#if 1

/\*\*

 \* @file lcd\_rtthread.c

 \* @brief ST7789 LCD驱动实现 - RT-Thread适配版

 \*

 \* 主要功能：

 \* - LCD初始化配置

 \* - 基本绘图功能实现

 \* - 文本显示功能

 \* - 底层硬件接口封装

 \*

 \* 移植说明：

 \* 1. 基于原厂lcd.c驱动移植

 \* 2. 适配RT-Thread SPI设备框架

 \* 3. 使用RT-Thread PIN框架控制GPIO

 \*/

#include "lcd\_rtthread.h"

#include <rtthread.h>

#include <rtdevice.h>

#include <drv\_common.h>

#include <drv\_spi.h>   /\* 添加SPI驱动头文件 \*/

#include "font.h"      /\* 包含字体数据头文件 \*/

#include "gt911.h"     /\* 包含GT911触摸驱动头文件 \*/

#include "cyhal\_dma.h" /\* 添加DMA驱动头文件 \*/

#include "cyhal\_syspm.h"

#include "cy\_pdl.h"

#include "cy\_dma.h"

///\* 函数前向声明 \*/

// static u32 LCD\_Pow(u8 m, u8 n);

#define DBG\_TAG "lcd.rt"

#define DBG\_LVL DBG\_INFO

#include <rtdbg.h>

/\* DMA相关定义 \*/

#define LCD\_DMA\_ENABLE 1         /\* 1:启用DMA, 0:禁用DMA \*/

#define LCD\_DMA\_PRIORITY 3       /\* DMA优先级(0-3), 3最高 \*/

#define LCD\_DMA\_BUFFER\_SIZE 1024 /\* DMA缓冲区大小 \*/

u8 DFT\_SCAN\_DIR;

\_lcd\_dev lcddev;

static struct rt\_spi\_device \*lcd\_dev;

#if LCD\_DMA\_ENABLE

/\* DMA相关变量 \*/

static cyhal\_dma\_t lcd\_dma;

static bool dma\_initialized = false;

static uint16\_t dma\_buffer[LCD\_DMA\_BUFFER\_SIZE];

static volatile bool dma\_transfer\_complete = true;

/\* DMA传输完成回调函数 \*/

static void lcd\_dma\_callback(void \*callback\_arg, cyhal\_dma\_event\_t event)

{

    RT\_UNUSED(callback\_arg);

    if (event & CYHAL\_DMA\_TRANSFER\_COMPLETE)

    {

        dma\_transfer\_complete = true;

    }

}

/\*\*

 \* @brief 初始化LCD使用的DMA

 \*

 \* @return rt\_err\_t 初始化结果

 \*/

static rt\_err\_t lcd\_dma\_init(void)

{

    cy\_rslt\_t result;

    if (dma\_initialized)

        return RT\_EOK;

    // 初始化DMA通道

    result = cyhal\_dma\_init(&lcd\_dma, CYHAL\_DMA\_PRIORITY\_DEFAULT, CYHAL\_DMA\_DIRECTION\_MEM2PERIPH);

    if (result != CY\_RSLT\_SUCCESS)

    {

        LOG\_E("Failed to initialize DMA, error code: %d", result);

        return -RT\_ERROR;

    }

    // 注册DMA传输完成回调函数

    cyhal\_dma\_register\_callback(&lcd\_dma, lcd\_dma\_callback, NULL);

    cyhal\_dma\_enable\_event(&lcd\_dma, CYHAL\_DMA\_TRANSFER\_COMPLETE, LCD\_DMA\_PRIORITY, true);

    dma\_initialized = true;

    LOG\_I("LCD DMA initialized successfully");

    return RT\_EOK;

}

/\*\*

 \* @brief 释放LCD使用的DMA资源

 \*/

static void lcd\_dma\_free(void)

{

    if (dma\_initialized)

    {

        cyhal\_dma\_free(&lcd\_dma);

        dma\_initialized = false;

        LOG\_I("LCD DMA resources freed");

    }

}

/\*\*

 \* @brief 使用DMA发送数据到LCD

 \*

 \* @param data 数据缓冲区

 \* @param size 数据大小(16位单位)

 \* @return rt\_err\_t 发送结果

 \*/

static rt\_err\_t lcd\_dma\_transmit(const uint16\_t \*data, uint32\_t size)

{

    cy\_rslt\_t result;

    uint32\_t chunk\_size;

    uint32\_t sent = 0;

    if (!dma\_initialized)

    {

        result = lcd\_dma\_init();

        if (result != RT\_EOK)

            return result;

    }

    LCD\_DC\_SET; // 设置为数据模式

    while (sent < size)

    {

        // 计算当前需要发送的数据块大小(不超过缓冲区大小)

        chunk\_size = (size - sent) > LCD\_DMA\_BUFFER\_SIZE ? LCD\_DMA\_BUFFER\_SIZE : (size - sent);

        // 将数据复制到DMA缓冲区(可能需要做字节序转换)

        for (uint32\_t i = 0; i < chunk\_size; i++)

        {

            dma\_buffer[i] = data[sent + i];

        }

        // 等待之前的DMA传输完成

        while (!dma\_transfer\_complete)

        {

            rt\_thread\_mdelay(1);

        }

        dma\_transfer\_complete = false;

        // 修正：确保使用正确的SPI传输地址

        struct rt\_spi\_message message;

        message.send\_buf = dma\_buffer;

        message.recv\_buf = RT\_NULL;

        message.length = chunk\_size \* 2; // 每个16位数据占2字节

        message.cs\_take = 1;

        message.cs\_release = 1;

        message.next = RT\_NULL;

        // 配置DMA传输

        cyhal\_dma\_cfg\_t dma\_cfg = {

            .src\_addr = (uint32\_t)dma\_buffer,

            .src\_increment = 1,

            .dst\_addr = (uint32\_t)&message, // 修正目标地址

            .dst\_increment = 0,

            .transfer\_width = 16, // 16位传输

            .length = chunk\_size,

            .burst\_size = 0,

            .action = CYHAL\_DMA\_TRANSFER\_FULL

        };

        result = cyhal\_dma\_configure(&lcd\_dma, &dma\_cfg);

        if (result != CY\_RSLT\_SUCCESS)

        {

            LOG\_E("DMA configure failed: %d", result);

            // 回退到普通SPI传输

            rt\_spi\_transfer(lcd\_dev, dma\_buffer, RT\_NULL, chunk\_size \* 2);

            sent += chunk\_size;

            continue;

        }

        result = cyhal\_dma\_enable(&lcd\_dma);

        if (result != CY\_RSLT\_SUCCESS)

        {

            LOG\_E("DMA enable failed: %d", result);

            // 回退到普通SPI传输

            rt\_spi\_transfer(lcd\_dev, dma\_buffer, RT\_NULL, chunk\_size \* 2);

            sent += chunk\_size;

            continue;

        }

        result = cyhal\_dma\_start\_transfer(&lcd\_dma);

        if (result != CY\_RSLT\_SUCCESS)

        {

            LOG\_E("DMA start transfer failed: %d", result);

            // 回退到普通SPI传输

            rt\_spi\_transfer(lcd\_dev, dma\_buffer, RT\_NULL, chunk\_size \* 2);

            sent += chunk\_size;

            continue;

        }

        sent += chunk\_size;

    }

    // 等待最后一次传输完成

    while (!dma\_transfer\_complete)

    {

        rt\_thread\_mdelay(1);

    }

    return RT\_EOK;

