RT-Thread 的 LCD 设备驱动程序

LCD 属于 RT-Thread 的一个设备，LCD 设备具备一般 RT-Thread 设备的所有属性：init/open/close/read/write/control。其设备结构体 rt_lcd_device 由 struct rt_device 派生而来。此外，LCD 还具备特有的属性：绘图操作。要在 RT-Thread 中正常使用 LCD，需要按照 RT-Thread 的设备框架编写 LCD 设备驱动程序。

一、RT-Thread 的 LCD 设备框架

RT-Thread 的 LCD 设备驱动由以下 4 个内容组成。

- 1 个结构体
  rt_lcd_device

- 2 个硬件接口函数
  lcd_init();
  lcd_control();

- 5 个基础绘图函数,并用它们初始化 rt_device_graphic_ops
  void rt_hw_lcd_set_pixel(const char *pixel, int x, int y);
  void rt_hw_lcd_get_pixel(char *pixel, int x, int y);
  void rt_hw_lcd_draw_hline(const char *pixel, int x1, int x2, int y);
  void rt_hw_lcd_draw_vline(const char *pixel, int x, int y1, int y2);
  void rt_hw_lcd_draw_raw_hline(const char *pixel, int x, int y, rt_size_t size);

- 向 RTT 注册 LCD 设备，初始化并打开 LCD 设备
  调用函数 int rt_hw_lcd_init(void)，向 RTT 注册、初始化并打开 LCD 设备。

1.1 结构体 rt_lcd_device

rt_lcd_device 结构体原型如下：

```
struct rt_lcd_device
{
    struct rt_device    parent;

    rt_uint8_t    pixel_format;
    rt_uint8_t    bits_per_pixel;
    rt_uint16_t   width;
    rt_uint16_t   height;
    rt_uint8_t    direction;
    rt_uint16_t   id;

    void *framebuffer;

};
typedef struct rt_lcd_device rt_lcd_t;
```

编写 LCD 驱动的需要填充此结构体。

1.2 2 个硬件接口函数

这两个接口函数是所有 RTT 管理的设备都具备的公共函数，
RTT 的设备还具备其它的公共函数，但对于 LCD 设备，我们
只需要填充这两个函数

（1）static rt_err_t lcd_init(rt_device_t dev)

这个是 lcd 硬件设备的初始化函数，需要做的工作有：

1.初始化和 lcd 连接的处理器的接口；

2.初始化液晶驱动芯片。

（2）static rt_err_t lcd_control(rt_device_t dev, rt_uint8_t cmd, void *args)

这个是 lcd 设备的控制操作函数，通过它设置一些 lcd 的参数。

1.3 5 个基本的绘图函数

5 个基本函数分别是：

1.画点

2.获取点颜色

3.画水平线

4.画垂直线

5.画起始水平线

UIENGINE 的所有绘制操作都是以这 5 个绘图函数为基础的，
是使用 LCD 必须提供的函数。

函数定义：void rt_hw_lcd_set_pixel(const char *pixel, int x, int y);

参数说明：pixel 是颜色信息，x，液晶的 x 坐标；y，液晶的 y 坐标。

函数定义：void rt_hw_lcd_get_pixel(char *pixel, int x, int y);

参数说明：pixel 是颜色信息，x，液晶的 x 坐标；y，液晶的 y 坐标。

函数定义：void rt_hw_lcd_draw_hline(const char *pixel, int x1, int x2, int y);

参数说明：pixel 是颜色信息，x1，液晶的 x 轴起始坐标；x2，液晶的 x 轴结束坐标，y，液晶的 y 轴坐标。

函数定义：void rt_hw_lcd_draw_vline(const char *pixel, int x, int y1, int y2);

参数说明：pixel 是颜色信息，x，液晶的 x 轴坐标；y1，液晶的 y 轴起始坐标，y2，液晶的 y 轴结束坐标。

函数定义：void rt_hw_lcd_draw_raw_hline(const char *pixel, int x, int y, rt_size_t size);

