_LCD**" globally in the uboot source code to view all source code implementations involving screen ID recognition functions. In the following, only the main code is analyzed and introduced.

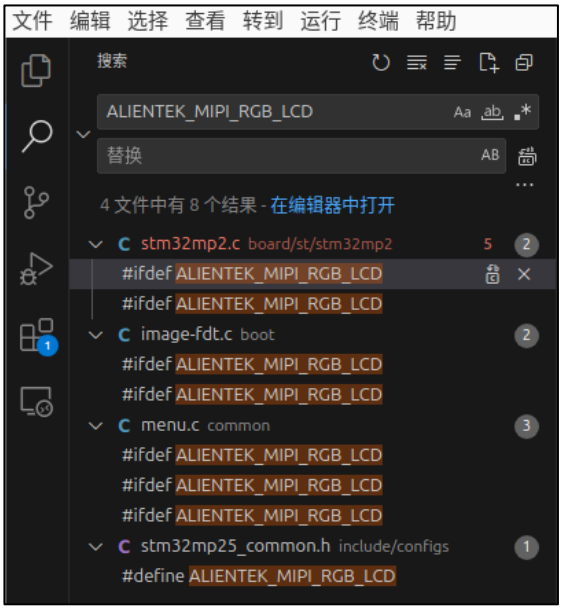


Figure 1.2.1-1 vscode searches for macro definitions

The macro definition in the uboot source files include/configs/stm32mp25_common h, can be open or shielding the macro definition, open or close the screen ID recognition.

```
#define ALIENTEK_MIPI_RGB_LCD
```

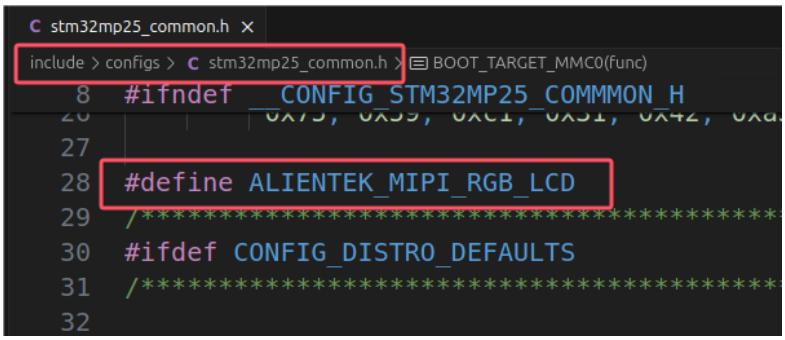


Figure 1.2.1-2 Macro definition path

Screen ID function source code segment mainly concentrated in the uboot source file board/st/stm32mp2 stm32mp2. C.

Enter the following code in board_late_init:

```
#ifdef ALIENTEK_MIPI_RGB_LCD
    int ret;
    ret = alientek_set_mipi_lcd();
    if(ret < 0 || ret == 1) { //no mipi lcd, detect rgb lcd
        alientek_set_rgb_lcd();
    }
#endif
```

The code first determines whether the MIPI screen produced by ALIENTEK has been connected. If the MIPI screen has been connected, only the MIPI screen configuration is carried out. If the MIPI screen is not connected, it is determined whether the RGB screen is connected.

When an RGB screen is plugged in, the following code is executed:

```
int alientek_set_rgb_lcd(void)
{
    ofnode node;
    int ret, i;
    struct gpio_desc priv_rgb[3];
    unsigned int read_id = 0;
    unsigned int rgb_lcd_id = 0;
    int ts_valid = -1;

    node = ofnode_path("/rgb_lcd_id_pinctrl");
    if(!ofnode_valid(node)) {
        printf("%s: cannot find /rgb_lcd_id_pinctrl node\n", __func__);
        return -1;
    }

    ret = gpio_request_by_name_nodev(node, "gpior", 0,&priv_rgb[0], GPIOD_IS_IN);
    if(ret) {
        printf("%s: cannot get GPIO: ret=%d\n", __func__, ret);
        return -1;
    }
```

```
ret = gpio_request_by_name_nodev(node, "gpiog", 0,&priv_rgb[1], GPIOD_IS_IN);
if(ret) {
        printf("%s: cannot get GPIO: ret=%d\n", __func__, ret);
        return -1;
}

ret = gpio_request_by_name_nodev(node, "gpiob", 0,&priv_rgb[2], GPIOD_IS_IN);
if(ret) {
        printf("%s: cannot get GPIO: ret=%d\n", __func__, ret);
        return -1;
}

ts_valid = detect_alientek_rgb_lcd_touchscreen();
if(!ts_valid) {
        for(i = 0; i < 3; i++) {
                ret = dm_gpio_get_value(&priv_rgb[i]);
                if(!ret)
                        read_id |= (0x1 << i);
        }
        //printf("alientek rgb_lcd_id: %d\n", read_id);
        switch(read_id) {
                case 1:     //atk_rgb_lcd_7_800x480
                        rgb_lcd_id = 1;
                        env_set("rgb_lcd_id", "1");
                        break;
                case 2:     //atk_rgb_lcd_7_1024x600
                        rgb_lcd_id = 2;
                        env_set("rgb_lcd_id", "2");
                        break;
                case 4:     //atk_rgb_lcd_4.3_800x480
                        rgb_lcd_id = 4;
                        env_set("rgb_lcd_id", "4");
                        break;
                case 5:     //atk_rgb_lcd_10.1_1280x800
                        rgb_lcd_id = 5;
                        env_set("rgb_lcd_id", "5");
                        break;
                default :
                        rgb_lcd_id = 0;
                        env_set("rgb_lcd_id", "0");
                        break;
        }
```

```
        } else {
            rgb_lcd_id = 0;
            env_set("rgb_lcd_id", "0");
            printf("Invalid detect_alientek_rgb_lcd_touchscreen\n");
        }

        gpio_free_list_nodev(&priv_rgb[0], 3);
        return rgb_lcd_id;
    }
```