}

#endif

/\* 使用lcd\_rtthread.h中定义的引脚操作宏 \*/

/\* 移植后的SPI接口函数 \*/

/\*\*

 \* @brief 写LCD寄存器命令

 \* @param reg 寄存器地址/命令

 \*

 \* 功能说明：

 \* 1. 拉低DC线表示写命令

 \* 2. 通过SPI发送命令字节

 \* 3. 拉高DC线恢复数据模式

 \*/

static void LCD\_WR\_REG(uint8\_t reg)

{

    LCD\_DC\_CLR;

#if LCD\_DMA\_ENABLE

    if (dma\_initialized || lcd\_dma\_init() == RT\_EOK)

    {

        // 使用DMA发送命令

        uint8\_t cmd = reg;

        dma\_buffer[0] = reg;

        dma\_transfer\_complete = false;

        // 配置DMA传输

        cyhal\_dma\_cfg\_t dma\_cfg = {

            .src\_addr = (uint32\_t)&dma\_buffer[0],

            .src\_increment = 0,

            .dst\_addr = (uint32\_t)&lcd\_dev->bus->ops->xfer,

            .dst\_increment = 0,

            .transfer\_width = 8,  // 8位传输

            .length = 1,

            .burst\_size = 0,

            .action = CYHAL\_DMA\_TRANSFER\_FULL

        };

        cy\_rslt\_t result = cyhal\_dma\_configure(&lcd\_dma, &dma\_cfg);

        if (result == CY\_RSLT\_SUCCESS)

        {

            cyhal\_dma\_enable(&lcd\_dma);

            cyhal\_dma\_start\_transfer(&lcd\_dma);

            // 等待传输完成

            while (!dma\_transfer\_complete)

            {

                // 短暂等待

            }

        }

        else

        {

            // DMA配置失败，回退到普通传输

            rt\_spi\_send(lcd\_dev, &reg, 1);

        }

    }

    else

    {

        // DMA未初始化，使用普通传输

        rt\_spi\_send(lcd\_dev, &reg, 1);

    }

#else

    rt\_spi\_send(lcd\_dev, &reg, 1);

#endif

    LCD\_DC\_SET;

}

/\*\*

 \* @brief 写LCD数据(8位)

 \* @param data 要写入的数据

 \*

 \* 功能说明：

 \* 1. 保持DC线为高表示写数据

 \* 2. 通过SPI发送单字节数据

 \*/

static void LCD\_WR\_DATA(uint8\_t data)

{

    LCD\_DC\_SET;

#if LCD\_DMA\_ENABLE

    if (dma\_initialized || lcd\_dma\_init() == RT\_EOK)

    {

        // 使用DMA发送数据

        uint8\_t\* data\_ptr = (uint8\_t\*)dma\_buffer;

        data\_ptr[0] = data;

        dma\_transfer\_complete = false;

        // 配置DMA传输

        cyhal\_dma\_cfg\_t dma\_cfg = {

            .src\_addr = (uint32\_t)data\_ptr,

            .src\_increment = 0,

            .dst\_addr = (uint32\_t)&lcd\_dev->bus->ops->xfer,

            .dst\_increment = 0,

            .transfer\_width = 8,  // 8位传输

            .length = 1,

            .burst\_size = 0,

            .action = CYHAL\_DMA\_TRANSFER\_FULL

        };

        cy\_rslt\_t result = cyhal\_dma\_configure(&lcd\_dma, &dma\_cfg);

        if (result == CY\_RSLT\_SUCCESS)

        {

            cyhal\_dma\_enable(&lcd\_dma);

            cyhal\_dma\_start\_transfer(&lcd\_dma);

            // 等待传输完成

            while (!dma\_transfer\_complete)

            {

                // 短暂等待

            }

            return;

        }

    }

#endif

    // DMA未启用或初始化失败，使用普通传输

    rt\_spi\_send(lcd\_dev, &data, 1);

}

/\*\*

 \* @brief 写LCD数据(16位)

 \* @param data 要写入的16位数据

 \*

 \* 功能说明：

 \* 1. 保持DC线为高表示写数据

 \* 2. 将16位数据拆分为高低字节

 \* 3. 通过SPI发送两个字节

 \*/

static void LCD\_WR\_DATA\_16BIT(uint16\_t data)

{

    LCD\_DC\_SET;

#if LCD\_DMA\_ENABLE

    if (dma\_initialized || lcd\_dma\_init() == RT\_EOK)

    {

        // 使用DMA发送数据

        uint8\_t\* data\_ptr = (uint8\_t\*)dma\_buffer;

        data\_ptr[0] = data >> 8;    // 高字节

        data\_ptr[1] = data & 0xFF;  // 低字节

        dma\_transfer\_complete = false;

        // 配置DMA传输

        cyhal\_dma\_cfg\_t dma\_cfg = {

            .src\_addr = (uint32\_t)data\_ptr,

            .src\_increment = 1,

            .dst\_addr = (uint32\_t)&lcd\_dev->bus->ops->xfer,

            .dst\_increment = 0,

            .transfer\_width = 8,  // 8位传输

            .length = 2,  // 两个字节(16位)

            .burst\_size = 0,

            .action = CYHAL\_DMA\_TRANSFER\_FULL

        };

        cy\_rslt\_t result = cyhal\_dma\_configure(&lcd\_dma, &dma\_cfg);

        if (result == CY\_RSLT\_SUCCESS)

        {

            cyhal\_dma\_enable(&lcd\_dma);

            cyhal\_dma\_start\_transfer(&lcd\_dma);

            // 等待传输完成

            while (!dma\_transfer\_complete)

            {

                // 短暂等待

            }

            return;

        }

    }

#endif

    // DMA未启用或初始化失败，使用普通传输

    uint8\_t buf[2] = {data >> 8, data & 0xFF};

    rt\_spi\_send(lcd\_dev, buf, 2);

}

/\* 新增GRAM操作函数 \*/

/\*\*

 \* @brief 写GRAM数据

 \* @param RGB\_Code RGB565格式颜色值

 \*

 \* 功能说明：

 \* 1. 调用16位数据写入函数

 \* 2. 用于连续写入GRAM数据

 \*/

static void LCD\_WriteRAM(uint16\_t RGB\_Code)

{

    LCD\_WR\_DATA\_16BIT(RGB\_Code);

}

/\*\*

 \* @brief 准备GRAM写入

 \*

 \* 功能说明：

 \* 1. 发送GRAM写入命令

 \* 2. 必须在连续写入GRAM数据前调用

 \*/

static void LCD\_WriteRAM\_Prepare(void)

{

    LCD\_WR\_REG(lcddev.wramcmd);

}

/\*\*

 \* @brief 写LCD寄存器值

 \* @param LCD\_Reg 寄存器地址

 \* @param LCD\_RegValue 要写入的寄存器值

 \*

 \* 功能说明：

 \* 1. 先写寄存器地址

 \* 2. 再写寄存器值(16位)

 \*/

static void LCD\_WriteReg(uint8\_t LCD\_Reg, uint16\_t LCD\_RegValue)

{

    LCD\_WR\_REG(LCD\_Reg);

    LCD\_WR\_DATA(LCD\_RegValue);

}

/\* 保留lcd.c中的初始化命令序列 \*/

static void LCD\_INIT\_CODE(u8 dir)

{

    // LCD\_BLK\_CLR;

    LCD\_WR\_REG(0x11);

    rt\_thread\_mdelay(120); // Delay 120ms

    //------------------------------display and color format setting--------------------------------//

    LCD\_WR\_REG(0x36);

    LCD\_WR\_DATA(0x00);

    LCD\_WR\_REG(0x3a);

    LCD\_WR\_DATA(0x05);

    //--------------------------------ST7789V Frame rate setting----------------------------------//

    LCD\_WR\_REG(0xb2);

    LCD\_WR\_DATA(0x0c);

