LoongIDE 1.1 的组件编程简介

目录

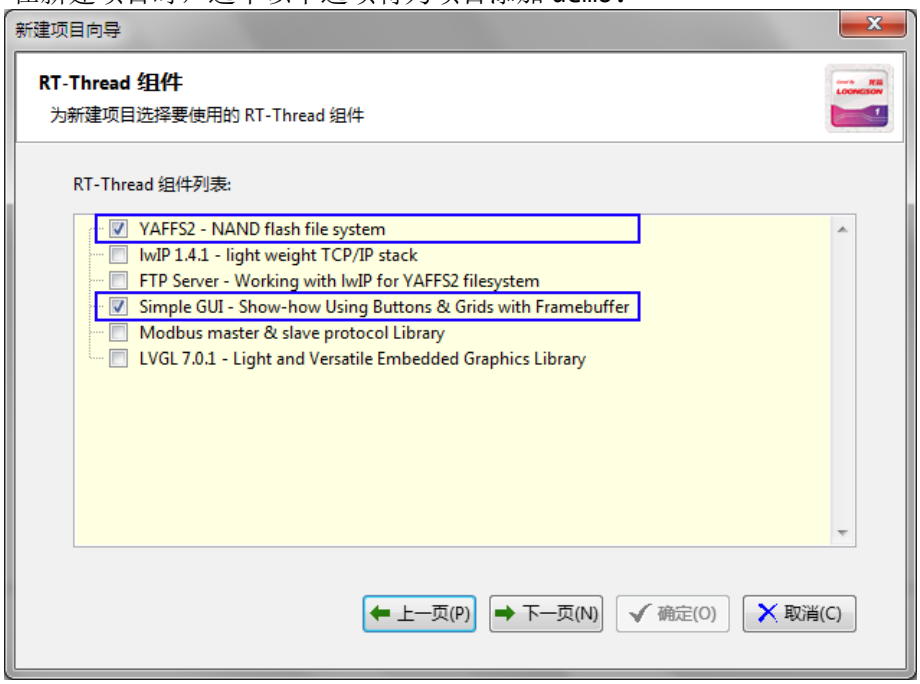
说明

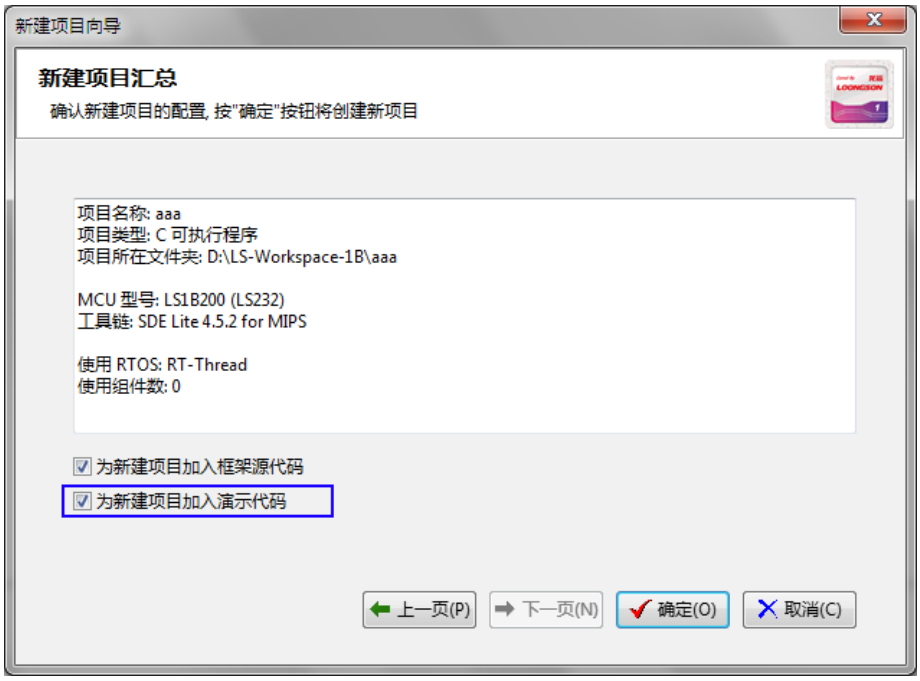
使用 loongide 1.0 版本创建的项目，需要用户在使用设备驱动程序、组件前显式调用各自的初始化函数。裸机、uCOSII、FreeRTOS 编程环境下需要先对内存堆的管理进行初始化，以实现动态内存分配。

使用 loongide 1.1 版本创建的项目，在 main()中调用函数 ls1x_drv_init()实现设备驱动程序的初始化，调用函数 install_3th_libraries()实现组件的初始化。具体的初始化设备、组件根据 bsp.h 的配置自动执行。

loongide 1.1 示例程序

在新建项目时，选中以下选项将为项目添加 demo.





一、YAFFS2 文件系统

YAFFS2 用于管理 NAND Flash 文件系统，loongide 已经为 LS1B200、LS1C300 移植好代码，用户可以直接使用。

1、移植说明

移植文件有三个，分别是：

①、yaffs2/port/yaffs_osglue.c

初始下 ysffs2 在裸机、RTOS 编程下的读写互斥量和内存申请释放操作。

②、yaffs2/port/ ls1x_nand_ecc.c

实现 NAND flash 读写时的 ecc 校验算法。

③、yaffs2/port/ ls1x_yaffs.c

实现 yaffs2 和 loongide 内置的 NAND flash 驱动程序交互接口。

同时实现 yaffs2 文件系统的初始化和挂载。

NAND flash 芯片初始化

```
typedef struct YAFFS_config
{
    unsigned    bytes_of_page;          // 每页字节数
    int         pages_of_block;         // 每块包含的页数
    int         oobbytes_of_page;       // 每页包含的 spare 字节数
    int         start_block;            // yaffs 文件系统起始块号
    int         end_block;              // yaffs 文件系统结束块号
    int         reserved_blocks;        // yaffs 文件系统保留块数
    int         cache_blocks;           // yaffs 文件系统缓存块数
    int         nand_flash_chip;        // NAND flash 芯片型号
    int         nand_rw_fullpage;       // =1: 对 NAND flash 芯片整页读写
    int         register_and_mount_fs;  // =1: 已挂载文件系统名称，例如 "/ndd"
    int         initialized;            // =1: 已经初始化
} YAFFS_config_t;
```

此结构必须根据使用的 NAND Flash 芯片参数填写。

目前支持 Samsung K9F1G08U 系列及兼容芯片。

初始化函数

```
/*
 * 功能：实现 yaffs2 初始化并挂载文件系统
 * 参数：mount_name   挂载的文件系统名称，比如 /ndd
 * 返回：0            成功
 */
int yaffs_startup_and_mount(const char *mount_name);
```

2、编程示例

①、配置 bsp.h

打开以下配置项：

```
#define BSP_USE_NAND
#define USE_YAFFS2
```

②、初始化

```
#ifdef BSP_USE_NAND
    ls1x_nand_init(devNAND, NULL);
#endif
#ifdef USE_YAFFS2
    yaffs_startup_and_mount(RYFS_MOUNTED_FS_NAME);
#endif
```