The code first obtains three signal pin definitions of R7, G7 and B7 of RGB screen from the device tree, and configures the electrical properties of these three pins as input states, then reads the three pin levels, and then obtains the screen ID value by displacement combination processing of the pin levels, and finally determines which resolution size screen it belongs to. The env_set function is used to set the rgb_lcd_id environment variable value, which is the RGB screen ID value that will be used later. It is observed that the code obtains the touch chip ID value of the RGB capacitive touch screen through I2C communication before reading the three pin levels, and only reads the three pin levels when the touch chip ID value is successfully obtained, which indirectly improves the anti-interference ability.

uboot device tree file path is arch/arm/dts/stm32mp257d-atk-ddr-1GB.dts or arch/arm/dts/stm32mp257d-atk-ddr-2GB.dts (determined by the DDR memory capacity of the core board). The following RGB signal pin configuration is observed in the device tree file:

```
    rgb_lcd_id_pinctrl {
        gpior = <&gpiog 7       GPIO_ACTIVE_LOW>;     //LCD_R7
        gpiog = <&gpiog 13      GPIO_ACTIVE_LOW>;     //LCD_G7
        gpiob = <&gpioi 4  GPIO_ACTIVE_LOW>;          //LCD_B7
    };
```

RGB screen touch interface i2c3 configuration:

```
&i2c3 {
        pinctrl-names = "default", "sleep";
        pinctrl-0 = <&i2c3_pins_a>;
        pinctrl-1 = <&i2c3_sleep_pins_a>;
        status = "okay";
        /* spare dmas for other usage */
        /delete-property/dmas;
        /delete-property/dma-names;

        touchscreen@14 {
            compatible = "alientek,rgb-ts";
            reg = <0x14>; //GOODIX Address
            ft5426,cst340,reg = <0x38>;  //FT5426,CST340 Address
            interrupt-parent = <&gpiof>;
            interrupts = <11 IRQ_TYPE_EDGE_RISING>;
            irq-gpios = <&gpiof 11 GPIO_ACTIVE_HIGH>;
            reset-gpios = <&gpiof 13 GPIO_ACTIVE_HIGH>;
```

```
                status = "okay";
            };
        };
```

The following introduces the source code implementation of obtaining the touch chip ID value, and analyzes the function detect_alientek_rgb_lcd_touchscreen.

```
int detect_alientek_rgb_lcd_touchscreen(void)
{
    int i, ret;
    ofnode node;
    unsigned char goodix_id[GOODIX_ID_LEN] = {0};
    unsigned char ft5426_id[FT5426_ID_LEN] = {0};
    unsigned char cst340_id[CST340_ID_LEN] = {0};
    unsigned int goodix_addr = 0, ft_cst_addr = 0;
    char *goodix_id_list[] = {
        "911", "1151", "9271", "9147", "928", "1158",
    };

    node = ofnode_by_compatible(ofnode_null(), "alientek,rgb-ts");
    if(!ofnode_valid(node)) {
        printf("cannot find alientek,rgb-ts compatible\n");
        return -1;
    }

    ret = ofnode_read_u32(node, "reg", &goodix_addr);
    if (ret) {
        printf("cannot read goodix I2C address in %s\n", ofnode_get_name(node));
        return ret;
    }

    ofnode_read_u32(node, "ft5426,cst340,reg", &ft_cst_addr);

    ret = goodix_reset_irq_gpio(node);
    if(ret) {
        printf("error rgb lcd goodix_reset_irq_gpio\n");
        return -1;
    }

    ret = atk_i2c_read(node, goodix_addr, GOODIX_REG_ID, goodix_id, sizeof(goodix_id), 2);
    if (!ret) {
        for (i = 0; i < ARRAY_SIZE(goodix_id_list); i++) {
            if (!strncmp(goodix_id, goodix_id_list[i], sizeof(goodix_id))) {
                //printf("ts: Goodix (ID %s)\n", goodix_id);
                return 0;
```

```
                }
            }
        }

        if (ft_cst_addr != 0) {
            //FT5426
            ret = atk_i2c_read(node, ft_cst_addr, FT5426_REG_ID, ft5426_id, sizeof(ft5426_id), 1);
            if (!ret) {
                if (ft5426_id[0] == 0x54) {
                    //printf("ts: FocalTech FT5426 (ID 0x%X)\n", ft5426_id[0]);
                    return 0;
                }
            }
            //CST340
            ret = atk_i2c_read(node, ft_cst_addr, CST340_REG_ID, cst340_id, sizeof(cst340_id), 2);
            if (!ret) {
                if (cst340_id[0] == 0xa5) {
                    //printf("ts: Hynitron CST340 (ID 0x%x 0x%x)\n", cst340_id[0], cst340_id[1]);
                    return 0;
                }
            }
        }

        return -1;
}
```

This code mainly detects GT series touch chips from Goodix brand, such as GT911\GT1151\GT9271, etc., and two old RGB screen touch chips FT5426 and CST340 of ALIENTEK. Due to the different specifications of RGB screens produced, different touch chip models may be used, so after obtaining the touch chip ID value, several commonly used touch chip models are matched to improve code compatibility. If the touch chip ID value obtained by the user using his screen is not in this range, the corresponding model can be added in this code section.

As mentioned above, after the RGB screen ID value is obtained, the rgb_lcd_id environment variable value is set through the env_set function. This environment variable value is the RGB screen ID value. This value will have two important uses: 1) pass to the menu device tree selection menu of uboot, which is used to select the device tree to start the RGB screen; 2) It is passed to the kernel device tree, so that the kernel RGB screen driver can select the timing parameters of the corresponding RGB screen to realize normal display.

To accomplish the first of these uses, modify the uboot source file common/menu.c to run the code in the menu_default_set function:

```
    #ifdef ALIENTEK_MIPI_RGB_LCD
        int dsi_timings_id;
        int rgb_timings_id;
```

```
        dsi_timings_id = (int)((*(env_get("dsi_lcd_id"))) - '0');
        if (dsi_timings_id == 2 || dsi_timings_id == 3 || dsi_timings_id == 4) {
            m->default_item = menu_item_by_key(m, "3");
        } else { //no mipi lcd, detect rgb lcd
            rgb_timings_id = (int)((*(env_get("rgb_lcd_id"))) - '0');
            if (rgb_timings_id == 1 || rgb_timings_id == 2 || rgb_timings_id == 4
                || rgb_timings_id == 5) {
                m->default_item = menu_item_by_key(m, "2");
            }
        }
    #endif
```

This code is used to set the startup item in the menu device tree selection menu. When the MIPI screen is detected, the third device tree startup item is selected, and when the RGB screen is detected, the second device tree startup item is selected.