    LCD\_WR\_DATA(0x0c);

    LCD\_WR\_DATA(0x00);

    LCD\_WR\_DATA(0x33);

    LCD\_WR\_DATA(0x33);

    LCD\_Display\_Dir(dir);

    LCD\_WR\_REG(0xb7);

    LCD\_WR\_DATA(0x35);

    //---------------------------------ST7789V Power setting--------------------------------------//

    LCD\_WR\_REG(0xbb);

    LCD\_WR\_DATA(0x28);

    LCD\_WR\_REG(0xc0);

    LCD\_WR\_DATA(0x2c);

    LCD\_WR\_REG(0xc2);

    LCD\_WR\_DATA(0x01);

    LCD\_WR\_REG(0xc3);

    LCD\_WR\_DATA(0x10);

    LCD\_WR\_REG(0xc4);

    LCD\_WR\_DATA(0x20);

    LCD\_WR\_REG(0xc6);

    LCD\_WR\_DATA(0x0f);

    LCD\_WR\_REG(0xd0);

    LCD\_WR\_DATA(0xa4);

    LCD\_WR\_DATA(0xa1);

    //--------------------------------ST7789V gamma setting---------------------------------------//

    // 0xD0, 0x04, 0x0D, 0x11, 0x13, 0x2B, 0x3F, 0x54, 0x4C, 0x18, 0x0D, 0x0B, 0x1F, 0x23

    LCD\_WR\_REG(0xe0);

    LCD\_WR\_DATA(0xd0);

    LCD\_WR\_DATA(0x00);

    LCD\_WR\_DATA(0x02);

    LCD\_WR\_DATA(0x07);

    LCD\_WR\_DATA(0x0a);

    LCD\_WR\_DATA(0x28);

    LCD\_WR\_DATA(0x32);

    LCD\_WR\_DATA(0x44);

    LCD\_WR\_DATA(0x42);

    LCD\_WR\_DATA(0x06);

    LCD\_WR\_DATA(0x0e);

    LCD\_WR\_DATA(0x12);

    LCD\_WR\_DATA(0x14);

    LCD\_WR\_DATA(0x17);

    // 0xD0, 0x04, 0x0C, 0x11, 0x13, 0x2C, 0x3F, 0x44, 0x51, 0x2F, 0x1F, 0x1F, 0x20, 0x23

    LCD\_WR\_REG(0xe1);

    LCD\_WR\_DATA(0xd0);

    LCD\_WR\_DATA(0x00);

    LCD\_WR\_DATA(0x02);

    LCD\_WR\_DATA(0x07);

    LCD\_WR\_DATA(0x0a);

    LCD\_WR\_DATA(0x28);

    LCD\_WR\_DATA(0x31);

    LCD\_WR\_DATA(0x54);

    LCD\_WR\_DATA(0x47);

    LCD\_WR\_DATA(0x0e);

    LCD\_WR\_DATA(0x1c);

    LCD\_WR\_DATA(0x17);

    LCD\_WR\_DATA(0x1b);

    LCD\_WR\_DATA(0x1e);

    LCD\_WR\_REG(0x29);

}

/\* 移植lcd.c中的功能函数 \*/

/\*\*

 \* @brief 设置光标位置

 \* @param Xpos X坐标

 \* @param Ypos Y坐标

 \*

 \* 功能说明：

 \* 1. 设置X方向光标位置

 \* 2. 设置Y方向光标位置

 \* 3. 用于后续数据写入定位

 \*/

void LCD\_SetCursor(uint16\_t Xpos, uint16\_t Ypos)

{

    LCD\_WR\_REG(lcddev.setxcmd);

    LCD\_WR\_DATA(Xpos >> 8);

    LCD\_WR\_DATA(Xpos & 0X00FF);

    LCD\_WR\_DATA(Xpos >> 8);

    LCD\_WR\_DATA(Xpos & 0X00FF);

    LCD\_WR\_REG(lcddev.setycmd);

    LCD\_WR\_DATA(Ypos >> 8);

    LCD\_WR\_DATA(Ypos & 0X000FF);

    LCD\_WR\_DATA(Ypos >> 8);

    LCD\_WR\_DATA(Ypos & 0X00FF);

}

/\* 设置窗口函数 - 优化为原厂风格 \*/

/\*\*

 \* @brief 设置LCD显示窗口

 \* @param sx 起始X坐标

 \* @param sy 起始Y坐标

 \* @param width 窗口宽度

 \* @param height 窗口高度

 \*

 \* 功能说明：

 \* 1. 设置X方向起始和结束地址

 \* 2. 设置Y方向起始和结束地址

 \* 3. 准备GRAM写入

 \*/

static void LCD\_SetWindows(uint16\_t sx, uint16\_t sy, uint16\_t width, uint16\_t height)

{

    width = sx + width - 1;

    height = sy + height - 1;

    LCD\_WR\_REG(lcddev.setxcmd);

    LCD\_WR\_DATA(sx >> 8);

    LCD\_WR\_DATA(sx & 0X00FF);

    LCD\_WR\_DATA(width >> 8);

    LCD\_WR\_DATA(width & 0X00FF);

    LCD\_WR\_REG(lcddev.setycmd);

    LCD\_WR\_DATA(sy >> 8);

    LCD\_WR\_DATA(sy & 0X00FF);

    LCD\_WR\_DATA(height >> 8);

    LCD\_WR\_DATA(height & 0X00FF);

}

/\* 兼容旧接口 \*/

/\*\*

 \* @brief 设置显示窗口(兼容旧接口)

 \* @param sx 起始X坐标

 \* @param sy 起始Y坐标

 \* @param ex 结束X坐标

 \* @param ey 结束Y坐标

 \*

 \* 功能说明：

 \* 1. 转换为宽度/高度格式

 \* 2. 调用新式窗口设置函数

 \*/

static void LCD\_SetWindow(u16 sx, u16 sy, u16 ex, u16 ey)

{

    LCD\_SetWindows(sx, sy, ex - sx + 1, ey - sy + 1);

}

/\*\*

 \* @brief 清屏函数

 \* @param Color 填充颜色(RGB565)

 \*

 \* 功能说明：

 \* 1. 设置全屏窗口

 \* 2. 循环写入颜色数据

 \* 3. 用于快速清屏

 \*/

void LCD\_Clear(uint16\_t Color)

{

    uint32\_t index;

    uint32\_t total = lcddev.width \* lcddev.height;

    LCD\_SetWindows(0, 0, lcddev.width, lcddev.height);

    LCD\_WriteRAM\_Prepare(); // 开始写入GRAM

#if LCD\_DMA\_ENABLE

    // 使用DMA方式快速清屏

    if (total > 0 && dma\_buffer != NULL)

    {

        // 填充缓冲区

        uint32\_t i;

        uint32\_t buffer\_size = total > LCD\_DMA\_BUFFER\_SIZE ? LCD\_DMA\_BUFFER\_SIZE : total;

        for (i = 0; i < buffer\_size; i++)

        {

            dma\_buffer[i] = Color;

        }

        // 分块传输所有像素

        uint32\_t remaining = total;

        while (remaining > 0)

        {

            uint32\_t chunk\_size = remaining > LCD\_DMA\_BUFFER\_SIZE ? LCD\_DMA\_BUFFER\_SIZE : remaining;

            lcd\_dma\_transmit(dma\_buffer, chunk\_size);

            remaining -= chunk\_size;

        }

        return;

    }

#endif

    // 传统方式清屏(当DMA不可用时)

    for (index = 0; index < total; index++)

    {

#if (FAST == 0)