参数说明：pixel 是颜色信息，x，液晶的 x 轴坐标；y，液晶的 y 轴坐标，size，液晶水平轴上的像素个数。

1.4　向 RTT 注册 LCD 设备，初始化并打开 LCD 设备

调用 rt_hw_lcd_init()函数，向 RTT 注册 LCD 设备，初始化并打开 LCD 设备。其函数原型如下：

```
void rt_hw_lcd_init(void)
{
    rt_device_t lcd;

    /*初始化 LCD 设备结构体*/
    lcd_device.parent.type          = RT_Device_Class_Graphic;   /*设备类型*/
    lcd_device.parent.init          = lcd_init;
    lcd_device.parent.open          = lcd_open;
    lcd_device.parent.close         = lcd_close;
    lcd_device.parent.read          = RT_NULL;
    lcd_device.parent.write         = RT_NULL;
    lcd_device.parent.control       = lcd_control;
    lcd_device.parent.user_data = &lcd_ili_ops; /*设置用户私有数据*/

    lcd_device.pixel_format     = RTGRAPHIC_PIXEL_FORMAT_RGB565P;
    lcd_device.bits_per_pixel = LCD_BITS_PER_PIXEL;

    /*注册图形设备驱动*/
    rt_device_register(&(lcd_device.parent),
                       "lcd",
                        RT_DEVICE_FLAG_RDWR|
T_DEVICE_FLAG_STANDALONE);
    lcd = rt_device_find("lcd");   /*查找 LCD 设备                    */

    /*初始化*/
    rt_device_init(lcd);
    rt_device_open(lcd,RT_DEVICE_OFLAG_RDWR);
}
```

二、以 **STM32** 为例，编写一个 **LCD** 驱动程序

下面利用上述操作步骤编写一个 RTT 的 LCD 驱动程序：
单片机：STM32F407IGT6
LCD 模块：原子哥的 2.4/2.8 寸液晶，液晶驱动芯片 ILI9341

单片机接口用了 FSMC 端口
端口接线：

液晶接口		单片机接口	作用
CS	----	FSMC_A25	片选
RS	----	FSMC_A15	数据/命令
WR	----	FSMC_NWE	写使能
RD	----	FSMC_NOE	读使能
RST	----	FSMC_A19	复位
D0-D15	----	FSMC_D0-FSMC_D15	数据线
BL	----	PE2	背光

2.1 编写硬件初始化函数

2.1.1 配置单片机引脚

```
RCC_Configuration();
GPIO_Configuration();
FSMC_Configuration();
```

2.1.2 设置 FSMC 驱动液晶的显存地址和寄存器地址

```
#define LCD_REG (*(volatile rt_uint16_t*)(rt_uint32_t)0x60000000)     //显存地址，用于写显示数据
#define LCD_RAM (*(volatile rt_uint16_t*)(rt_uint32_t)0x60010000)     //寄存器地址，用于写显示器的配置寄存器
```

2.1.3 液晶的寄存器操作函数

```
/*写显示寄存器，用于显示数据*/
 void LCD_WR_RAM(rt_uint16_t Val)
 {
       LCD_RAM = Val;
 }
 /*写显示寄存器，用于配置显示器*/
 void LCD_WR_REG(rt_uint16_t index)
{
     LCD_REG = index;
}
 /*读显示寄存器，读取当前显示的内容*/
rt_uint16_t LCD_RD_DATA(void)
{
     return LCD_RAM;
}
```

2.2 初始化 lcd 设备结构体

```
struct rt_lcd_device
{
	struct rt_device	parent;

	rt_uint8_t	pixel_format;
	rt_uint8_t	bits_per_pixel;
	rt_uint16_t width;
	rt_uint16_t height;
	rt_uint8_t	direction;
	rt_uint16_t id;

	void *framebuffer;
};

struct rt_lcd_device	lcd_device;
```