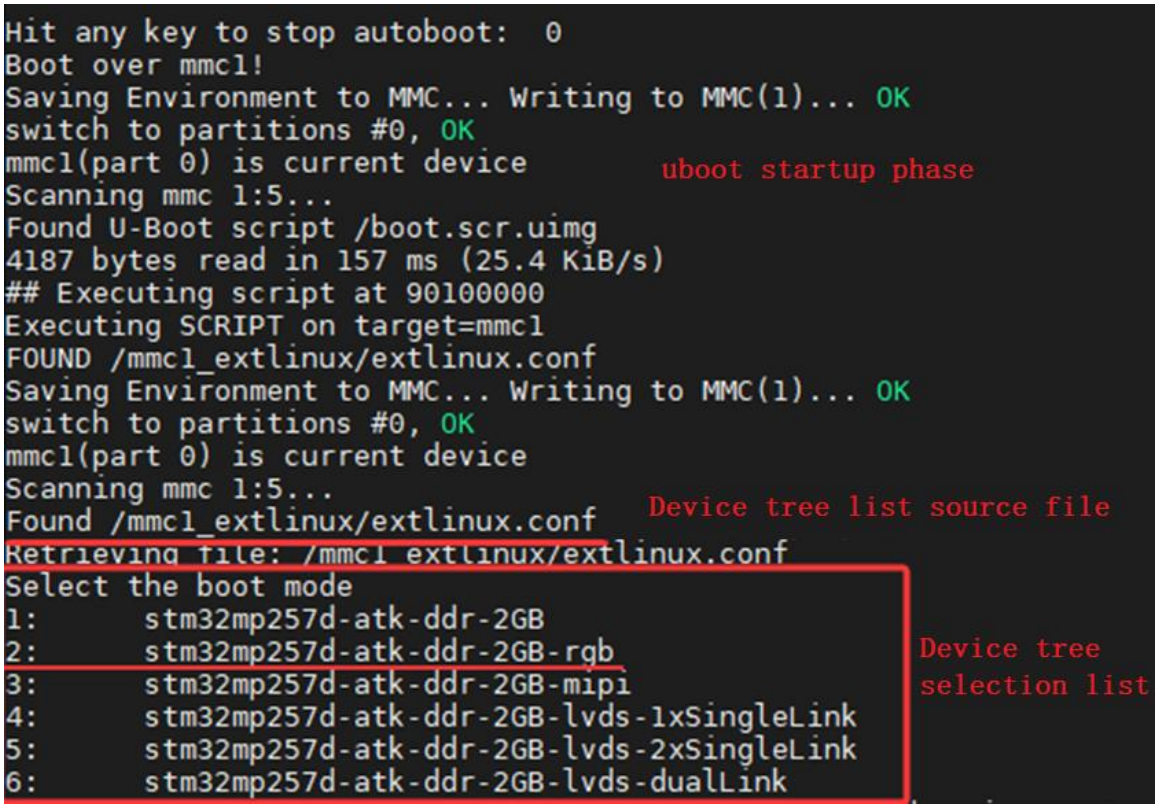


Figure 1.2.1-3 Device tree selection list

To implement the second use, modify the uboot source file boot/image-fdt.c to execute the code in the boot_relocate_fdt function:

```
    #ifdef ALIENTEK_MIPI_RGB_LCD
        dsi_timings_id = (int)((*(env_get("dsi_lcd_id"))) - '0');
        if (dsi_timings_id == 2 || dsi_timings_id == 3 || dsi_timings_id == 4) {
            nodeoffset_dsi = fdt_path_offset(fdt_blob, "/dsi_lcd_id");
            if (nodeoffset_dsi < 0) {
                printf("cannot find /dsi_lcd_id in linux kernel fdtfile\n");
```

```
            }
            fdt_setprop_u32(fdt_blob, nodeoffset_dsi, "dsi_select_id", dsi_timings_id);
        } else { //no mipi lcd, detect rgb lcd
            rgb_timings_id = (int)((*(env_get("rgb_lcd_id"))) - '0');
            if (rgb_timings_id == 1 || rgb_timings_id == 2 || rgb_timings_id == 4
                || rgb_timings_id == 5) {
                nodeoffset_rgb = fdt_path_offset(fdt_blob, "/rgb_lcd_id");
                if (nodeoffset_rgb < 0) {
                    printf("cannot find /rgb_lcd_id in linux kernel fdtfile\n");
                }
                fdt_setprop_u32(fdt_blob, nodeoffset_rgb, "rgb_select_id", rgb_timings_id);
            }
        }
    #endif
```

This code checks whether there is an "rgb_lcd_id" node in the loaded kernel device tree file. If there is an "rgb_lcd_id" node, it modifies its "rgb_select_id" attribute parameter value to the RGB screen ID value obtained by uboot. The value of the rgb_select_id attribute in the kernel device tree is then set to the latest RGB screen ID value, causing the kernel RGB screen driver to load the corresponding screen time sequence.

At this point, the RGB screen ID recognition function in the uboot source code is introduced, and the kernel source code involves screen ID recognition.

1.2.2 kernel source code

The RGB screen ID recognition part of the kernel source code is divided into the kernel device tree and the RGB screen driver.