        LCD\_WriteRAM(Color);  // 修复变量名错误：color -> Color

#else

        // 使用以下代码可以提速

        uint8\_t buf[2] = {Color >> 8, Color & 0xFF};

        rt\_spi\_send(lcd\_dev, buf, 2);

#endif

    }

}

/\* 保留其他图形绘制函数... \*/

/\* RT-Thread设备初始化 \*/

static void lcd\_pin\_init(void)

{

    rt\_pin\_mode(LCD\_DC\_PIN, PIN\_MODE\_OUTPUT);

    rt\_pin\_mode(LCD\_RES\_PIN, PIN\_MODE\_OUTPUT);

    rt\_pin\_mode(LCD\_BLK\_PIN, PIN\_MODE\_OUTPUT);

    rt\_pin\_mode(LCD\_CS\_PIN, PIN\_MODE\_OUTPUT);

}

int spi\_lcd\_init(void)

{

    rt\_err\_t res = RT\_EOK;

    /\* 初始化引脚 \*/

    lcd\_pin\_init();

    /\* 附加SPI设备 \*/

    rt\_hw\_spi\_device\_attach(LCD\_SPI\_BUS, "spi\_lcd", LCD\_CS\_PIN);

    /\* 查找SPI设备 \*/

    lcd\_dev = (struct rt\_spi\_device \*)rt\_device\_find("spi\_lcd");

    if (lcd\_dev != RT\_NULL)

    {

        /\* 配置SPI参数 \*/

        struct rt\_spi\_configuration spi\_config;

        spi\_config.data\_width = 8;

        spi\_config.max\_hz = 25 \* 1000 \* 1000; /\* 25MHz \*/

        spi\_config.mode = RT\_SPI\_MASTER | RT\_SPI\_MODE\_0 | RT\_SPI\_MSB;

        rt\_spi\_configure(lcd\_dev, &spi\_config);

    }

    else

    {

        res = -RT\_ERROR;

        LOG\_E("SPI device not found!");

    }

    // /\* 初始化LCD设备参数 \*/

    // lcddev.width = LCD\_W;

    // lcddev.height = LCD\_H;

    // lcddev.wramcmd = 0x2C;

    // lcddev.setxcmd = 0x2A;

    // lcddev.setycmd = 0x2B;

    // lcddev.dir = 1; // 默认横屏(与头文件定义一致)

    /\* 初始化LCD \*/

    /\* RT-Thread适配的复位时序 \*/

    // LCD\_RES\_SET;

    rt\_thread\_mdelay(25);

    LCD\_RES\_CLR;

    rt\_thread\_mdelay(25);

    LCD\_RES\_SET;

    rt\_thread\_mdelay(50);

    /\* 执行LCD初始化序列 \*/

    LCD\_INIT\_CODE(Landscape);

    /\* 设置显示方向(使用头文件定义的默认方向) \*/

    LOG\_D("Setting default display direction: %s", Landscape == LANDSCAPE ? "LANDSCAPE" : "PORTRAIT");

    // LCD\_Display\_Dir(Landscape);

    rt\_thread\_mdelay(10); // 确保方向设置完成

    // /\* 背光控制 - 完全按照lcd.h定义 \*/

    // #if (BACKLIGHT\_ACTIVE\_LEVEL == 1)

    //     LCD\_BLK\_SET; // 高电平点亮

    // #else

    //     LCD\_BLK\_CLR; // 低电平点亮

    // #endif

    //     /\* 详细初始化流程 \*/

    //     LOG\_D("--- Starting LCD Initialization ---");

    //     // 1. 复位序列

    //     LOG\_D("1. Reset sequence");

    //     LCD\_RES\_SET;

    //     rt\_thread\_mdelay(10);

    //     LCD\_RES\_CLR;

    //     rt\_thread\_mdelay(20);

    //     LCD\_RES\_SET;

    //     rt\_thread\_mdelay(120); // 延长复位后等待时间

    //     // 2. 发送初始化命令

    //     LOG\_D("2. Sending init commands");

    //     LCD\_INIT\_CODE();

    //     // 3. 设置颜色接口(0x3A) - 完全按照原厂设置

    //     LOG\_D("3. Set color format");

    //     LCD\_WR\_REG(0x3A);

    //     LCD\_WR\_DATA(0x05); // RGB565格式

    //     rt\_thread\_mdelay(10);

    //     // 4. 设置扫描方向(0x36)

    //     LOG\_D("4. Set scan direction");

    //     LCD\_WR\_REG(0x36);

    //     LCD\_WR\_DATA(0x00); // 竖屏模式

    //     rt\_thread\_mdelay(10);

    //     // 5. 开启显示(0x29)

    //     LOG\_D("5. Display ON");

    //     LCD\_WR\_REG(0x29);

    //     rt\_thread\_mdelay(100); // 确保显示完全开启

    //     LOG\_D("--- LCD Initialization Complete ---");

    //     LOG\_I("LCD init success");

    //     return RT\_EOK;

}

INIT\_COMPONENT\_EXPORT(spi\_lcd\_init);

/\* 移植图形绘制函数 \*/

/\*\*

 \* @brief 在指定位置画点

 \* @param x X坐标

 \* @param y Y坐标

 \* @param color 点的颜色(RGB565)

 \*

 \* 功能说明：

 \* 1. 设置光标位置

 \* 2. 准备GRAM写入

 \* 3. 写入颜色数据

 \*/

void LCD\_DrawPoint(u16 x, u16 y, u16 color)

{

    LCD\_SetCursor(x, y);

    LCD\_WriteRAM\_Prepare();

    LCD\_WriteRAM(color);

}

void LCD\_Fill(u16 sx, u16 sy, u16 ex, u16 ey, u16 color)

{

    u16 i, j;

    u16 width = ex - sx + 1;

    u16 height = ey - sy + 1;

    uint32\_t total\_pixels = width \* height;

    LCD\_SetWindows(sx, sy, width, height);

    LCD\_WriteRAM\_Prepare(); // 开始写入GRAM

#if LCD\_DMA\_ENABLE

    // 使用DMA方式填充

    if (total\_pixels > 3) // 对于小区域，直接使用传统方法

    {

        // 创建临时缓冲区

        uint16\_t buffer\_size = total\_pixels > LCD\_DMA\_BUFFER\_SIZE ? LCD\_DMA\_BUFFER\_SIZE : total\_pixels;

        // 填充DMA缓冲区，所有像素都是相同颜色

        for (i = 0; i < buffer\_size; i++)

        {

            dma\_buffer[i] = color;

        }

        // 使用DMA发送数据

        uint32\_t remaining = total\_pixels;

        while (remaining > 0)

        {

            uint32\_t chunk\_size = remaining > LCD\_DMA\_BUFFER\_SIZE ? LCD\_DMA\_BUFFER\_SIZE : remaining;

            lcd\_dma\_transmit(dma\_buffer, chunk\_size);

            remaining -= chunk\_size;

        }

        return;

    }

#endif

    // 传统方式填充(当DMA被禁用或者区域很小时使用)

    for (i = 0; i < height; i++)

    {

        for (j = 0; j < width; j++)

        {

            LCD\_WR\_DATA\_16BIT(color);

        }

    }

}

/\* 字体显示函数 \*/

/\*\*

 \* @brief 显示单个字符

 \* @param x X坐标

 \* @param y Y坐标

 \* @param chr 要显示的字符

 \* @param size 字体大小(12/16)

 \* @param mode 显示模式(0-覆盖模式,1-叠加模式)

 \* @param color 字符颜色

 \*

 \* 功能说明：

 \* 1. 计算字符在字库中的偏移

 \* 2. 获取字模数据

 \* 3. 按位绘制字符点阵

 \*/

void LCD\_ShowChar(uint16\_t x, uint16\_t y, char chr, uint8\_t size, uint8\_t mode, uint16\_t color, uint16\_t bg\_color)