③、示例

```
#include "bsp.h"

#ifdef USE_YAFFS2

#include "ls1x_nand.h"
#include "yaffs2/port/ls1x_yaffs.h"
#include "yaffs2/direct/yaffsfs.h"
```

```c
/*
 * 打印一个目录下的内容。这是一个递归，注意栈溢出!
 */
static void dump_directory(const char *dname)
{
    yaffs_DIR *d;
    yaffs_dirent *de;
    struct yaffs_stat s;
    char str[100];

    d = yaffs_opendir(dname);          // 打开目录
    printk("\r\n");
    if (!d)
    {
        printk("opendir failed\n");
    }
    else
    {
        while ((de = yaffs_readdir(d)) != NULL)         // 读目录内容
        {
            sprintf(str, "%s/%s", dname, de->d_name);
            yaffs_lstat(str, &s);                       // 目录属性
            printk("%s inode %d length %d mode %X ", str, s.st_ino, (int)s.st_size, s.st_mode);
            switch (s.st_mode & S_IFMT)
            {
                case S_IFREG: printk("data file"); break;
                case S_IFDIR: printk("directory"); break;
                case S_IFLNK:
                    printk("symlink -->");
                    if (yaffs_readlink(str,str,100) < 0)
                        printk("no alias");
                    else
                        printk("\"%s\"", str);
                    break;
                default: printk("unknown"); break;
            }

            printk(" \r\n");
            if ((s.st_mode & S_IFMT) == S_IFDIR)
                dump_directory(str);
        }

        yaffs_closedir(d);     // 关闭目录
    }
}
```

```c
/*
 * 测试函数
 */
void yaffs_test(int remainopen)
{
    //lwmem_initialize(0);                  // 初始化内存管理
    //ls1x_nand_init(devNAND, NULL);        // 初始化 NAND 驱动程序
    //yaffs_startup_and_mount("/ndd");      // 初始化 yaffs2 文件系统
    yaffs_dump_dev("/ndd");

    yaffs_mkdir("/ndd/data", S_IFDIR);      // 创建目录
    #if 1
    {
        /* 打开文件时，由 O_CREAT 决定是否在文件不存在时创建 */
        int fd = yaffs_open("/ndd/data/test.dat", O_CREAT | O_RDWR, 0777);
        if (fd >= 0)                        // 成功时，返回值 >= 0
        {
            unsigned int val = 0x12345678, rd = 0;
            yaffs_ftruncate(fd, 0);                     // 设置文件长度为 0
            yaffs_write(fd, (const void *)&val, 4);     // 写文件，返回写入字节数
            yaffs_flush(fd);                            // 把文件从 cache 刷新到 flash 中
            yaffs_read(fd, (const void *)&val, 4);      // 读文件，返回读出字节数
            yaffs_close(fd);                            // 关闭文件
            yaffs_unlink("/ndd/test.dat");              // 删除文件
        }
    }
    #endif

    dump_directory("/ndd");             // 查看/ndd
    yaffs_rmdir("/ndd/script");         // 删除目录
    dump_directory("/ndd");             //查看/ndd

    if (!remainopen)
        yaffs_unmount("/ndd");
}

#endif
```

二、lwIP 1.4.1 网络协议栈

lwIP 是一套用于嵌入式系统的开放源代码 TCP/IP 协议栈，可以在裸机模式、RTOS 模式下运行。loongide 内置移植了对 RTThread、uCOSII、FreeRTOS 的支持。

1、移植说明

移植文件有四个，分别是：

①、lwip-1.4.1/port/include/arch/sys_arch.h

rtthead、uCOSII、FreeRTOS 下对 lwip 实现支持的定义。

②、lwip-1.4.1/port/include/lwipopts.h

lwip 的参数配置，用户可以根据需要进行修改。

③、lwip-1.4.1/port/sys_arch.c

lwip 的互斥量、信号灯、队列等在 rtthead、uCOSII、FreeRTOS 下实现。

④、lwip-1.4.1/port/ls1x_ethernetif.c

实现 lwip 和 LS1B200、LS1C300 的网卡驱动程序的挂接。

mac 地址：

```
static char ls1x_gmac0_mac_addr[6]
static char ls1x_gmac0_mac_addr[6]
```

接口函数

```
/*
 * 功能：裸机编程时读入网卡输入；RTOS 下 lwip 自动调用。
 * 参数：struct netif *p_gmac0_netif 或者 p_gmac1_netif
 */
void ethernetif_input(void *pParams)

/*
 * 功能：初始化 lwip 网络协议栈、实现和网卡驱动程序的挂接
 * 参数：ip0      第一个网卡的 IPv4 地址
 *       ip1      第二个网卡的 IPv4 地址
 */
void ls1x_initialize_lwip(unsigned char *ip0, unsigned char *ip1)
```

2、编程示例

①、配置 bsp.h

打开以下配置项：

```
#define BSP_USE_GMAC0
#define USE_LWIP
```

②、初始化

```
#ifdef USE_LWIP
  #ifdef OS_NONE
     lwip_init();                          // Initilaize the LwIP stack
  #endif
    ls1x_initialize_lwip(NULL, NULL);      // Initilaize the LwIP & GMAC0 glue
#endif
```