The kernel device tree file path of the arch/arm64 / boot/DTS/st/stm32mp257d-atk-ddr- **1GB-rgb**. DTS or arch/arm64 / boot/DTS/st/stm32mp257d-atk-ddr-**2GB-rgb**.dts (determined by the core board DDR memory capacity). Here's part of the device tree:

```
    rgb_lcd_id {
            rgb_select_id = <2>;        //default, atk_rgb_lcd_7_1024x600
        };

        panel_rgb: panel-rgb {
            compatible = "panel-dpi";
            pinctrl-names = "default", "sleep";
            pinctrl-0 = <&ltdc_pins_a>;
            pinctrl-1 = <&ltdc_sleep_pins_a>;
            backlight = <&panel_rgb_backlight>;
            status = "okay";

            data-mapping = "rgb565";

            //atk_rgb_lcd_7_800x480
```

```
        panel-timing-1 {
            clock-frequency = <33300000>;
            hactive = <800>;
            vactive = <480>;
            hback-porch = <46>;
            hfront-porch = <210>;
            hsync-len = <1>;
            vback-porch = <23>;
            vfront-porch = <22>;
            vsync-len = <1>;
        };

        //atk_rgb_lcd_7_1024x600
        panel-timing-2 {
            clock-frequency = <51200000>;
            hactive = <1024>;
            vactive = <600>;
            hback-porch = <140>;
            hfront-porch = <160>;
            hsync-len = <20>;
            vback-porch = <20>;
            vfront-porch = <12>;
            vsync-len = <3>;
        };

        //atk_rgb_lcd_4.3_800x480
        panel-timing-4 {
            clock-frequency = <31000000>;
            hactive = <800>;
            vactive = <480>;
            hback-porch = <88>;
            hfront-porch = <40>;
            hsync-len = <48>;
            vback-porch = <32>;
            vfront-porch = <13>;
            vsync-len = <3>;
        };

        //atk_rgb_lcd_10.1_1280x800
        panel-timing-5 {
            clock-frequency = <71100000>;
            hactive = <1280>;
            vactive = <800>;
```

```
                hback-porch = <80>;
                hfront-porch = <70>;
                hsync-len = <10>;
                vback-porch = <10>;
                vfront-porch = <10>;
                vsync-len = <3>;
            };

            port {
                rgb_panel_in: endpoint {
                    remote-endpoint = <&ltdc_ep0_out>;
                };
            };
        };
```

In the above content, you can see that the "rgb_lcd_id" node and the attribute parameter "rgb_select_id" are set to 2, that is, the default is set to the ALIENTEK 7 inch RGB screen 1024x600, which can be updated by the screen ID value obtained by uboot. Then the "panel_rgb" node lists a number of ALIENTEK RGB screen timing parameters, which are selected and loaded by the RGB driver source code.

RGB screen driver source code is located in the drivers/gpu/DRM/panel/panel - simple. C, performs the screen ID in function panel_dpi_probe processing code, according to the ID value to select the loading screen timing parameters.

```
static int panel_dpi_probe(struct device *dev,
                           struct panel_simple *panel)
{
    ……
    u32 tmp = 2;
    char result[20];
    struct device_node *np_id = NULL;
    np_id = of_find_node_by_name(NULL, "rgb_lcd_id");
    of_property_read_u32(np_id, "rgb_select_id", &tmp);
    …….
    sprintf(result, "%s-%d", "panel-timing", tmp);
    ret = of_get_display_timing(np, result, timing);
    ……
}
```

The preceding code takes the value of the "rgb_select_id" attribute parameter, combines it with the string "panel-timing" to form a specified identifier, such as "panel-timing-2," and then uses the of_get_display_timing function to load the appropriate screen timing parameter in the device tree.

In addition, for the RGB screen touch chip driver source code part, Goodix GT series touch chip model, as well as FT5426 and CST340 models (both compatible) are supported, and the device tree configuration needs to choose one of two:

```
&i2c3 {
```

```
    //ft5426, cst340 touchscreen
    /*
    ft5426_ts@38 {
        compatible = "edt,edt-ft5206";
        reg = <0x38>;
        interrupt-parent = <&gpiof>;
        interrupts = <11 IRQ_TYPE_EDGE_RISING>;
        irq-gpios = <&gpiof 11 GPIO_ACTIVE_HIGH>;
        reset-gpios = <&gpiof 13 GPIO_ACTIVE_LOW>;
        status = "okay";
    };
    */

    //goodix touchscreen
    goodix_ts@14 {
        compatible = "goodix,gt9271", "goodix,gt911";
        reg = <0x14>;
        pinctrl-names = "default";
        interrupt-parent = <&gpiof>;
        interrupts = <11 IRQ_TYPE_EDGE_RISING>;
        irq-gpios = <&gpiof 11 GPIO_ACTIVE_HIGH>;
        reset-gpios = <&gpiof 13 GPIO_ACTIVE_HIGH>;
        status = "okay";
    };
};
```

By default, Goodix GT series touch chip is supported. If the user's own RGB screen touch chip is FT5426 or CST340, Goodix configuration needs to be blocked and FT5426 configuration can be enabled.

At this point, the RGB screen ID recognition function of the kernel source code implementation part is introduced.

Chapter 2. MIPI screen ID recognition function realization

We produce a variety of MIPI capacitively touch screens with different sizes and resolutions, including 5.5 "720x1280, 5.5" 1080x1920 and 10.1 "800x1280. For specific screen parameters, you can download the corresponding screen information from the ALIENTEK data download center. This section introduces the principle of MIPI screen ID recognition and the source code implementation process for reference.

2.1 Setting value of MIPI screen ID

MIPI screens produced by ALIENTEK are distinguished between different screens by setting the eighth pin of the screen interface as the screen ID pin LCD_ID. The pin has been pulled up and down in the MIPI screen backplane (resistance accuracy 1%), and the LCD_ID pin voltage of different resolution screens is different.