{

    uint8\_t temp, t1, t;

    uint16\_t y0 = y;

    uint8\_t csize = (size / 8 + ((size % 8) ? 1 : 0)) \* (size / 2);

    uint8\_t \*pfont = 0;

    // 字符偏移量计算

    chr = chr - ' ';

    LOG\_D("Drawing char '%c'(%d) at (%d,%d) size:%d", chr + ' ', chr + ' ', x, y, size);

    // 获取字体数据指针

    switch (size)

    {

    case 12:

        pfont = (uint8\_t \*)asc2\_1206[chr];

        break;

    case 16:

        pfont = (uint8\_t \*)asc2\_1608[chr];

        break;

    default:

        LOG\_E("Unsupported font size: %d", size);

        return;

    }

    for (t = 0; t < csize; t++)

    {

        temp = pfont[t]; /\* 获取字符的点阵数据 \*/

        for (t1 = 0; t1 < 8; t1++) /\* 一个字节8个点 \*/

        {

            if (temp & 0x80) /\* 有效点,需要显示 \*/

            {

                LCD\_DrawPoint(x, y, color); /\* 画点出来,要显示这个点 \*/

            }

            else if (mode == 0) /\* 无效点,不显示 \*/

            {

                LCD\_DrawPoint(x, y, bg\_color); /\* 画指定背景色 \*/

            }

            temp <<= 1; /\* 移位, 以便获取下一个位的状态 \*/

            y++;

            if (y >= lcddev.height)

                return; /\* 超区域了 \*/

            if ((y - y0) == size) /\* 显示完一列了? \*/

            {

                y = y0; /\* y坐标复位 \*/

                x++;    /\* x坐标递增 \*/

                if (x >= lcddev.width)

                    return; /\* x坐标超区域了 \*/

                break;

            }

        }

    }

}

/\*\*

 \* @brief 在LCD上显示字符串

 \* @param x 起始X坐标(像素)

 \* @param y 起始Y坐标(像素)

 \* @param width 显示区域宽度(像素)

 \* @param height 显示区域高度(像素)

 \* @param size 字体大小(12/16/24等)

 \* @param p 要显示的字符串指针

 \* @param color 文字颜色(RGB565格式)

 \* @param bg\_color 背景颜色(RGB565格式)

 \*

 \* 功能说明：

 \* 1. 在指定区域内显示字符串

 \* 2. 自动换行处理(当超出width时换到下一行)

 \* 3. 支持ASCII字符(32~126)

 \* 4. 超出显示区域的内容会被裁剪

 \* 5. 可自定义文字和背景颜色

 \*

 \* 注意：

 \* - 使用前需先初始化LCD

 \* - 确保坐标和区域大小在屏幕范围内

 \* - 背景色参数为新增功能，旧代码需要更新

 \*/

void LCD\_ShowString(u16 x, u16 y, u16 width, u16 height, u8 size, u8 \*p, u16 color, uint16\_t bg\_color)

{

    u8 x0 = x;

    width += x;

    height += y;

    while ((\*p <= '~') && (\*p >= ' ')) // 判断是不是非法字符!

    {

        if (x >= width)

        {

            x = x0;

            y += size;

        }

        if (y >= height)

            break; // 退出

        LCD\_ShowChar(x, y, \*p, size, 0, color, bg\_color);

        x += size / 2;

        p++;

    }

}

/\*\*

 \* @brief 显示数字

 \* @param x X坐标

 \* @param y Y坐标

 \* @param num 要显示的数字

 \* @param len 数字长度

 \* @param size 字体大小

 \* @param color 颜色

 \*

 \* 功能说明：

 \* 1. 支持固定长度显示

 \* 2. 自动处理前导零

 \* 3. 调用字符显示函数实现

 \*/

/\*\*

 \* @brief 显示数字

 \* @param x X坐标

 \* @param y Y坐标

 \* @param num 要显示的数字

 \* @param len 数字长度

 \* @param size 字体大小

 \* @param color 文字颜色

 \* @param bg\_color 背景颜色

 \*

 \* 功能说明：

 \* 1. 支持固定长度显示

 \* 2. 自动处理前导零

 \* 3. 可自定义文字和背景颜色

 \*/

void LCD\_ShowNum(u16 x, u16 y, u32 num, u8 len, u8 size, u16 color, uint16\_t bg\_color)

{

    u8 t, temp;

    u8 enshow = 0;

    for (t = 0; t < len; t++)

    {

        temp = (num / LCD\_Pow(10, len - t - 1)) % 10;

        if (enshow == 0 && t < (len - 1))

        {

            if (temp == 0)

            {

                LCD\_ShowChar(x + (size / 2) \* t, y, ' ', size, 0, color, bg\_color);

                continue;

            }

            else

                enshow = 1;

        }

        LCD\_ShowChar(x + (size / 2) \* t, y, temp + '0', size, 0, color, bg\_color);

    }

}

/\*\*

 \* @brief 显示数字(增强版)

 \* @param x X坐标

 \* @param y Y坐标

 \* @param num 要显示的数字

 \* @param len 数字长度

 \* @param size 字体大小

 \* @param mode 显示模式

 \* @param color 颜色

 \*

 \* 功能说明：

 \* 1. 支持多种显示模式

 \* 2. 可控制前导零显示

 \* 3. 调用字符显示函数实现

 \*/

void LCD\_ShowxNum(u16 x, u16 y, u32 num, u8 len, u8 size, u8 mode, u16 color, uint16\_t bg\_color)

{

    u8 t, temp;

    u8 enshow = 0;

    for (t = 0; t < len; t++)

    {

        temp = (num / LCD\_Pow(10, len - t - 1)) % 10;

        if (enshow == 0 && t < (len - 1))

        {

            if (temp == 0)

            {

                if (mode & 0X80)

                    LCD\_ShowChar(x + (size / 2) \* t, y, '0', size, mode & 0X01, color, bg\_color);

                else

                    LCD\_ShowChar(x + (size / 2) \* t, y, ' ', size, mode & 0X01, color, BACK\_COLOR);

                continue;

            }

            else

                enshow = 1;

        }

        LCD\_ShowChar(x + (size / 2) \* t, y, temp + '0', size, mode & 0X01, color, BACK\_COLOR);

    }

}

// m^n函数

// 返回值:m^n次方.

/\*\*

 \* @brief 计算m的n次方

 \* @param m 底数

 \* @param n 指数

 \* @return m^n结果

 \*

 \* 功能说明：

 \* 1. 用于数字显示计算

 \* 2. 简单循环实现

 \*/

u32 LCD\_Pow(u8 m, u8 n)

{

    u32 result = 1;

    while (n--)

        result \*= m;

    return result;

}

/\* 图片显示函数 \*/

/\*\*

 \* @brief 显示图片

 \* @param x X坐标

 \* @param y Y坐标

 \* @param width 图片宽度

 \* @param height 图片高度

 \* @param p 图片数据指针(RGB565格式)

 \*

 \* 功能说明：

 \* 1. 设置显示窗口

 \* 2. 连续写入像素数据

 \* 3. 支持RGB565格式图片

 \*/

void LCD\_ShowImage(u16 x, u16 y, u16 width, u16 height, const u16 \*p)

{

    u32 i;

    u16 color;

    LCD\_SetWindows(x, y, width, height);

    for (i = 0; i < width \* height; i++)

    {

        color = \*p++;

        LCD\_WR\_DATA\_16BIT(color);