③、RTOS 下编程示例

TCP 服务器

```
#include "lwip/sockets.h"
#include "lwip/err.h"
#include "lwip/sys.h"

// 数据收发缓冲区
static char data_buf[TCP_SERVER_BUFSIZE] = "hello, I'm tcp server!\n";

static void tcp_server_thread(void *arg)
{
    struct sockaddr_in local_addr;                    // 本地 socket 地址
    int sock_fd, err;

    sock_fd = socket(AF_INET, SOCK_STREAM, 6);        // 建立 socket 连接，类型 TCP
    if (sock_fd == -1)
    {
        printk("failed to create sock_fd!\n");
        return;
    }

    memset(&local_addr, 0, sizeof(local_addr));
    local_addr.sin_family = AF_INET;
    local_addr.sin_addr.s_addr = inet_addr(local_IP);    // 192.168.1.123:9060
    local_addr.sin_port = htons(TCP_LOCAL_PORT);
    local_addr.sin_len = sizeof(local_addr);

    // 把 192.168.1.123:9060 绑定到新建的 socket
    err = bind(sock_fd, (struct sockaddr *)&local_addr, sizeof(struct sockaddr));
    if (err != ERR_OK)
    {
        closesocket(sock_fd);                         // 失败，关闭 socket 连接
        printk("failed to bind()!\n");
        return;
    }
```

```
err = listen(sock_fd, 3);                                   // 启动监听 socket
if (err != ERR_OK)
{
    closesocket(sock_fd);                                   // 失败，关闭 socket 连接
    printk("failed to listen()!\n");
     return;
}

/*
 * loop first. 循环等待客户接入
 */
while (1)
{
   int client_fd;
   struct sockaddr_in client_addr;                          // 客户 socket 地址
   int addrlen = sizeof(client_addr);

   // 接收客户连接，当有客户连接请求时返回；RTOS 阻塞此函数
   client_fd = accept(sock_fd, (struct sockaddr*)&client_addr, (socklen_t)&addrlen);
   if (client_fd > 0)
   {
       printk("client incoming...\r\n");

       /*
        * loop second. 和连接的客户端保持通信；
        *              多客户连接支持通过创建单独的线程来实现。
        */
       for (;;)
       {
           memset(data_buf, 0, TCP_SERVER_BUFSIZE);

           // 读入客户输入
           err = recv(client_fd, data_buf, TCP_SERVER_BUFSIZE, 0);
           if (err > 0)
           {
               printk("RECV: %s", data_buf);
               send(client_fd, data_buf, err, 0);           // 向客户发送信息
           }
           else if (err == ERR_CLSD)                        /* 客户连接已断开 */
           {
                closesocket(client_fd);                     // 关闭客户连接
                printk("client disconnected.\r\n");
                break;
           }
           else if (err <= 0)
           {
                closesocket(client_fd);                     // 出错时
                printk("disconnect client...\r\n");
                break;
           }
       }
   }

   // delay_ms(200);
}
```

```
    /*
     * NEVER GO HERE!
     */
    closesocket(sock_fd);

    printk("tcp_server_thread stop!\r\n");
}


/*
 * 初始化 TCP 服务器，供用户调用
 */
void tcp_server_init(void)
{
    // ls1x_initialize_lwip(NULL, NULL);
    sys_thread_new("tcp_server",
                   tcp_server_thread,
                   NULL,
                   DEFAULT_THREAD_STACKSIZE,
                   DEFAULT_THREAD_PRIO + 1);
}
```

TCP 客户端

```
#include "lwip/sockets.h"
#include "lwip/err.h"
#include "lwip/sys.h"

// 数据收发缓冲区
static char data_buf[TCP_CLIENT_BUFSIZE] = "hello, you are connected!\n";

static void tcp_client_thread(void *arg)
{
    struct sockaddr_in remote_addr;                         // 远程 socket 地址
    int sock_fd, err, count=0;

    sock_fd = socket(AF_INET, SOCK_STREAM, 0);              // 建立 socket 连接，类型 TCP
    if (sock_fd == -1)
    {
        printk("failed to create sock_fd!\n");
        return;
    }

    memset(&remote_addr, 0, sizeof(remote_addr));
    remote_addr.sin_family = AF_INET;
    remote_addr.sin_addr.s_addr = inet_addr(remote_IP);    // 192.168.1.111:9061
    remote_addr.sin_port = htons(TCP_REMOTE_PORT);

    // 连接指定 IP:PORT 的远程 TCP 服务器
    err = connect(sock_fd, (struct sockaddr *)&remote_addr, sizeof(struct sockaddr));
    if (err != ERR_OK)
    {
        closesocket(sock_fd);                               // 失败，关闭 socket 连接
        printk("failed to connect to server!\n");
        return;
    }
```

```
    // 成功建立连接后进入通信
    while (1)
    {
        int rdbytes = 0;
        unsigned int ticks = get_clock_ticks();

        memset(data_buf, 0, TCP_CLIENT_BUFSIZE);                    // 准备发送数据
        snprintf(data_buf, 99, "client ticks = %i.\n", ticks);

        if (send(sock_fd, data_buf, strlen(data_buf), 0) <= 0)      // 发送数据
        {
            delay_ms(1000);
            continue;
        };

        rdbytes = recv(sock_fd, data_buf, TCP_CLIENT_BUFSIZE, 0);   // 接收回复
        if (rdbytes > 0)
        {
            printk("SERVER REPLAY: %s", data_buf);
        }

        delay_ms(100);
        if (count++ >= 20)        /* 结束 */
           break;
    }

    closesocket(sock_fd);         // 关闭连接
    printk("tcp_client_thread stop!\r\n");
}


/*
 * 初始化 TCP 客户端函数，供用户调用
 */
void tcp_client_init(void)
{
    // ls1x_initialize_lwip(NULL, NULL);
    sys_thread_new("tcp_client",
                   tcp_client_thread,
                   NULL,
                   DEFAULT_THREAD_STACKSIZE,
                   DEFAULT_THREAD_PRIO + 1);
}
```

UDP 服务器

```
#include "lwip/sockets.h"
#include "lwip/err.h"
#include "lwip/sys.h"

// 数据收发缓冲区
static char data_buf[UDP_SERVER_BUFSIZE];

static void udp_server_thread(void *arg)
{
    struct sockaddr_in local_addr;                      // 本地 socket 地址
    int sock_fd;
    int err;

    sock_fd = socket(AF_INET, SOCK_DGRAM, 0);           // 建立 socket 连接，类型 UDP
    if (sock_fd == -1)
    {
        printk("failed to create sock_fd!\n");
        return;
    }

    memset(&local_addr, 0, sizeof(local_addr));
    local_addr.sin_family = AF_INET;
    local_addr.sin_addr.s_addr = inet_addr(local_IP);  // 192.168.1.123:9062
    local_addr.sin_port = htons(UDP_LOCAL_PORT);

    // 把 192.168.1.123:9062 绑定到新建的 socket
    err = bind(sock_fd, (struct sockaddr *)&local_addr, sizeof(local_addr));
    if (err == -1)
    {
        closesocket(sock_fd);                           // 失败，关闭 socket
        printk("udp server bind() error.\r\n");
        return;
    }

    while (1)
    {
       struct sockaddr from;                            // 客户端 IP:PORT
       socklen_t fromlen;

       memset(data_buf, 0, UDP_SERVER_BUFSIZE);
       // 接收来自 UDP 客户端的连接，返回接收到的字节数
       err = recvfrom(sock_fd, data_buf, UDP_SERVER_BUFSIZE, 0, &from, &fromlen);
       if (err <= 0)
       {
           delay_ms(100);
           continue;
       }

        printk("RECV: %s\r\n", data_buf);

        /* 给 UDP 客户端 IP:PORT 回复数据 */
        sendto(sock_fd, data_buf, err, 0, &from, fromlen); //

        // delay_ms(100);
```

```
    }

    /* NEVER GO HERE! */
    closesocket(sock_fd);

    printk("udp_server_thread stop!\r\n");
}


/*
 * 初始化 UDP 服务器，供用户调用
 */
void udp_server_init(void)
{
    // ls1x_initialize_lwip(NULL, NULL);
    sys_thread_new("udp_server_thread",
                   udp_server_thread,
                   NULL,
                   DEFAULT_THREAD_STACKSIZE,
                   DEFAULT_THREAD_PRIO + 1);
}
```