2.3 编写 2 个硬件接口函数

```
/*	LCD 设备初始化	*/
static rt_err_t lcd_init(rt_device_t dev)
{
	RCC_Configuration();
	GPIO_Configuration();
	FSMC_Configuration();

	delay_ms(50); // delay 50 ms
	LCD_WriteReg(0x0000,0x0001);
	delay_ms(50); // delay 50 ms
	lcd_device.id = LCD_ReadReg(0x0000);

	//尝试 9341 ID 的读取
	LCD_WR_REG(0XD3);
	LCD_RD_DATA();				//dummy read
	LCD_RD_DATA();			//读到 0X00
	lcd_device.id=LCD_RD_DATA();		//读取 93

	lcd_device.id<<=8;
	lcd_device.id|=LCD_RD_DATA();		//读取 41
	if(lcd_device.id != 0X9341)		//非 9341
	{
```

```
        rt_kprintf("LCD ID is not 0x9341.");
        return RT_ERROR;
 }

rt_kprintf("LCD ID: %x\r\n",lcd_device.id); //打印 LCD ID

if(lcd_device.id==0X9341)//9341 初始化
{
      LCD_WR_REG(0xCF);
      LCD_WR_DATA(0x00);
      LCD_WR_DATA(0xC1);
      LCD_WR_DATA(0X30);
      LCD_WR_REG(0xED);
      LCD_WR_DATA(0x64);
      LCD_WR_DATA(0x03);
      LCD_WR_DATA(0X12);
      LCD_WR_DATA(0X81);
      LCD_WR_REG(0xE8);
      LCD_WR_DATA(0x85);
      LCD_WR_DATA(0x10);
      LCD_WR_DATA(0x7A);
      LCD_WR_REG(0xCB);
      LCD_WR_DATA(0x39);
      LCD_WR_DATA(0x2C);
      LCD_WR_DATA(0x00);
      LCD_WR_DATA(0x34);
      LCD_WR_DATA(0x02);
      LCD_WR_REG(0xF7);
      LCD_WR_DATA(0x20);
      LCD_WR_REG(0xEA);
      LCD_WR_DATA(0x00);
      LCD_WR_DATA(0x00);
      LCD_WR_REG(0xC0);       //Power control
      LCD_WR_DATA(0x1B);      //VRH[5:0]
      LCD_WR_REG(0xC1);       //Power control
      LCD_WR_DATA(0x01);      //SAP[2:0];BT[3:0]
      LCD_WR_REG(0xC5);       //VCM control
      LCD_WR_DATA(0x30);        //3F
      LCD_WR_DATA(0x30);        //3C
      LCD_WR_REG(0xC7);       //VCM control2
      LCD_WR_DATA(0XB7);
      LCD_WR_REG(0x36);       // Memory Access Control
      LCD_WR_DATA(0x48);
      LCD_WR_REG(0x3A);
```

```
LCD_WR_DATA(0x55);
LCD_WR_REG(0xB1);
LCD_WR_DATA(0x00);
LCD_WR_DATA(0x1A);
LCD_WR_REG(0xB6);       // Display Function Control
LCD_WR_DATA(0x0A);
LCD_WR_DATA(0xA2);
LCD_WR_REG(0xF2);       // 3Gamma Function Disable
LCD_WR_DATA(0x00);
LCD_WR_REG(0x26);       //Gamma curve selected
LCD_WR_DATA(0x01);
LCD_WR_REG(0xE0);       //Set Gamma
LCD_WR_DATA(0x0F);
LCD_WR_DATA(0x2A);
LCD_WR_DATA(0x28);
LCD_WR_DATA(0x08);
LCD_WR_DATA(0x0E);
LCD_WR_DATA(0x08);
LCD_WR_DATA(0x54);
LCD_WR_DATA(0XA9);
LCD_WR_DATA(0x43);
LCD_WR_DATA(0x0A);
LCD_WR_DATA(0x0F);
LCD_WR_DATA(0x00);
LCD_WR_DATA(0x00);
LCD_WR_DATA(0x00);
LCD_WR_DATA(0x00);
LCD_WR_REG(0XE1);       //Set Gamma
LCD_WR_DATA(0x00);
LCD_WR_DATA(0x15);
LCD_WR_DATA(0x17);
LCD_WR_DATA(0x07);
LCD_WR_DATA(0x11);
LCD_WR_DATA(0x06);
LCD_WR_DATA(0x2B);
LCD_WR_DATA(0x56);
LCD_WR_DATA(0x3C);
LCD_WR_DATA(0x05);
LCD_WR_DATA(0x10);
LCD_WR_DATA(0x0F);
LCD_WR_DATA(0x3F);
LCD_WR_DATA(0x3F);
LCD_WR_DATA(0x0F);
LCD_WR_REG(0x2B);
```

```
            LCD_WR_DATA(0x00);
            LCD_WR_DATA(0x00);
            LCD_WR_DATA(0x01);
            LCD_WR_DATA(0x3f);
            LCD_WR_REG(0x2A);
            LCD_WR_DATA(0x00);
            LCD_WR_DATA(0x00);
            LCD_WR_DATA(0x00);
            LCD_WR_DATA(0xef);
            LCD_WR_REG(0x11); //Exit Sleep
            delay_ms(120); // delay 120 ms
            LCD_WR_REG(0x29); //display on
        }

        if(LCD_DIRNORMAL == LCD_DIRV)//竖屏
        {
            lcd_device.direction = LCD_DIRNORMAL;   //竖屏
            lcd_device.width=LCD_WIDTH;
            lcd_device.height=LCD_HEIGHT;
        }
        else//横屏
        {
            lcd_device.direction = LCD_DIRNORMAL;   //横屏
            lcd_device.width=LCD_HEIGHT;
            lcd_device.height=LCD_WIDTH;
        }

        LCD_Scan_Dir(DFT_SCAN_DIR);     //默认扫描方向

        GPIO_SetBits(GPIOE, GPIO_Pin_2);                    //点亮背光

        return (RT_EOK);
}

/*    LCD 设备控制    */
static rt_err_t lcd_control(rt_device_t dev, rt_uint8_t cmd, void *args)
{
        switch(cmd)
        {
            case RTGRAPHIC_CTRL_GET_INFO:    /*  返回 LCD 屏幕信息  */
            {
                struct rt_device_graphic_info *info;

                info = (struct rt_device_graphic_info*)args;
```

```
            RT_ASSERT(info != RT_NULL);

            info->bits_per_pixel = lcd_device.bits_per_pixel;
            info->pixel_format   = lcd_device.pixel_format;
            info->framebuffer    = RT_NULL;
            info->width          = lcd_device.width;
            info->height         = lcd_device.height;
        }
        break;

        default:
            break;
    }

    return (RT_EOK);
}
```