    }

}

// 画线

// x1,y1:起点坐标

// x2,y2:终点坐标

// color:颜色

/\*\*

 \* @brief 画线函数

 \* @param x1 起点X坐标

 \* @param y1 起点Y坐标

 \* @param x2 终点X坐标

 \* @param y2 终点Y坐标

 \* @param color 线条颜色

 \*

 \* 功能说明：

 \* 1. 使用Bresenham算法

 \* 2. 支持任意方向直线

 \* 3. 调用画点函数实现

 \*/

void LCD\_DrawLine(u16 x1, u16 y1, u16 x2, u16 y2, u16 color)

{

    u16 t;

    int xerr = 0, yerr = 0, delta\_x, delta\_y, distance;

    int incx, incy, uRow, uCol;

    delta\_x = x2 - x1; // 计算坐标增量

    delta\_y = y2 - y1;

    uRow = x1;

    uCol = y1;

    if (delta\_x > 0)

        incx = 1; // 设置单步方向

    else if (delta\_x == 0)

        incx = 0; // 垂直线

    else

    {

        incx = -1;

        delta\_x = -delta\_x;

    }

    if (delta\_y > 0)

        incy = 1;

    else if (delta\_y == 0)

        incy = 0; // 水平线

    else

    {

        incy = -1;

        delta\_y = -delta\_y;

    }

    if (delta\_x > delta\_y)

        distance = delta\_x; // 选取基本增量坐标轴

    else

        distance = delta\_y;

    for (t = 0; t <= distance + 1; t++) // 画线输出

    {

        LCD\_DrawPoint(uRow, uCol, color); // 画点

        xerr += delta\_x;

        yerr += delta\_y;

        if (xerr > distance)

        {

            xerr -= distance;

            uRow += incx;

        }

        if (yerr > distance)

        {

            yerr -= distance;

            uCol += incy;

        }

    }

}

// 画矩形

//(x1,y1),(x2,y2):矩形的对角坐标

/\*\*

 \* @brief 画矩形

 \* @param x1 左上角X坐标

 \* @param y1 左上角Y坐标

 \* @param x2 右下角X坐标

 \* @param y2 右下角Y坐标

 \* @param color 边框颜色

 \*

 \* 功能说明：

 \* 1. 调用画线函数实现

 \* 2. 绘制四条边组成矩形

 \*/

void LCD\_DrawRectangle(u16 x1, u16 y1, u16 x2, u16 y2, u16 color)

{

    LCD\_DrawLine(x1, y1, x2, y1, color);

    LCD\_DrawLine(x1, y1, x1, y2, color);

    LCD\_DrawLine(x1, y2, x2, y2, color);

    LCD\_DrawLine(x2, y1, x2, y2, color);

}

// 在指定位置画一个指定大小的圆

//(x,y):中心点

// r    :半径

/\*\*

 \* @brief 画圆

 \* @param x0 圆心X坐标

 \* @param y0 圆心Y坐标

 \* @param r 半径

 \* @param color 圆颜色

 \*

 \* 功能说明：

 \* 1. 使用Bresenham算法

 \* 2. 对称绘制8个点

 \* 3. 调用画点函数实现

 \*/

void Draw\_Circle(u16 x0, u16 y0, u8 r, u16 color)

{

    int a, b;

    int di;

    a = 0;

    b = r;

    di = 3 - (r << 1); // 判断下个点位置的标志

    while (a <= b)

    {

        LCD\_DrawPoint(x0 + a, y0 - b, color); // 5

        LCD\_DrawPoint(x0 + b, y0 - a, color); // 0

        LCD\_DrawPoint(x0 + b, y0 + a, color); // 4

        LCD\_DrawPoint(x0 + a, y0 + b, color); // 6

        LCD\_DrawPoint(x0 - a, y0 + b, color); // 1

        LCD\_DrawPoint(x0 - b, y0 + a, color);

        LCD\_DrawPoint(x0 - a, y0 - b, color); // 2

        LCD\_DrawPoint(x0 - b, y0 - a, color); // 7

        a++;

        // 使用Bresenham算法画圆

        if (di < 0)

            di += 4 \* a + 6;

        else

        {

            di += 10 + 4 \* (a - b);

            b--;

        }

    }

}

// 设置LCD的自动扫描方向

// dir:0~7,代表8个方向(具体定义见lcd.h)

/\*\*

 \* @brief 设置LCD扫描方向

 \* @param dir 方向参数(0~7)

 \*

 \* 功能说明：

 \* 1. 配置扫描方向寄存器

 \* 2. 影响显示内容方向

 \* 3. 支持8种方向设置

 \*/

void LCD\_Scan\_Dir(u8 dir)

{

    u16 regval = 0;

    u8 dirreg = 0;

    //  u16 temp;

    /\* 方向参数转换 - 与原版lcd.c保持一致 \*/

    switch (dir)

    {

    case 0:

        dir = 6;

        break;

    case 1:

        dir = 7;

        break;

    case 2:

        dir = 4;

        break;

    case 3:

        dir = 5;

        break;

    case 4:

        dir = 1;

        break;

    case 5:

        dir = 0;

        break;

    case 6:

        dir = 3;

        break;

    case 7:

        dir = 2;

        break;

    }

    rt\_kprintf("dir:%d\n", dir);

    switch (dir)

    {

    case L2R\_U2D: // 从左到右,从上到下

        regval = (0 << 7) | (0 << 6) | (0 << 5);

        break;

    case L2R\_D2U: // 从左到右,从下到上

        regval = (1 << 7) | (0 << 6) | (0 << 5);

        break;

    case R2L\_U2D: // 从右到左,从上到下

        regval = (0 << 7) | (1 << 6) | (0 << 5);

        break;

    case R2L\_D2U: // 从右到左,从下到上

        regval = (1 << 7) | (1 << 6) | (0 << 5);

        break;

    case U2D\_L2R: // 从上到下,从左到右

        regval = (0 << 7) | (0 << 6) | (1 << 5);

        break;

    case U2D\_R2L: // 从上到下,从右到左

        regval = (0 << 7) | (1 << 6) | (1 << 5);

        break;

    case D2U\_L2R: // 从下到上,从左到右

        regval = (1 << 7) | (0 << 6) | (1 << 5);

        break;

    case D2U\_R2L: // 从下到上,从右到左

        regval = (1 << 7) | (1 << 6) | (1 << 5);

        break;

    }

    dirreg = 0X36;

    regval |= 0x00; // 0x08 0x00  红蓝反色可以通过这里修改

    rt\_kprintf("regval:%x\n", regval);

    LCD\_WriteReg(dirreg, regval);

    LCD\_WR\_REG(lcddev.setxcmd);

    LCD\_WR\_DATA(0);

    LCD\_WR\_DATA(0);

    LCD\_WR\_DATA((lcddev.width - 1) >> 8);

    LCD\_WR\_DATA((lcddev.width - 1) & 0XFF);

    LCD\_WR\_REG(lcddev.setycmd);

    LCD\_WR\_DATA(0);

    LCD\_WR\_DATA(0);

    LCD\_WR\_DATA((lcddev.height - 1) >> 8);

    LCD\_WR\_DATA((lcddev.height - 1) & 0XFF);

}

// 设置LCD显示方向（6804不支持横屏显示）

// dir:0,竖屏；1,横屏

/\*\*

 \* @brief 设置LCD显示方向

 \* @param dir 方向(0-竖屏,1-横屏)

 \*

 \* 功能说明：

 \* 1. 配置显示参数

 \* 2. 更新宽高值

 \* 3. 设置默认扫描方向

 \*/

void LCD\_Display\_Dir(u8 dir)

{

    if (dir == 0)

    {

        lcddev.dir = 0; // 竖屏

        lcddev.width = LCD\_W;

        lcddev.height = LCD\_H;

        lcddev.wramcmd = 0X2C;

        lcddev.setxcmd = 0X2A;

        lcddev.setycmd = 0X2B;