UDP 客户端

```
#include "lwip/sockets.h"
#include "lwip/err.h"
#include "lwip/sys.h"

// 数据收发缓冲区
static char data_buf[UDP_CLIENT_BUFSIZE] = "this is a UDP test package";

static void udp_client_thread(void *arg)
{
    struct sockaddr_in remote_addr;                         // 远程服务器 socket 地址
    int sock_fd;
    int err;
    int count = 0;

    sock_fd = socket(AF_INET, SOCK_DGRAM, 0);               // 建立 socket 连接，类型 UDP
    if (sock_fd == -1)
    {
        printk("failed to create sock_fd!\n");
        return;
    }

    memset(&remote_addr, 0, sizeof(remote_addr));
    remote_addr.sin_family = AF_INET;
    remote_addr.sin_addr.s_addr = inet_addr(remote_IP);     // 192.168.1.111:9063
    remote_addr.sin_port = htons(UDP_REMOTE_PORT);

    while (1)
    {
        struct sockaddr from;
        socklen_t fromlen;
        unsigned int ticks = get_clock_ticks();
```

```
        memset(data_buf, 0, UDP_CLIENT_BUFSIZE);
        snprintf(data_buf, UDP_CLIENT_BUFSIZE-1, "client ticks = %i.", ticks);

        // 把数据发送到 UDP 服务器
        err = sendto(sock_fd,
                     data_buf,
                     strlen(data_buf),
                     0,
                     (struct sockaddr *)&remote_addr,
                     sizeof(remote_addr));

        if (err <= 0)
        {
           delay_ms(1000);
           continue;
        }

        delay_ms(100);

        // 接收 UDP 服务器的回复
        err = recvfrom(sock_fd,
                       data_buf,
                       UDP_CLIENT_BUFSIZE,
                       MSG_DONTWAIT,
                       &from,
                       &fromlen);
        if (err > 0)
        {
            printk("SERVER REPLAY: %s\r\n", data_buf);
        }

        if (count++ >= 20)       /* 结束 */
           break;

    }

    closesocket(sock_fd);

    printk("udp_client_thread stop!\r\n");
}


/*
 * 初始化 UDP 客户端，供用户调用
 */
void udp_client_init(void)
{
    // ls1x_initialize_lwip(NULL, NULL);
    sys_thread_new("udp_client_thread",
                   udp_client_thread,
                   NULL,
                   DEFAULT_THREAD_STACKSIZE,
                   DEFAULT_THREAD_PRIO + 1);
}
```

三、FTP 服务器

FTP 服务器通过 lwIP 管理 NAND flash 的 yaffs2 文件系统，可以在 NAND flash 上创建、重命名、删除 yaffs2 的文件夹和文件，比如管理图片、字库等数据文件。
FTP 服务器在 RTOS 下运行。

1、文件与函数

文件有三个，分别是：
①、ftpd/ftpd.h
　　ftpd 实现相关的定义。
②、ftpd/ftpd.c
　　ftpd 的 lwip 实现，和 yaffs2 文件系统的挂接。
③、ftpd/start_ftp_server.c
　　初始化并启动 ftp 服务器。

ftp 初始化

```
// 用户自定义钩子函数类型，该钩子函数用于对特定文件进行用户自定义处理
typedef int (*ftpd_hook_func)(char *, size_t);

// 用户登录时需要用户名和密码验证
typedef int (*ftpd_login_func)(const char *user, char *password);

// ftp 服务器配置参数
struct ftpd_configuration
{
    int               priority;           /* ftp 线程优先级  */
    unsigned long     max_hook_filesize;  /* 钩子函数可以处理的最大文件缓存字节数 */
    int               port;               /* ftp 端口号，默认 21 */
    struct ftpd_hook *hooks;              /* 钩子函数数组 */
    char const       *root;               /* ftp 服务器根目录，  0=/ */
    int               tasks_count;        /* 同时连接的客户数 */
    int               idle;               /* 接收等待时间，单位秒，0= 一直等待 */
    int               access;             /* 0=读写，1=只读，2=只写，3=浏览 */
    ftpd_login_func   login;              /* 用户登录验证函数 */
};
```

初始化函数

```
/*
 * 功能：初始化并启动 ftp 服务器
 * 返回：0            成功
 */
int lwip_initialize_ftpd(void)

/*
 * 功能：初始化并启动 ftp 服务器，包含了 yaffs2、lwip 的初始化
 * 返回：0            成功
 */
int start_ftpd_server(void)
```