2.4 编写 5 个基本的 LCD 操作函数，并用它们初始化 rt_device_graphic_ops

```
/*画点*/
void rt_hw_lcd_set_pixel(const char * pixel, int x, int y)
{
    LCD_SetCursor(x,y);

    LCD_WriteRAM_Prepare();
    LCD_WriteRAM(*(rt_uint16_t*)pixel);
}


/*获取点颜色*/
void rt_hw_lcd_get_pixel(char * pixel, int x, int y)
{
    *(rt_uint16_t*)pixel = LCD_ReadPoint(x,y);
}

/*画水平线*/
void rt_hw_lcd_draw_hline(const char * pixel, int x1, int x2, int y)
{
    LCD_SetCursor(x1,y);
    LCD_WriteRAM_Prepare();
```

```
    while(x1<x2)
    {
        LCD_WriteRAM(*(rt_uint16_t*)pixel);
        x1 ++;
    }
}

/*画垂直线*/
void rt_hw_lcd_draw_vline(const char * pixel, int x, int y1, int y2)
{
    LCD_SetCursor(x,y1);
    LCD_WriteRAM_Prepare();

    while(y1<y2)
    {
        LCD_WriteRAM(*(rt_uint16_t*)pixel);
        y1 ++;
    }
}

/*画起始水平线*/
void rt_hw_lcd_draw_raw_hline(const char * pixels, int x, int y, rt_size_t size)
{
    rt_uint16_t * ptr;
    ptr = (rt_uint16_t*)pixels;

    LCD_SetCursor(x,y);
    LCD_WriteRAM_Prepare();

    while(size)
    {
        LCD_WriteRAM(*ptr++);
        size --;
    }
}

/*实现图形设备操作结构体*/
struct rt_device_graphic_ops    lcd_ili_ops =
{
    rt_hw_lcd_set_pixel,                    /*画点*/
    rt_hw_lcd_get_pixel,                    /*获取点颜色*/
    rt_hw_lcd_draw_hline,                    /*画水平线*/
    rt_hw_lcd_draw_vline,                    /*画垂直线*/
    rt_hw_lcd_draw_raw_hline                 /*画起始水平线（填充一行数据）*/
```

```
};
```