The voltage division of the LCD_ID pin of the ALIENTEK MIPI screen is as follows:

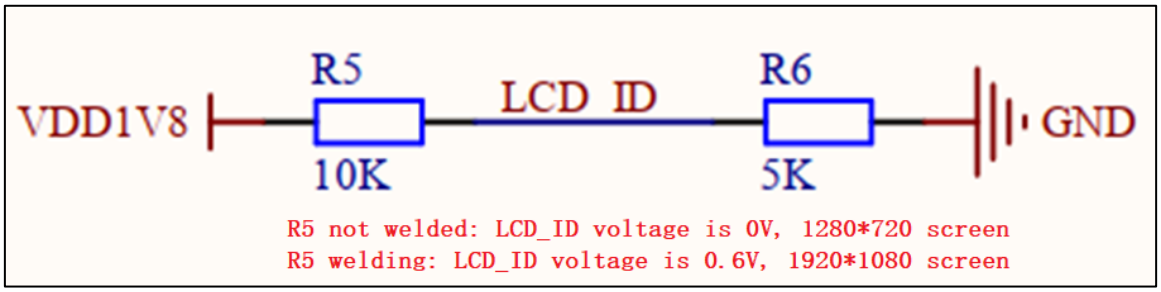


Figure 1.2.2-15.5 inch MIPI screen LCD ID pin voltage

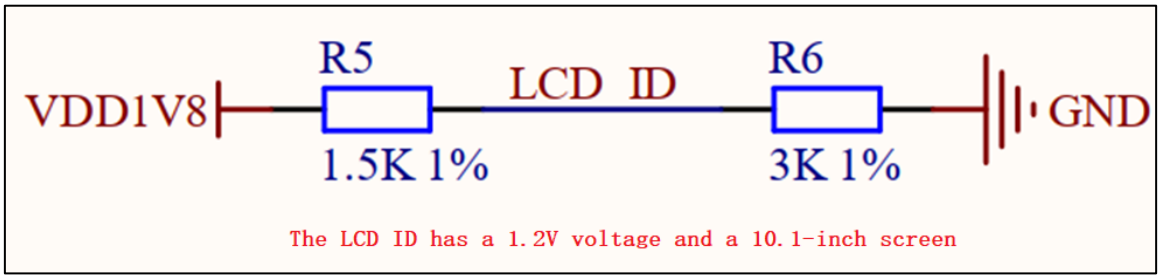


Figure 1.2.2-210.1 "MIPI screen LCD_ID pin voltage

According to the above figure, the MIPI screen LCD_ID pin voltage produced by ALIENTEK is shown in the table below.

Table 2.1.1 MIPI screen LCD_ID pin voltage description table

LCD ID pin voltage	MIPI screen model (size, resolution)
0V	ATK-MD0550-7201280（5.5 inch 720x1280）
0.6V	ATK-MD0550-10801920（5.5 inch 1080x1920）
1.2V	ATK-MD1001-8001280（10.1 inch 800x1280）

Therefore, we only need to use a pin with ADC function to collect the voltage of the LCD_ID pin of the screen, and the screen ID voltage value can be obtained to realize the differentiation of MIPI screens.

The following analysis of the ATK-DLMP257B development board MIPI screen interface design, the schematic diagram is shown below.

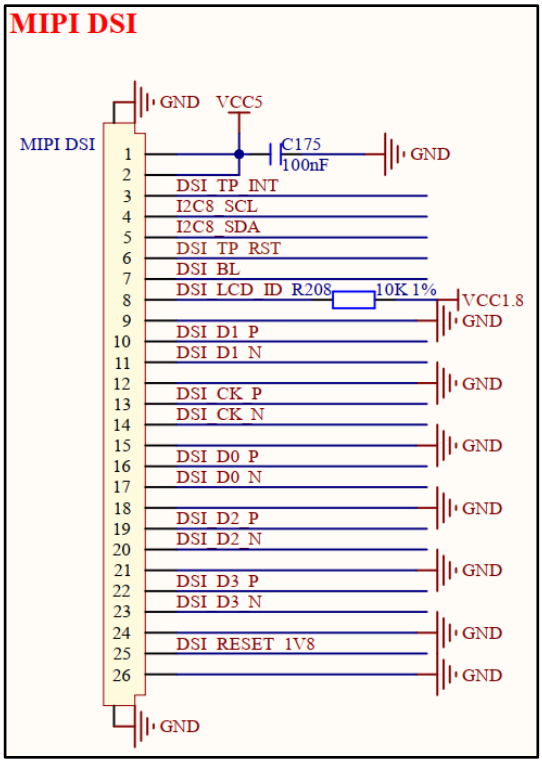


Figure 1.2.2-3 ATK-DLMP257B development board MIPI screen interface definition

In the figure above, the eighth pin DSI_LCD_ID pin is an ADC-capable pin, which can collect the voltage value of the ALIENTEK MIPI screen LCD_ID pin to distinguish different MIPI screens. Here, the R208 resistor (resistance value 10K, accuracy 1%) is used to pull up to 1.8V, in order to solve the problem that the ADC sampling value is not fixed when the development board is not connected to the MIPI screen, and the ADC sampling voltage value is fixed to 1.8V. Since the pullup resistor is added separately, it will be connected in parallel with the pullup resistor of the MIPI screen ID pin itself, so the voltage value of the MIPI screen ID pin will be updated as shown in the following table:

Table 2.1.2 ATK-DLMP257 development board MIPI screen ID pin voltage description table

DSI LCD ID pin voltage	MIPI screen model (size, resolution)
0.6V	ATK-MD0550-7201280（5.5 inch 720x1280）
0.9V	ATK-MD0550-10801920（5.5 inch 1080x1920）
1.25V	ATK-MD1001-8001280（10.1 inch 800x1280）

Note that due to the error caused by the accuracy of the resistor and the sampling accuracy of the ADC itself, it is recommended to judge the voltage range, such as 0.8V to 1.0V, rather than the specific voltage value when reading the MIPI screen ID voltage value for screen recognition.

The author can also discuss another implementation scheme here for reference. For example, for the MIPI screen with I2C communication touch function, the touch chip ID value can be read through I2C communication to indirectly detect whether the current MIPI screen is connected. When the identification is not connected to the MIPI screen, the ADC is not used for voltage acquisition, so as to avoid the problem that the ADC sampling value is not fixed and then affects the screen ID recognition. This can be achieved without the need to add a pull-up resistance to the DSI_LCD_ID pin.