2、编程示例

①、配置 bsp.h

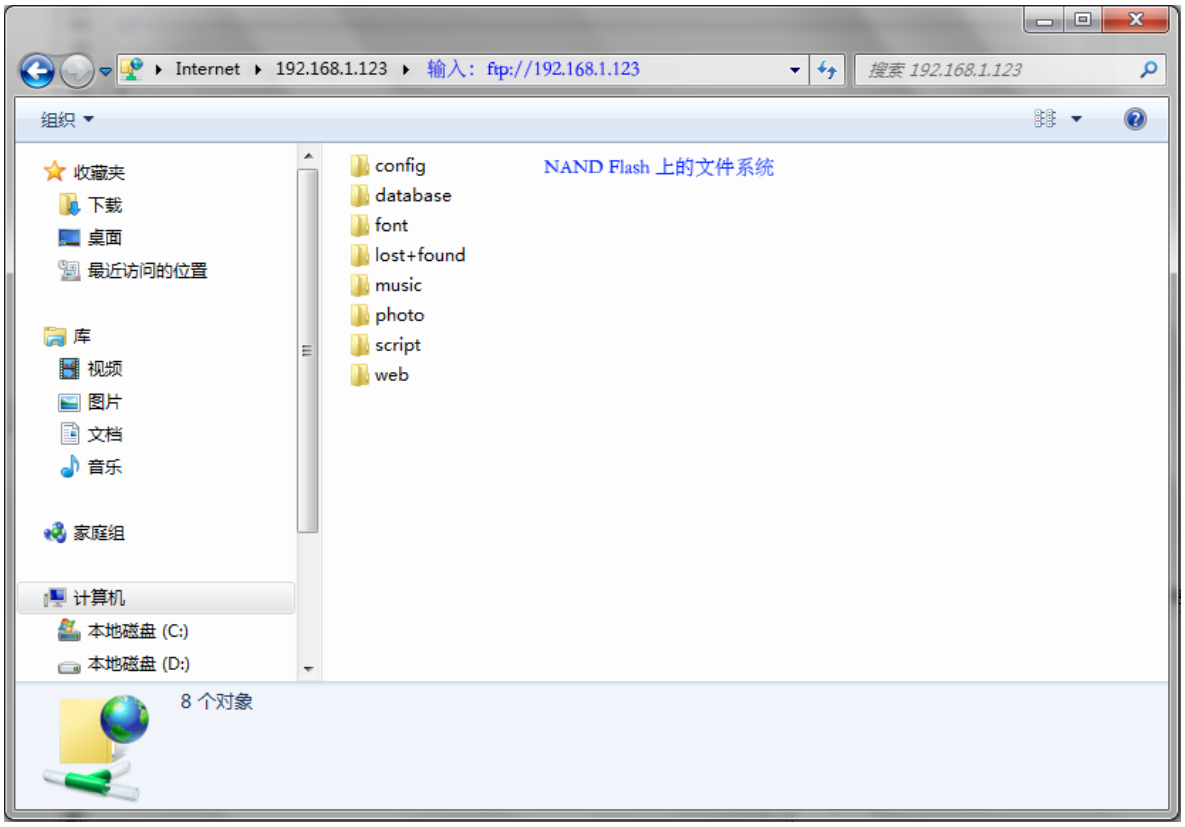
打开以下配置项：

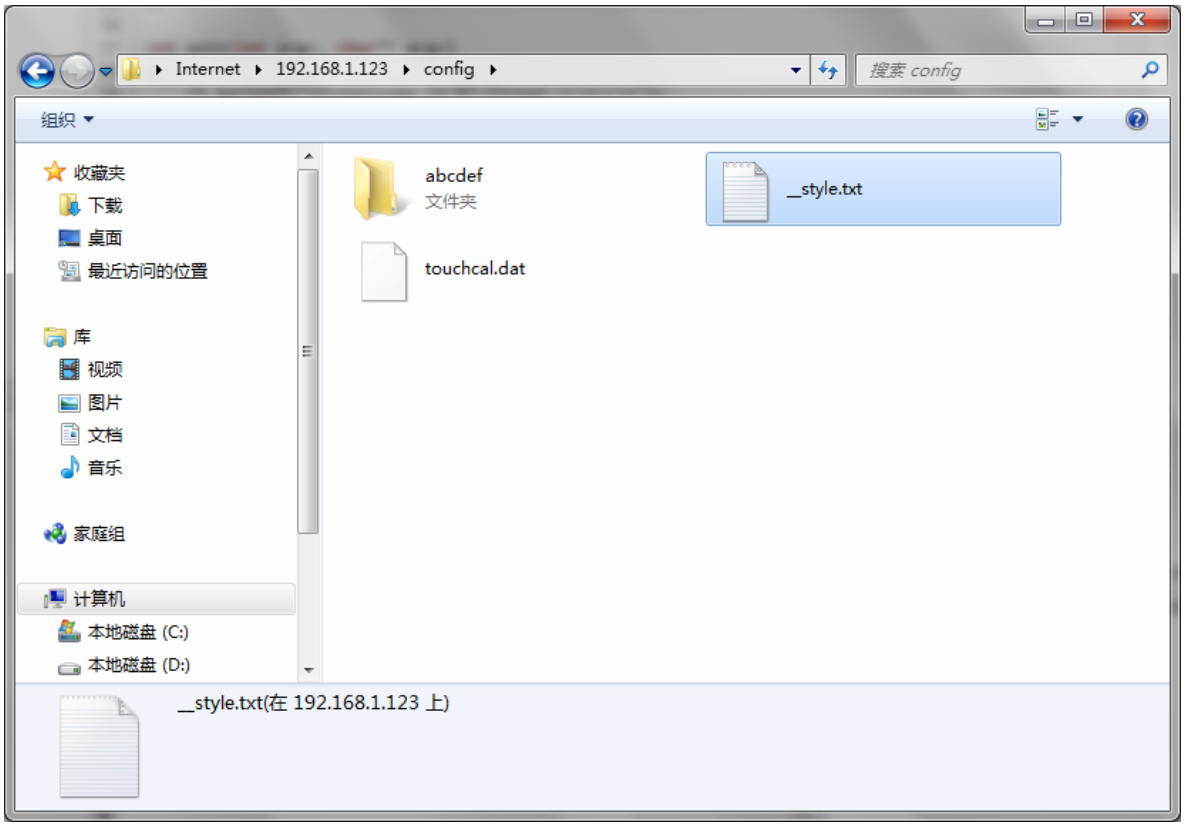
```
#define BSP_USE_NAND
#define USE_YAFFS2
#define USE_LWIP
#define USE_FTPD
```

②、初始化

```
#ifdef BSP_USE_NAND
    ls1x_nand_init(devNAND, NULL);
#endif
#ifdef USE_YAFFS2
    yaffs_startup_and_mount(RYFS_MOUNTED_FS_NAME);
#endif
#if BSP_USE_OS
 #ifdef USE_FTPD
   start_ftpd_server();
 #endif
#endif
```

③、使用 Windows Explorer 管理 FTP 站点





BUG： ***因为 explorer 的原因，不能删除目录。***

四、Simple-GUI 组件

Simple-GUI 是为展示龙芯 1x 的 FrameBuffer 而编写的一个简易软件组件，实现了 TGrid 和 TButton 两个元件。

TGrid 元件模拟 Windows 下的 Grid 样式，以表格形式显示用户数据。

TButton 元件模拟通用 Button 按钮，通过回调函数响应按钮事件。

用户创建 TGrid 和 TButton 实例时使用**组号**实现分组显示管理，通过激活当前活动组号，实现在 LCD 上显示不同的内容。

1、文件与函数

实现文件有四个，分别是：

①、gui/simple-gui/simple_gui.h

TGrid 和 TButton 实现相关的定义。

②、gui/simple-gui/simple_gui.c

TGrid 和 TButton 的具体实现，和显示控制线程等。

③、gui/demo_gui.h

demo 程序接口。

④、gui/demo_gui.c

demo 程序的实现，在屏幕上画一个 TGrid 个四个 TButton。

simple-gui 主要接口函数

```
/*
 * 功能：新建一个按钮
 * 参数：rt       待建按钮的矩形区域，
 *       guid     待建按钮的全局唯一编号
 *       group    待建按钮的所属分组编号
 *       caption  按钮标签文字
 *       click    按钮事件响应回调函数
 * 返回：TButton* 新建按钮，NULL：失败
 */
TButton *new_button(TRect *rt, int guid, int group, const char *caption, OnClick click);
```

```
/*
 * 功能：新建一个网格
 * 参数：rt        待建网格的矩形区域，
 *       rows      网格的总行数(不包含 title 行)
 *       cols      网格的总列数
 *       guid      待建网格的全局唯一编号
 *       group     待建网格的所属分组编号
 * 返回：TGrid*    新建网格，NULL：失败
 */
TGrid *create_grid(TRect *rt, int rows, int cols, int guid, int group);

/*
 * 功能：启动 simple-gui 任务。RTOS 下使用
 * 返回：无
 */
int start_gui_monitor_task(void);

/*
 * 功能：在 framebuffer 上画 simple-gui。裸机编程下使用
 * 返回：无
 */
void paint_my_simple_gui(void);

/*
 * 功能：根据获得的触摸屏焦点，设置获得焦点的按钮并响应事件。裸机编程下使用
 * 参数：x     触摸屏 x 坐标
 *       y     触摸屏 y 坐标
 * 返回：无
 */
void set_focused_button(int x, int y);

/*
 * 功能：获取网格的某一列
 * 参数：grid      待操作网格
 *       index     索引号
 * 返回：Tcolumn* NULL=未找到
 */
TColumn *grid_get_column(TGrid *grid, int index);
```

```c
/*
 * 功能：设置网格的列宽
 * 参数：grid      待操作网格
 *       index     索引号
 *       width     待设置的列宽度
 * 返回：无
 */
void grid_set_column_width(TGrid *grid, int index, int width);

/*
 * 功能：设置网格的列标题
 * 参数：column    网格的列，使用 grid_get_column()获取
 *       title     待设置的标题
 * 返回：无
 */
void grid_set_column_title(TColumn *column, const char *title);
```

demo 接口函数

```c
/*
 * 功能：初始化 demo
 * 返回：无
 */
void start_my_gui(void);

/*
 * 功能：在 demo 创建的 TGrid [row,col] 位置显示指定字符串。
 * 参数：row       网格的行坐标，从 0 开始编号
 *       col       网格的纵坐标，从 0 开始编号
 *       str       待显示的字符串
 * 返回：无
 */
void gui_drawtext_in_grid(int row, int col, const char *str);
```