2.5 调用 rt_hw_lcd_init()向 RTT 注册 LCD 设备，初始化并打开 LCD 设备

```
Void    rt_hw_lcd_init(void)
{
      rt_device_t lcd;

      /*初始化 LCD 设备结构体*/
      lcd_device.parent.type          = RT_Device_Class_Graphic;    /*设备类型*/
      lcd_device.parent.init          = lcd_init;
      lcd_device.parent.open          = lcd_open;
      lcd_device.parent.close         = lcd_close;
      lcd_device.parent.read          = RT_NULL;
      lcd_device.parent.write         = RT_NULL;
      lcd_device.parent.control       = lcd_control;
      lcd_device.parent.user_data = &lcd_ili_ops; /*设置用户私有数据*/

      lcd_device.pixel_format      = RTGRAPHIC_PIXEL_FORMAT_RGB565P;
     lcd_device.bits_per_pixel = LCD_BITS_PER_PIXEL;

      /*注册图形设备驱动*/
      rt_device_register(&(lcd_device.parent),
                                    "lcd",
                                    RT_DEVICE_FLAG_RDWR                          |
RT_DEVICE_FLAG_STANDALONE);
      lcd = rt_device_find("lcd");    /*查找 LCD 设备                    */

      /*初始化*/
      rt_device_init(lcd);
      rt_device_open(lcd,RT_DEVICE_OFLAG_RDWR);
}
```

2.6 验证 LCD 设备驱动程序

到此为止，已经成功将 LCD 液晶添加到 RTT 的设备管理器中，可以在 UIENGINE 中对 LCD 进行操作。
下面通过在液晶上绘制一个窗口对驱动进行验证：
1、在 void rt_init_thread_entry(void* parameter)函数最后添加以下代码，初始化 LCD。

```
        rt_device_t lcd;

        rt_hw_lcd_init();                   /*初始化并注册 LCD 设备          */

        lcd = rt_device_find("lcd");    /*查找 LCD 设备                      */
        if(lcd == RT_NULL)
        {
            rt_kprintf("no graphic device in the system.\n");
        }
        rtgui_graphic_set_device(lcd); /*设置 LCD 设备为 GUI 驱动        */

        rtgui_system_server_init();
```

2、在 application.c 中新建一个 LCD 测试线程入口函数

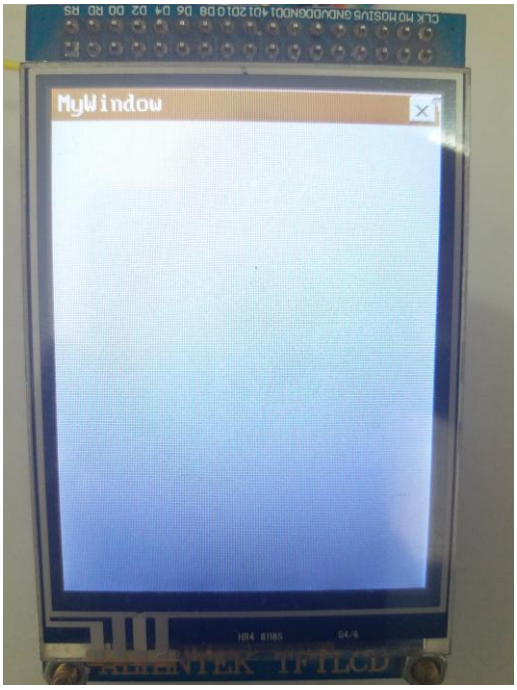
```
void    LCD_demo_entry(void* parameter)
{

    struct rtgui_app *app;
    struct rtgui_win *win;
    struct rtgui_box *box;

    struct rtgui_rect rect={0,20,240,320};

    app = rtgui_app_create("MyApp");
    RT_ASSERT(app != RT_NULL);

    win = rtgui_win_create(RT_NULL, "MyWindow", &rect,
RTGUI_WIN_STYLE_DEFAULT);

    box = rtgui_box_create(RTGUI_HORIZONTAL, 20);
    rtgui_container_set_box(RTGUI_CONTAINER(win), box);

    rtgui_container_layout(RTGUI_CONTAINER(win));

    rtgui_win_show(win, RT_FALSE);

    rtgui_app_run(app);

    rtgui_win_destroy(win);
    rtgui_app_destroy(app);

    rt_kprintf("MyApp Quit.\n");
}
```

3、将下面代码添加到 rt_application_init()函数最后。

```
#ifdef RT_USING_GUIENGINE
```

```
        tid = rt_thread_create("gui_demo",
                               gui_application_entry,
                               RT_NULL,
                               2048*8,
                               25,
                               10);
    if(tid != RT_NULL)
        rt_thread_startup(tid);
#endif
```

将程序下载到测试板上，在 LCD 中看到以下画面：



三、其它不带 frambuffer 的 LCD 设备驱动

四、带 frambuffer 的 LCD 设备驱动程序