2.2 MIPI screen ID identification source code implementation

The voltage values of MIPI screen ID pins of three types of ALIENTEK have been obtained in the previous subsection. Then, the voltage values of MIPI screen ID pins can be collected by using the ADC function pins in the system source code to realize the MIPI screen ID identification function. The following section describes the source code implementation of MIPI screen ID recognition function, which is also divided into two parts: uboot source code and kernel source code.

2.2.1 The uboot source code

As shown in section 1.2.1, the source code of MIPI screen ID recognition part has been listed when analyzing the RGB screen ID recognition function. Instead of repeating it here, only the main code is analyzed below.

Use vscode software to open the uboot source code, and search the macro definition "ALIENTEK_MIPI_RGB_LCD" globally in the uboot source code to view all source code implementations involving screen ID recognition functions.

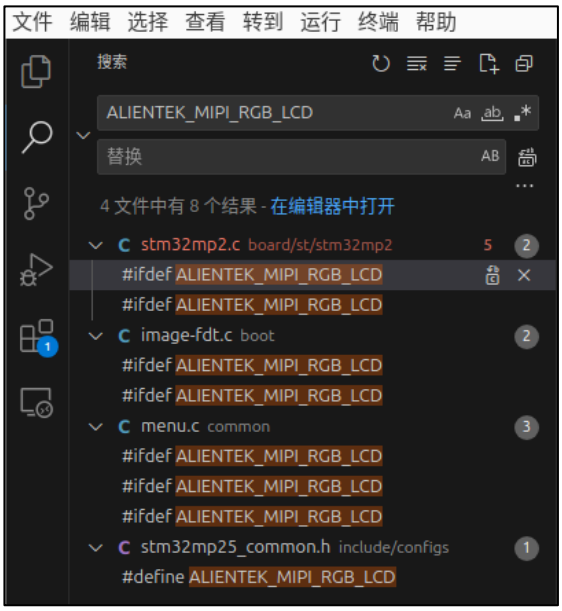


Figure 2.2.1-1 vscode searches for macro definitions

The macro definition in the uboot source files include/configs/stm32mp25_common h, can be open or shielding the macro definition, open or close the screen ID recognition.

```
#define ALIENTEK_MIPI_RGB_LCD
```

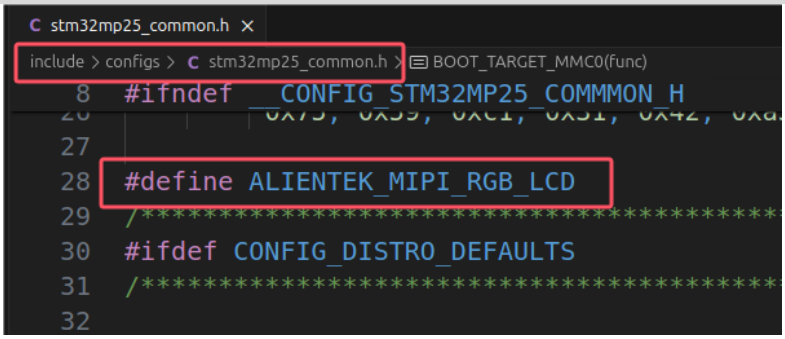


Figure 2.2.1-2 Macro definition path

MIPI screen ID function source code segment mainly concentrated in the uboot source file board/st/stm32mp2 stm32mp2. C.

Enter the following code in board_late_init:

```
#ifdef ALIENTEK_MIPI_RGB_LCD
    int ret;
    ret = alientek_set_mipi_lcd();
    if(ret < 0 || ret == 1) { //no mipi lcd, detect rgb lcd
        alientek_set_rgb_lcd();
    }
#endif
```

The code first determines whether the MIPI screen produced by ALIENTEK has been connected. If the MIPI screen has been connected, only the MIPI screen configuration is carried out. If the MIPI screen is not connected, it is determined whether the RGB screen is connected.

When the MIPI screen is accessed, the following code is executed:

```
int alientek_set_mipi_lcd(void)
{
    int ret;
    int dsi_lcd_id = 1; //atk_no_mipi
    int ts_valid = -1;
    unsigned int adc_value = 0;
    ofnode node;

    node = ofnode_path("/config_adc_mipi_dsi_lcd_id");
    if(!ofnode_valid(node)) {
        log_debug("cannot find /config_adc_mipi_dsi_lcd_id node\n");
        return -ENOENT;
    }

    ts_valid = detect_alientek_mipi_lcd_touchscreen();
    if(!ts_valid) {
        ret = adc_mipi_dsi_lcd_measurement(node, &adc_value);
        if(ret) {
            return -1;
        }
        if(adc_value > 500 && adc_value < 700) { //atk_mipi_dsi_5x5_720x1280
            dsi_lcd_id = 2;
            env_set("dsi_lcd_id", "2");
        } else if(adc_value > 800 && adc_value < 1000) { //atk_mipi_dsi_5x5_1080x1920
            dsi_lcd_id = 3;
            env_set("dsi_lcd_id", "3");
        } else if(adc_value > 1150 && adc_value < 1350) { //atk_mipi_dsi_10x1_800x1280
            dsi_lcd_id = 4;
            env_set("dsi_lcd_id", "4");
```

```
            } else { //atk_no_mipi
                dsi_lcd_id = 1;
                env_set("dsi_lcd_id", "1");
            }
            //printf("alientek dsi_lcd_id: %d\n", dsi_lcd_id);
        } else {
            dsi_lcd_id = 1;
            env_set("dsi_lcd_id", "1");
            printf("Invalid detect_alientek_mipi_lcd_touchscreen\n");
        }

        return dsi_lcd_id;
    }
```

The main function of the code is to first read the MIPI screen touch chip ID value through I2C, if the touch chip ID value is valid, it means that the current MIPI screen has been connected, and then initialize the ADC, and use the ADC to collect the DSI_LCD_ID pin voltage value, and finally determine the MIPI screen resolution according to the voltage value range. The env_set function is used to set the value of the dsi_lcd_id environment variable, which is the MIPI screen ID value that will be used later.