2、编程示例

①、配置 bsp.h

打开以下配置项：

```
#define BSP_USE_SPI0                // 7 寸横屏 LCD
//#define BSP_USE_I2C0              // 4.3 寸竖屏 LCD
#define XPT2046_DRV                 // SPI0-CS1 上挂接的 XPT2046 触摸屏芯片
//#define GT1151_DRV                // I2C0 上挂接的 GT1151 触摸屏芯片
#define BSP_USE_FB
```

②、初始化

```
#ifdef BSP_USE_SPI0
    ls1x_spi0_initialie(busSPI0, NULL);
#endif
#ifdef XPT2046_DRV
    ls1x_xpt2046_init(busSPI0, NULL);
#endif
#if BSP_USE_FB
    fb_open();
#endif
```

③、编程示例

请参阅：gui/demo_gui.c

五、Modbus 协议栈

本 modbus 协议栈根据某开源协议栈改写和移植，实现了 RTU/ASCII 协议的通信，可用于裸机、RTThread、uCOSII、FreeRTOS 下创建主站和从站。

1、移植说明

modbus 协议栈文件较多，主要介绍用户编程需要处理的内容。

①、modbus/app/app_cfg.h

```
#define MODBUS_CFG_SLAVE_EN       0       /* 是否使用 Modbus 从站 */
#define MODBUS_CFG_MASTER_EN      1       /* 是否使用 Modbus 主站 */
#define MODBUS_CFG_ASCII_EN       1       /* 使能 Modbus ASCII 协议 */
#define MODBUS_CFG_RTU_EN         0       /* 使能 Modbus RTU 协议 */
```

用户根据待实现的 modbus 协议栈配置以上宏定义为 0 或者 1(使能)。

```
#define MODBUS_CFG_CHNL_MAX       5       /* 预分配 Modbus 通道总数 */
```

②、modbus/app/mb_data.c

当用户需要实现 modbus 从站时(`#define MODBUS_CFG_SLAVE_EN 1`)，**用户在此编写代码实现具体协议，以响应 modbus 主站的读取请求**。

接口函数

```
/*
 * 初始化 modbus 系统。
 * 参数 freq 用于 RTU 通信时的结束定时器，一般取值 100HZ (10ms)。
 */
void modbus_init(uint32_t freq);
/*
 * 用以配置 modbus 节点，从 mb_devices_tbl 设备表中返回一个未使用的设备。
 * 返回: MODBUS_t *, NULL=失败
 */
MODBUS_t *modbus_config_node(uint8_t  node_addr,       // 节点地址，1~255，0：广播地址
                             uint8_t  master_slave,    // 站点作为主机或者从机
                             uint32_t rx_timeout,      // 用作主机时查询从机超时等待毫秒数
                             uint8_t  modbus_mode,     // ASCII 或者 RTU 通信协议
                             void    *uart_device,     // 龙芯 1x 的 UART 设备: NS16550_t *
                             uint32_t baud,            // 通信波特率，2400~115200
                             uint8_t  bits,            // 通信数据位数，常用值 8
                             uint8_t  parity,          // 通信校验位，'N':无校验; 'E':偶校验
                             uint8_t  stops,           // 通信停止位，1~2
                             uint8_t  wr_en);          // 站点作为从机时，是否允许主机写入
```

2、编程示例

①、配置 bsp.h

打开以下配置项：

```
#define BSP_USE_UART4              // 用于 modbus 通信的 485 或者 232 设备
#define BSP_USE_RTC                // 用于 modbus RTU 通信方式的定时器
#define USE_MODBUS                 // 使用 modbus
```

②、初始化

```
#ifdef BSP_USE_UART4
    ls1x_uart_init(devUART4, NULL);
#endif
#ifdef BSP_USE_RTC
    ls1x_rtc_init(NULL, NULL);
#endif
#if USE_MODBUS
    modbus_init(100);
#endif
```