        DFT\_SCAN\_DIR = U2D\_R2L; // 竖显-设定显示方向

    }

    else // 横屏

    {

        lcddev.dir = 1;

        lcddev.width = LCD\_H;

        lcddev.height = LCD\_W;

        lcddev.wramcmd = 0X2C;

        lcddev.setxcmd = 0X2A;

        lcddev.setycmd = 0X2B;

        DFT\_SCAN\_DIR = L2R\_U2D; // 横显-设定显示方向

    }

    LCD\_Scan\_Dir(DFT\_SCAN\_DIR); // 默认扫描方向

}

// 在指定区域内填充指定颜色块

//(sx,sy),(ex,ey):填充矩形对角坐标,区域大小为:(ex-sx+1)\*(ey-sy+1)

// color:要填充的颜色

/\*\*

 \* @brief 颜色块填充

 \* @param sx 起始X坐标

 \* @param sy 起始Y坐标

 \* @param ex 结束X坐标

 \* @param ey 结束Y坐标

 \* @param color 颜色数据指针

 \*

 \* 功能说明：

 \* 1. 设置显示窗口

 \* 2. 按行填充颜色数据

 \* 3. 支持任意矩形区域

 \*/

void LCD\_Color\_Fill(u16 sx, u16 sy, u16 ex, u16 ey, u16 \*color)

{

    u16 height, width;

    u16 i, j;

    width = ex - sx + 1;  // 得到填充的宽度

    height = ey - sy + 1; // 高度

    for (i = 0; i < height; i++)

    {

        LCD\_SetCursor(sx, sy + i); // 设置光标位置

        LCD\_WriteRAM\_Prepare();    // 开始写入GRAM

        for (j = 0; j < width; j++)

            LCD\_WriteRAM(color[i \* height + j]); // 写入数据

    }

}

/\* 快速方向切换函数实现 \*/

void LCD\_SetPortrait(void)

{

    LCD\_Scan\_Dir(PORTRAIT);

    lcddev.dir = 0; // 竖屏

    lcddev.width = LCD\_W;

    lcddev.height = LCD\_H;

}

void LCD\_SetLandscape(void)

{

    LCD\_Scan\_Dir(LANDSCAPE);

    lcddev.dir = 1; // 横屏

    lcddev.width = LCD\_H;

    lcddev.height = LCD\_W;

}

/\* 测试命令 \*/

static uint16\_t color\_array[] = {

    WHITE, BLACK, BLUE, RED,

    GREEN, YELLOW};

/\*\*

 \* @brief 绘制测试图案

 \*

 \* 功能说明：

 \* 1. 绘制9宫格色块

 \* 2. 每种颜色不同亮度

 \* 3. 用于LCD功能测试

 \*/

static void LCD\_DrawTestPattern(void)

{

    LOG\_D("Starting optimized test pattern...");

/\* 确保背光开启 \*/

#if (BACKLIGHT\_ACTIVE\_LEVEL == 1)

    LCD\_BLK\_SET;

#else

    LCD\_BLK\_CLR;

#endif

    /\* 先清屏为白色 \*/

    LCD\_Clear(WHITE);

    rt\_thread\_mdelay(100);

    /\* 计算区域尺寸 \*/

    uint16\_t w = lcddev.width / 3;

    uint16\_t h = lcddev.height / 3;

    /\* 绘制9个色块 \*/

    for (int row = 0; row < 3; row++)

    {

        for (int col = 0; col < 3; col++)

        {

            uint16\_t color;

            /\* 确定基础颜色 \*/

            switch (col)

            {

            case 0:

                color = RED;

                break;

            case 1:

                color = GREEN;

                break;

            case 2:

                color = BLUE;

                break;

            }

            /\* 调整亮度 \*/

            switch (row)

            {

            case 0:

                color |= 0xFFFF;

                break; // 全亮

            case 1:

                color &= 0xFFE0;

                break; // 半亮

            case 2:

                color &= 0xF800;

                break; // 暗

            }

            /\* 计算区域坐标 \*/

            uint16\_t x1 = col \* w;

            uint16\_t y1 = row \* h;

            uint16\_t x2 = (col == 2) ? lcddev.width - 1 : (col + 1) \* w - 1;

            uint16\_t y2 = (row == 2) ? lcddev.height - 1 : (row + 1) \* h - 1;

            LOG\_D("Drawing block %d-%d: (%d,%d)-(%d,%d) color=0x%04X",

                  row, col, x1, y1, x2, y2, color);

            /\* 填充区域 \*/

            LCD\_Fill(x1, y1, x2, y2, color);

            /\* 添加短暂延时防止SPI过载 \*/

            rt\_thread\_mdelay(10);

        }

    }

    LOG\_D("Test pattern complete");

}

/\*\*

 \* @brief LCD测试命令

 \* @return 0-成功 -1-失败

 \*

 \* 功能说明：

 \* 1. 测试字符显示功能

 \* 2. 显示不同大小和颜色的文本

 \* 3. 可通过RT-Thread MSH调用

 \*/

int lcd(void)

{

    // if (argc > 1)

    // {

    //     rt\_kprintf("Usage: lcd\_test\n");

    //     return -1;

    // }

    // if (!lcd\_dev)

    // {

    //     rt\_kprintf("LCD not initialized, run 'rt\_hw\_lcd\_init' first\n");

    //     return -1;

    // }

    //    /\* 显式设置为横屏模式 \*/

    //    rt\_kprintf("Setting display to landscape mode...\n");

    //    LCD\_SetLandscape();

    //    rt\_thread\_mdelay(100); // 确保方向设置完成

    //        /\* 测试1: 纯色填充 \*/

    //            for (int i = 0; i < sizeof(color\_array) / sizeof(color\_array[0]); i++)

    //        {

    //                    LCD\_Clear(color\_array[i]);

//                    rt\_kprintf("Test1: Fill color 0x%04X\n", color\_array[i]);

//                    rt\_thread\_mdelay(500);

    //        }

    //

    //        /\* 测试2: 绘制测试图案 \*/

    //        LCD\_Clear(BLACK);

    //            rt\_kprintf("Test2: Drawing test pattern...\n");

//            LCD\_DrawTestPattern();

//            rt\_thread\_mdelay(1000);

//            /\* 测试3: 绘制渐变 - 优化为分块绘制 \*/

    // LCD\_Clear(BLACK);

    //        rt\_kprintf("Test3: Drawing optimized gradient...\n");

    //

    //        /\* 强制设置扫描方向为从左到右，从上到下 \*/

    //        rt\_thread\_mdelay(10); // 确保方向设置完成

    //

    //     #define GRADIENT\_BLOCK\_SIZE 16 // 16x16像素块

    //

    //        for (uint16\_t block\_y = 0; block\_y < lcddev.height; block\_y += GRADIENT\_BLOCK\_SIZE)

    //        {

    //            uint16\_t end\_y = (block\_y + GRADIENT\_BLOCK\_SIZE > lcddev.height) ? lcddev.height : block\_y + GRADIENT\_BLOCK\_SIZE;

    //

    //            for (uint16\_t block\_x = 0; block\_x < lcddev.width; block\_x += GRADIENT\_BLOCK\_SIZE)

    //            {

    //                uint16\_t end\_x = (block\_x + GRADIENT\_BLOCK\_SIZE > lcddev.width) ? lcddev.width : block\_x + GRADIENT\_BLOCK\_SIZE;

    //

    //                // 计算块四角颜色(双线性插值)

    //                uint16\_t x1 = block\_x;

    //                uint16\_t y1 = block\_y;