uboot device tree file path is arch/arm/dts/stm32mp257d-atk-ddr-1GB.dts or arch/arm/dts/stm32mp257d-atk-ddr-2GB.dts (determined by the DDR memory capacity of the core board). The following node configuration is observed in the device tree file:

```
        config_adc_mipi_dsi_lcd_id {
            st,adc_mipi_dsi_lcd_id = <&adc3 2>;
        };
```

ADC signal pin configuration, namely MIPI screen interface DSI_LCD_ID pin configuration:

```
    &adc_3 {
        pinctrl-names = "default";
        pinctrl-0 = <&adc3_in2_pins_a>;
        vdda-supply = <&vdda_1v8>;
        vref-supply = <&vddref_1v8>;
        status = "okay";

        adc3: adc@0 {
            #address-cells = <1>;
            #size-cells = <0>;
            status = "okay";

            channel@2 {
                reg = <2>;                              /* ADC3 in2 channel is used */
                st,min-sample-time-ns = <10000>;        /* 10µs sampling time */
            };
        };
```

```
};
```

MIPI screen touch interface i2c8 configuration:

```
&i2c8 {
        pinctrl-names = "default", "sleep";
        pinctrl-0 = <&i2c8_pins_a>;
        pinctrl-1 = <&i2c8_sleep_pins_a>;
        status = "okay";
        /* spare dmas for other usage */
        /delete-property/dmas;
        /delete-property/dma-names;

        goodix_ts@14 {
                compatible = "alientek,mipi-ts";
                reg = <0x14>;
                interrupt-parent = <&gpiob>;
                interrupts = <2 IRQ_TYPE_EDGE_RISING>;
                irq-gpios = <&gpiob 2 GPIO_ACTIVE_HIGH>;
                reset-gpios = <&gpiob 1 GPIO_ACTIVE_HIGH>;
                status = "okay";
        };
};
```

The following describes the source code implementation of obtaining the touch chip ID value, and analyzes the function detect_alientek_mipi_lcd_touchscreen.

```
int detect_alientek_mipi_lcd_touchscreen(void)
{
        ofnode node;
        char id[GOODIX_ID_LEN] = {0};
        int ret;

        node = ofnode_by_compatible(ofnode_null(), "alientek,mipi-ts");
        if(!ofnode_valid(node)) {
                printf("cannot find alientek,mipi-ts compatible\n");
                return -1;
        }

        ret = goodix_reset_irq_gpio(node);
        if(ret) {
                printf("error mipi lcd goodix_reset_irq_gpio\n");
                return -1;
        }

        ret = i2c_read(node, GOODIX_REG_ID, id, sizeof(id), 2);
        if((!strncmp(id, "911", sizeof(id))) || (!strncmp(id, "1151", sizeof(id)))
```

```
            || (!strncmp(id, "9271", sizeof(id))) || (!strncmp(id, "9147", sizeof(id)))
                || (!strncmp(id, "928", sizeof(id))) || (!strncmp(id, "1158", sizeof(id)))) {
                return 0;
        }

        return -1;
    }
```

In this code, the screen touch chip from Goodix brand is mainly detected, which communicates through the I2C interface, and two signal pins RESET and IRQ interrupt are required for initialization. Due to the different specifications of MIPI screens produced, different Goodix touch chip models may be used. Therefore, after obtaining the touch chip ID value, the code lists several commonly used touch chip models to improve code compatibility. If the obtained touch chip ID value is not in this range, the corresponding model needs to be added.

As mentioned earlier, once the MIPI screen ID pin voltage value is obtained, the dsi_lcd_id environment variable value is set through the env_set function. This environment variable value is the MIPI screen ID value, which will serve two important purposes: 1) The menu device tree selection menu passed to uboot is used to select the device tree to start the MIPI screen; 2) It is passed to the kernel device tree, so that the kernel MIPI screen driver can select the timing parameters of the corresponding MIPI screen to realize normal display.

To accomplish the first of these uses, modify the uboot source file common/menu.c to run the code in the menu_default_set function:

```
#ifdef ALIENTEK_MIPI_RGB_LCD
    int dsi_timings_id;
    int rgb_timings_id;

    dsi_timings_id = (int)((*(env_get("dsi_lcd_id"))) - '0');
    if (dsi_timings_id == 2 || dsi_timings_id == 3 || dsi_timings_id == 4) {
        m->default_item = menu_item_by_key(m, "3");
    } else { //no mipi lcd, detect rgb lcd
        rgb_timings_id = (int)((*(env_get("rgb_lcd_id"))) - '0');
        if (rgb_timings_id == 1 || rgb_timings_id == 2 || rgb_timings_id == 4
            || rgb_timings_id == 5) {
            m->default_item = menu_item_by_key(m, "2");
        }
    }
#endif
```

This code is used to set the startup item in the menu device tree selection menu. When the MIPI screen is detected, the third device tree startup item is selected, and when the RGB screen is detected, the second device tree startup item is selected.

To implement the second use, modify the uboot source file boot/image-fdt.c to execute the code in the boot_relocate_fdt function:

```
#ifdef ALIENTEK_MIPI_RGB_LCD
    dsi_timings_id = (int)((*(env_get("dsi_lcd_id"))) - '0');
```

```
        if (dsi_timings_id == 2 || dsi_timings_id == 3 || dsi_timings_id == 4) {
            nodeoffset_dsi = fdt_path_offset(fdt_blob, "/dsi_lcd_id");
            if (nodeoffset_dsi < 0) {
                printf("cannot find /dsi_lcd_id in linux kernel fdtfile\n");
            }
            fdt_setprop_u32(fdt_blob, nodeoffset_dsi, "dsi_select_id", dsi_timings_id);
        } …
    #endif
```

This code checks whether there is a "dsi_lcd_id" node in the loaded kernel device tree file. If there is a "dsi_lcd_id" node, it modifies its "dsi_select_id" attribute parameter value to the MIPI screen ID value obtained by uboot. At this point, the "dsi_select_id" attribute parameter value in the kernel device tree is assigned to the latest MIPI screen ID value, which in turn causes the kernel MIPI screen driver to load the corresponding screen timing.