③、编程示例

modbus 主站

```
#include "bsp.h"
#include "modbus/app/mb_cfg.h"

// 本代码工作于 RTOS 下
#if BSP_USE_OS && MODBUS_CFG_MASTER_EN

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MBSVR_STACK_SIZE    2*1024
#define MBSVR_PRIO          8

#if defined(OS_RTTHREAD)
#include "rtthread.h"
static rt_thread_t  mb_svr_thread;
#elif defined(OS_UCOS)
#include "os.h"
static OS_STK       mb_svr_stack[MBSVR_STACK_SIZE];
#elif defined(OS_FREERTOS)
#include "FreeRTOS.h"
#include "task.h"
static TaskHandle_t mb_svr_handle = NULL;
#endif
```

```c
#ifdef BSP_USE_FB
#include "ls1x_fb.h"
#endif

#include "ns16550.h"
#include "modbus/src/mb.h"

//-----------------------------------------------------------------------------------------
--------

static MODBUS_t *ptrMaster;

#define MB_NODEADDR     0

/*
 * 配置主站参数
 */
void mb_cfg_master(void)
{
    ptrMaster = modbus_config_node(MB_NODEADDR,         // Node Address
                                   MODBUS_MASTER,       // master
                                   100,                 // rx timeout = ticks?
                                   MODBUS_MODE_ASCII,   // MODBUS_MODE_RTU
                                   devUART4,            // Modbus device of LS1B
                                   9600,                // baudrate
                                   8,                   // bits
                                   'N',                 // parity: 'N'/'E'/''
                                   1,                   // stops
                                   MODBUS_WR_EN );      // wr_en

}


/*
 * 主站任务，在 RTOS 下运行
 */
static void mb_svr_task(void *arg)
{
    mb_cfg_master();          // 配置主站

    delay_ms(10);

    for ( ; ; )
    {
        uint8_t err;
        uint16_t regs[6];

        // 读从站 15 的数据
        modbus_set_address(ptrMaster, 15);

        // 读从站 15 的线圈保持寄存器
        err = modbus_master_fc03_read_holding_register(
                    ptrMaster,          // master, uart4
                    15,                 // slave_node,
                    0,                  // slave_addr,
                    regs,               // p_reg_tbl,
                    2                   // nbr_regs
```

```
                    );

        if (err == 0)
            printk("fc03 read(Addr=15): holding[0]=0x%04x, holding[1]=0x%04x\r\n",
                    (int)regs[0], (int)regs[1]);
        else
            printk("fc03 read(Addr=15): err = %i\r\n", (int)err);


        // 读从站 18 的数据
        modbus_set_address(ptrMaster, 18);

        // 读从站 18 的输入寄存器
        err = modbus_master_fc04_read_in_register(
                    ptrMaster,          // master, uart4
                    18,                 // slave_node,
                    0,                  // slave_addr,
                    regs,               // p_reg_tbl,
                    2                   // nbr_regs
                    );
        if (err == 0)
            printk("fc04 read(Addr=18): in[0]=0x%04x, in[1]=0x%04x\r\n", (int)regs[0],
                   (int)regs[1]);
        else
            printk("fc04 read(Addr=18): err = %i\r\n", (int)err);

        delay_ms(100);  // 100 可用.
    }
}


/*
 * 创建 modbus 主站线程
 */
int start_mb_master_task(void)
{
#if defined(OS_RTTHREAD)

    mb_svr_thread = rt_thread_create("MBSvrthread",
                                     mb_svr_task,
                                     NULL,                  // arg
                                     MBSVR_STACK_SIZE*4,    // statck size
                                     MBSVR_PRIO,            // priority
                                     10);                   // slice ticks

     if (mb_svr_thread == NULL)
     {
         printk("create modbus master thread fail!\r\n");
         return -1;
     }

    rt_thread_startup(mb_svr_thread);

#elif defined(OS_UCOS)

    unsigned char err;

    err = OSTaskCreate(mb_svr_task,
```

```
                        NULL,
        #if OS_STK_GROWTH == 1
                     (void *)&mb_svr_stack[MBSVR_STACK_SIZE - 1],
        #else
                     (void *)&mb_svr_stack[0],
        #endif
                      MBSVR_PRIO);

     if (err != 0)
     {
         printk("create modbus master task fail!\r\n");
         return -1;
     }

#elif defined(OS_FREERTOS)

    xTaskCreate(mb_svr_task,
                "MBSvrtask",
                MBSVR_STACK_SIZE,
                NULL,
                configMAX_PRIORITIES -  MBSVR_PRIO,
                &mb_svr_handle);

     if (mb_svr_handle == NULL)
     {
         printk("create modbus master task fail!\r\n");
         return -1;
     }

#endif

    printk("create modbus master task successful.\r\n");

    return 0;
}

#endif // #if MODBUS_CFG_MASTER_EN
```

modbus 从站

```
#include "bsp.h"
#include "modbus/app/mb_cfg.h"

#if BSP_USE_OS && MODBUS_CFG_SLAVE_EN

#include "ns16550.h"
#include "modbus/src/mb.h"

static MODBUS_t *ptrSlave;

#define MB_NODEADDR     15

/*
 * 配置从站参数，modbus 自动创建线程。
 */
void mb_cfg_slave(void)
{
    ptrSlave = modbus_config_node(MB_NODEADDR,          // Node Address
                                  MODBUS_SLAVE,         // slave
                                  0,                    // rx timeout = 0 when slave
                                  MODBUS_MODE_ASCII,    // MODBUS_MODE_RTU,
                                  devUART4,             // Modbus device of LS1B
                                  9600,                 // baudrate
                                  8,                    // bits
                                  'N',                  // parity: 'N'/'E'/''
                                  1,                    // stops
                                  MODBUS_WR_EN );       // wr_en

}

#endif // #if MODBUS_CFG_SLAVE_EN
```

modbus 裸机实现

配置主站或从站的方法同 RTOS 下一致。

在裸机的主循环中调用以下方法：

```
#if MODBUS_CFG_SLAVE_EN
    modbus_slave_poll();          // 从机查询，定义在 mb_os.c 中
#endif

#if MODBUS_CFG_MASTER_EN
    modbus_master_poll();         // 主机查询，自定义：使用上面 mb_svr_task()循环里的语句
#endif
```

六、LVGL 图形界面库

lvgl 是一款画面美观、内存占用率低的免费开源图形库，适合嵌入式 GUI 的开发和使用。本 lvgl 库移植的是 7.0.1 版本。

1、移植说明

①、lvgl/lv_conf.h

主要修改内容如下：

```
……
#ifdef GT1151_DRV             // 480*800 的竖屏触摸屏
#define LV_HOR_RES_MAX            (480)
#define LV_VER_RES_MAX            (800)
#elif defined(XPT2046_DRV)    // 800*480 的横屏触摸屏
#define LV_HOR_RES_MAX            (800)
#define LV_VER_RES_MAX            (480)
#else
#error "please define lcd pixels here. "
#endif
……
#define LV_COLOR_DEPTH            16      // 龙芯 1x 的 framebuffer 驱动使用 RGB565
……
#define LV_TICK_CUSTOM            1       // 自定义 tick
#if LV_TICK_CUSTOM == 1
#define LV_TICK_CUSTOM_INCLUDE    "bsp.h"                   /* tick 函数头文件 */
#define LV_TICK_CUSTOM_SYS_TIME_EXPR (get_clock_ticks())    /* 获取 tick 的函数 */
#endif   /*LV_TICK_CUSTOM*/
……
/*
 * 自定义字库微软雅黑 16/24
 */
#define HZ_MSYH_16                0       /* 一级二级汉字 */
#define HZ_MSYH_3500_16           0       /* 3500 个常用字 */
#define HZ_MSYH_24                0       /* 一级二级汉字 */
#define HZ_MSYH_3500_24           0       /* 3500 个常用字 */

/*
 * 自定义字库新宋体 16/24
 */
#define HZ_SIMSUN_16              0       /* 一级二级汉字 */
#define HZ_SIMSUN_3500_16         1       /* 3500 个常用字 */
#define HZ_SIMSUN_24              0       /* 一级二级汉字 */
#define HZ_SIMSUN_3500_24         0       /* 3500 个常用字 */

#define LV_FONT_CUSTOM_DECLARE    LV_FONT_DECLARE(hz_msyh_16)       \
                                  LV_FONT_DECLARE(hz_msyh_3500_16)  \
                                  LV_FONT_DECLARE(hz_msyh_24)       \
                                  LV_FONT_DECLARE(hz_msyh_3500_24)  \
                                  LV_FONT_DECLARE(hz_simsun_16)     \
                                  LV_FONT_DECLARE(hz_simsun_3500_16) \
                                  LV_FONT_DECLARE(hz_simsun_24)     \
                                  LV_FONT_DECLARE(hz_simsun_3500_24)
```

```
……
/*
 * 修改默认字库为自定义汉字
 */
#if HZ_SIMSUN_3500_16
#define LV_THEME_DEFAULT_FONT_SMALL         &hz_simsun_3500_16
#define LV_THEME_DEFAULT_FONT_SUBTITLE      &hz_simsun_3500_16
#define LV_THEME_DEFAULT_FONT_TITLE         &hz_simsun_3500_16
#elif HZ_MSYH_3500_16
#define LV_THEME_DEFAULT_FONT_SMALL         &hz_msyh_3500_16
#define LV_THEME_DEFAULT_FONT_SUBTITLE      &hz_msyh_3500_16
#define LV_THEME_DEFAULT_FONT_TITLE         &hz_msyh_3500_16
#elif HZ_SIMSUN_16
#define LV_THEME_DEFAULT_FONT_SMALL         &hz_simsun_16
#define LV_THEME_DEFAULT_FONT_SUBTITLE      &hz_simsun_16
#define LV_THEME_DEFAULT_FONT_TITLE         &hz_simsun_16
#elif HZ_MSYH_16
#define LV_THEME_DEFAULT_FONT_SMALL         &hz_msyh_16
#define LV_THEME_DEFAULT_FONT_SUBTITLE      &hz_msyh_16
#define LV_THEME_DEFAULT_FONT_TITLE         &hz_msyh_16
#else
#define LV_THEME_DEFAULT_FONT_SMALL         &lv_font_montserrat_16
#define LV_THEME_DEFAULT_FONT_SUBTITLE      &lv_font_montserrat_16
#define LV_THEME_DEFAULT_FONT_TITLE         &lv_font_montserrat_16
#endif

#if HZ_SIMSUN_3500_16
#define LV_THEME_DEFAULT_FONT_NORMAL        &hz_simsun_3500_16
#elif HZ_MSYH_3500_16
#define LV_THEME_DEFAULT_FONT_NORMAL        &hz_msyh_3500_16
#elif HZ_SIMSUN_16
#define LV_THEME_DEFAULT_FONT_NORMAL        &hz_simsun_16
#elif HZ_MSYH_16
#define LV_THEME_DEFAULT_FONT_NORMAL        &hz_msyh_16
#else
#define LV_THEME_DEFAULT_FONT_NORMAL        &lv_font_montserrat_16
#endif
```