    //                uint16\_t x2 = (block\_x + GRADIENT\_BLOCK\_SIZE >= lcddev.width) ? lcddev.width - 1 : block\_x + GRADIENT\_BLOCK\_SIZE;

    //                uint16\_t y2 = (block\_y + GRADIENT\_BLOCK\_SIZE >= lcddev.height) ? lcddev.height - 1 : block\_y + GRADIENT\_BLOCK\_SIZE;

    //

    //                // 计算四角颜色

    //                uint16\_t c00\_r = (x1 \* 31) / (lcddev.width - 1);

    //                uint16\_t c00\_g = (y1 \* 63) / (lcddev.height - 1);

    //                uint16\_t c00\_b = 31 - (x1 \* 31) / (lcddev.width - 1);

    //                uint16\_t c00 = (c00\_r << 11) | (c00\_g << 5) | c00\_b;

    //

    //                uint16\_t c01\_r = (x1 \* 31) / (lcddev.width - 1);

    //                uint16\_t c01\_g = (y2 \* 63) / (lcddev.height - 1);

    //                uint16\_t c01\_b = 31 - (x1 \* 31) / (lcddev.width - 1);

    //                uint16\_t c01 = (c01\_r << 11) | (c01\_g << 5) | c01\_b;

    //

    //                uint16\_t c10\_r = (x2 \* 31) / (lcddev.width - 1);

    //                uint16\_t c10\_g = (y1 \* 63) / (lcddev.height - 1);

    //                uint16\_t c10\_b = 31 - (x2 \* 31) / (lcddev.width - 1);

    //                uint16\_t c10 = (c10\_r << 11) | (c10\_g << 5) | c10\_b;

    //

    //                uint16\_t c11\_r = (x2 \* 31) / (lcddev.width - 1);

    //                uint16\_t c11\_g = (y2 \* 63) / (lcddev.height - 1);

    //                uint16\_t c11\_b = 31 - (x2 \* 31) / (lcddev.width - 1);

    //                uint16\_t c11 = (c11\_r << 11) | (c11\_g << 5) | c11\_b;

    //

    //                // 填充块(逐行绘制渐变)

    //                for (uint16\_t y = y1; y <= y2; y++)

    //                {

    //                    for (uint16\_t x = x1; x <= x2; x++)

    //                    {

    //                        // 双线性插值计算颜色

    //                        float tx = (float)(x - x1) / (x2 - x1);

    //                        float ty = (float)(y - y1) / (y2 - y1);

    //

    //                        uint16\_t r = (1 - tx) \* (1 - ty) \* c00\_r + tx \* (1 - ty) \* c10\_r +

    //                                     (1 - tx) \* ty \* c01\_r + tx \* ty \* c11\_r;

    //                        uint16\_t g = (1 - tx) \* (1 - ty) \* c00\_g + tx \* (1 - ty) \* c10\_g +

    //                                     (1 - tx) \* ty \* c01\_g + tx \* ty \* c11\_g;

    //                        uint16\_t b = (1 - tx) \* (1 - ty) \* c00\_b + tx \* (1 - ty) \* c10\_b +

    //                                     (1 - tx) \* ty \* c01\_b + tx \* ty \* c11\_b;

    //                        uint16\_t color = (r << 11) | (g << 5) | b;

    //

    //                        LCD\_DrawPoint(x, y, color);

    //                    }

    //                }

    //

    //                // 每5个块打印一次进度

    //                if ((block\_x / GRADIENT\_BLOCK\_SIZE) % 5 == 0)

    //                {

    //                    rt\_kprintf("Drawing block (%d,%d) color=0x%04X\n",

    //                               block\_x, block\_y);

    //                }

    //            }

    //

    //            // 每行块之间添加延时

    //            rt\_thread\_mdelay(5);

    //        }

    //        rt\_thread\_mdelay(500);

    /\* 测试4: 显示字符 \*/

// LCD\_Display\_Dir(0);

    // LCD\_Clear(BLACK);

//    rt\_kprintf("Test4: Drawing characters...\n");

    //

    //    // 显示不同大小和颜色的字符(仅测试支持的字体)

//    LCD\_ShowString(10, 10, 200, 16, 16, (u8 \*)"Moutain", WHITE, BACK\_COLOR);

//    LCD\_ShowString(10, 40, 200, 16, 16, (u8 \*)"is a", RED, BACK\_COLOR);

    //    LCD\_ShowString(10, 80, 200, 16, 16, (u8 \*)"bias", GREEN, BACK\_COLOR);

    //    LCD\_ShowString(10, 130, 200, 16, 16, (u8 \*)"I realy burned it out.", BLUE, BACK\_COLOR);

//

    //    // 显示ASCII字符表

    //    for (int i = 0; i < 5; i++)

    //    {

    //        char buf[20];

    //        rt\_snprintf(buf, sizeof(buf), "ASCII %d-%d", 32 + i \* 16, 47 + i \* 16);

    //            LCD\_ShowString(10, 160 + i \* 20, 200, 16, 16, (u8 \*)buf, YELLOW, BACK\_COLOR);

    //    }

    //    rt\_thread\_mdelay(2000);

    //

    //    /\* 测试5: 图形绘制测试 \*/

    //    // LCD\_Clear(BLACK);

    //    rt\_kprintf("\n=== Test5: Graphics drawing ===\n");

    //

    //        // 绘制线条

    //        for (int i = 0; i < 5; i++) {

    //            LCD\_DrawLine(10, 30 + i\*20, 200, 30 + i\*20, color\_array[i]);

    //        }

    //        rt\_kprintf("Lines drawn\n");

    //        rt\_thread\_mdelay(1000);

    //

    //        // 绘制矩形

    //        LCD\_DrawRectangle(50, 50, 150, 150, RED);

    //        LCD\_DrawRectangle(60, 60, 140, 140, GREEN);

    //        rt\_kprintf("Rectangles drawn\n");

    //        rt\_thread\_mdelay(1000);

    //

    //        // 绘制圆形

    //        for (int i = 0; i < 3; i++) {

    //            Draw\_Circle(100, 100, 30 + i\*10, BLUE);

    //        }

    //        rt\_kprintf("Circles drawn\n");

    //        rt\_thread\_mdelay(1000);

    //

    //    /\* 测试6: 扫描方向测试 \*/

    //    LCD\_Clear(BLACK);

    //    rt\_kprintf("\n=== Test6: Scan direction ===\n");

    //

    //    for (int dir = 0; dir < 8; dir++)

    //    {

    //        LCD\_Scan\_Dir(dir);

    //        LCD\_Clear(BLACK);

    //        LCD\_ShowString(10, 10, 200, 16, 16, (u8 \*)"Direction Test", WHITE, BACK\_COLOR);

    //        rt\_kprintf("Direction %d tested\n", dir);

    //        rt\_thread\_mdelay(500);

    //    }

    //    // 恢复默认方向

    //    LCD\_Scan\_Dir(DFT\_SCAN\_DIR);

    //    /\* 测试7: 性能测试 \*/

    //    LCD\_Clear(BLACK);

    //    rt\_kprintf("\n=== Test7: Performance test ===\n");

    //

    //    uint32\_t start = rt\_tick\_get();

    //    for (int i = 0; i < 1000; i++)

    //    {

    //        LCD\_DrawPoint(rand() % lcddev.width, rand() % lcddev.height, WHITE);

    //    }

    //    uint32\_t end = rt\_tick\_get();

    //    rt\_kprintf("1000 points drawn in %d ms\n", end - start);

    //    rt\_thread\_mdelay(1000);

    //

    //    return 0;

}

MSH\_CMD\_EXPORT(lcd, LCD test command : fill screen with different colors);

#endif