At this point, the MIPI screen ID recognition function in the uboot source code is introduced. The following introduces the kernel source code related to screen ID recognition.

2.2.2 kernel source code

The kernel source code involves MIPI screen ID recognition, which is divided into kernel device tree and MIPI screen driver.

The kernel device tree file path of the arch/arm64 / boot/DTS/st/stm32mp257d - atk - DDR - 1 gb - mipi. DTS or arch/arm64 / boot/DTS/st/stm32mp257d - atk - DDR - 2 gb - mi pi.dts (determined by the core board DDR memory capacity). Here's part of the device tree:

```
    dsi_lcd_id {
            dsi_select_id = <2>;       //default, atk_mipi_dsi_5x5_720x1280
        };
```

In the above content, you can see that the "dsi_lcd_id" node and the attribute parameter "dsi_select_id" are set to 2, that is, the default is set to the ALIENTEK 5.5 inch MIPI screen 720x1280, which can be updated by the screen ID value obtained by uboot.

MIPI screen driver source code is located in the drivers/gpu/DRM/panel/panel - alientek - MIPI. C, performs the screen ID in function alientek_mipi_enable processing code, choose according to ID value to initialization corresponds to the screen.

```
    static int alientek_mipi_enable(struct drm_panel *panel)
    {
        ……
        np = of_find_node_by_name(NULL, "dsi_lcd_id");
        if(!np) {
            printk("no found dsi_lcd_id\n");
            return -ENODEV;
        }
        ret = of_property_read_u32(np,"dsi_select_id", &dsi_select_id);
        if(ret < 0)
            return -ENODEV;
```

```
        //printk("ATK-DEBUG:dsi_select_id = %d\n", dsi_select_id);

        if(dsi_select_id == atk_mipi_dsi_5x5_720x1280) {
            hx839x_init_sequence_720p(ctx);
        } else if(dsi_select_id == atk_mipi_dsi_5x5_1080x1920) {
            hx839x_init_sequence_1080p(ctx);
        } else if(dsi_select_id == atk_mipi_dsi_10x1_800x1280) {
            for(i = 0; i < ARRAY_SIZE(alientek_init_ili9881c); i++) {
                instr = &alientek_init_ili9881c[i];
                if(instr->op == ILI9881C_SWITCH_PAGE)
                    ret = ili9881c_switch_page(ctx, instr->arg.page);
                else if(instr->op == ILI9881C_COMMAND)
                    ret = ili9881c_send_cmd_data(ctx, instr->arg.cmd.cmd,
                                                 instr->arg.cmd.data);
                if(ret)
                    return ret;
            }
            ret = ili9881c_switch_page(ctx, 0);
            if(ret)
                return ret;
        } else {
            hx839x_init_sequence_720p(ctx);
        }
        ……
    }
```

The alientek_mipi_get_modes function selects the appropriate screen timing parameters based on their ID values.

```
    static int alientek_mipi_get_modes(struct drm_panel *panel,
                                       struct drm_connector *connector)
    {
        int ret;
        int dsi_select_id;
        struct drm_display_mode *mode;
        struct device_node *np = NULL;

        np = of_find_node_by_name(NULL, "dsi_lcd_id");
        if(!np) {
            printk("no found dsi_lcd_id\n");
            return -ENODEV;
        }
        ret = of_property_read_u32(np,"dsi_select_id", &dsi_select_id);
        if(ret < 0)
            return -ENODEV;
```

```
        if(dsi_select_id == atk_mipi_dsi_5x5_720x1280) {
            mode = drm_mode_duplicate(connector->dev, &atk_mipi_dsi_5x5_720x1280_mode);
            if(!mode) {
                printk("failed to add mode %ux%u@%u\n",
                    atk_mipi_dsi_5x5_720x1280_mode.hdisplay,
atk_mipi_dsi_5x5_720x1280_mode.vdisplay,
                    drm_mode_vrefresh(&atk_mipi_dsi_5x5_720x1280_mode));
                return -ENOMEM;
            }
        } else if(dsi_select_id == atk_mipi_dsi_5x5_1080x1920) {
            mode = drm_mode_duplicate(connector->dev, &atk_mipi_dsi_5x5_1080x1920_mode);
            if(!mode) {
                printk("failed to add mode %ux%u@%u\n",
                    atk_mipi_dsi_5x5_1080x1920_mode.hdisplay,
atk_mipi_dsi_5x5_1080x1920_mode.vdisplay,
                    drm_mode_vrefresh(&atk_mipi_dsi_5x5_1080x1920_mode));
                return -ENOMEM;
            }
        } else if(dsi_select_id == atk_mipi_dsi_10x1_800x1280) {
            mode = drm_mode_duplicate(connector->dev, &atk_mipi_dsi_10x1_800x1280_mode);
            if(!mode) {
                printk("failed to add mode %ux%u@%u\n",
                    atk_mipi_dsi_10x1_800x1280_mode.hdisplay,
atk_mipi_dsi_10x1_800x1280_mode.vdisplay,
                    drm_mode_vrefresh(&atk_mipi_dsi_10x1_800x1280_mode));
                return -ENOMEM;
            }
        } else {
            mode = drm_mode_duplicate(connector->dev, &atk_mipi_dsi_5x5_720x1280_mode);
            if(!mode) {
                printk("failed to add mode %ux%u@%u\n",
                    atk_mipi_dsi_5x5_720x1280_mode.hdisplay,
atk_mipi_dsi_5x5_720x1280_mode.vdisplay,
                    drm_mode_vrefresh(&atk_mipi_dsi_5x5_720x1280_mode));
                return -ENOMEM;
            }
        }

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

```
        return 1;
    }
```

At this point, the MIPI screen ID recognition function of the kernel source code implementation part of the introduction is completed.

Chapter 3. Summary

For the implementation of the need to be compatible with a variety of different resolution size screen, different hardware solutions will lead to different implementation methods, the implementation method provided in this paper is for reference. If the product application does not need to be compatible with a variety of specifications screen, just fixed to a specification screen, you can remove or shield the screen ID identification source code introduced in this article.