②、lvgl/porting/lv_port_disp.c

lvgl 适配龙芯 framebuffer 驱动的移植。

③、lvgl/porting/lv_port_indev.c

lvgl 适配 XPT2046 触摸屏和 GT1151 触摸屏驱动的移植。

④、lvgl/porting/lv_port_fs.c

lvgl 适配 NAND Flash 的 yaffs2 文件的移植。

接口函数

```
/*
 * 初始化 lvgl 的函数
 */
void lv_init(void)
void lv_port_disp_init(void)
void lv_port_indev_init(void)
void lv_port_fs_init(void)

/*
 * lvgl 周期性执行的函数，在 RTOS 的线程中调用、或者在裸机编程主循环中调用
 */
uint32_t lv_task_handler(void)
```

2、编程示例

①、配置 bsp.h

打开以下配置项：

```
#define BSP_USE_FB
#define USE_LVGL
```

②、初始化

```
#ifdef BSP_USE_FB
    fb_open();
#endif
#ifdef USE_LVGL
    lv_init();
    lv_port_disp_init();
    lv_port_indev_init();
    lv_port_fs_init();
#endif
```

③、编程示例

```
#include "bsp.h"

#ifdef USE_LVGL

#include <stdio.h>
#include <stdlib.h>

#include "lvgl/lvgl.h"

//-----------------------------------------------------------------------------------

/*
 * 按钮回调函数，创建一个 messagebox
 */
static void btn1_clickevent_handler(lv_obj_t *obj, lv_event_t event)
{
    if (event == LV_EVENT_CLICKED) // 如果按钮按下
    {
        // 建立一个消息框
        lv_obj_t *mbox = lv_msgbox_create(lv_scr_act(), NULL);   // 创建消息框
        lv_obj_set_width(mbox, 240);                             // 设置宽度

        static const char *mbox_btns[] = {"\xE7\xA1\xAE"/*确*/
                                          "\xE5\xAE\x9A"/*定*/,
                                          "\xE5\x8F\x96"/*取*/
                                          "\xE6\xB6\x88"/*消*/,
                                          ""};                   // 设置按钮名字

        lv_msgbox_add_btns(mbox, mbox_btns); /* 给消息框添加一个按钮,默然按钮事件会关闭消息框*/
        lv_msgbox_set_text(mbox, "\xE9\xBE\x99"/*龙*/
                                 "\xE8\x8A\xAF"/*芯*/
                                 "\xE5\xB5\x8C"/*嵌*/
                                 "\xE5\x85\xA5"/*入*/
                                 "\xE5\xBC\x8F"/*式*/
                                 "..." );

        lv_obj_align(mbox, lv_scr_act(), LV_ALIGN_IN_TOP_MID, 0, LV_DPI / 2);   // 按钮居中
    }
}


/*
 * 创建一个按钮
 */
static void create_my_button()
{
    lv_obj_t *btn1, *label;

    btn1 = lv_btn_create(lv_scr_act(), NULL);
    lv_obj_set_event_cb(btn1, btn1_clickevent_handler); /* 按钮按下回调函数，可以不用设置，
                                                         * 可以多按钮同一个回调函数 */
    lv_obj_align(btn1, NULL, LV_ALIGN_CENTER, 0, -40);
    lv_obj_set_height(btn1, 40);
```

```
    label = lv_label_create(btn1, NULL);

    lv_label_set_text(label, "\xE9\xBE\x99"/*龙*/
                             "\xE8\x8A\xAF"/*芯*/
                             );
}

//-----------------------------------------------------------------------------------

extern void lv_port_disp_init(void);       // in "lv_port_disp.c"
extern void lv_port_indev_init(void);      // in "lv_port_indev.c"
extern void lv_port_fs_init(void);         // in "lv_port_fs.c"

//-----------------------------------------------------------------------------------

int lvgl_test(void)
{
    printf("Call lv_init...\n");

    lv_init();
    lv_port_disp_init();        // 显示接口
    lv_port_indev_init();       // 输入接口
    lv_port_fs_init();          // 文件接口

    create_my_button();

    printf("littlevgl test success!\n");

    return 0;
}

#endif
```

3、汉字字库和 UFT8 编码

汉字字库

借助工具软件 LvglFontTool 生成 7.0 的字库，然后拷贝到 lvgl/src/lv_font 目录下。

修改配置文件 lvgl/lv_conf.h 文件，将自定义字库加入。

汉字 UTF8 编码

loongide 使用 ANSI 进行字符编码，在代码编辑时通过快捷键“**ctrl+alt**”将汉字字符转换为 UTF8 的二进制编码。

例如：

选中编辑器的字符串："龙芯" → 按“**ctrl+alt**” →
转换为："\xE9\xBE\x99"/*龙*/"\xE8\x8A\xAF"/*芯*/

生成的 UTF8 编码可以让 lvgl 正确识别并显示汉字。