HAL SDK 文档(初版)

- **版本**: 1.1.0
- **生成日期**: 2026-07-08
- **说明**: 本文档涵盖所有硬件抽象层模块的API接口、数据结构和常量定义。

目录

- 1. 引言
- 2. UI开发
- 3. WiFi HAL (hal_wifi.h)
- 4. 无线射频 HAL (hal_wireless.h)
- 5. MCU状态 (mcu_status.h)
- 6. 通用HAL (hal_common.h)
- 7. GPIO HAL (hal_gpio.h)
- 8. PWM背光 (hal_pwm.h)
- 9. 遥控器调参 (hal_rc_tuner.h)
- 10. 云台控制 (hal_top.h)
- 11. 视频播放控制接口 (hal_videoplayer.h)
- 12. 升级功能
- 13. 调试方式
- 14. 旧砖方式

---

1. 引言

1.1 服务器环境搭建

- 概述
  - 本 SDK 开发环境是在 Ubuntu 系统上开发测试。我们推荐使用 Ubuntu 18.04 的系统进行编译

1.2 交叉编译链

- 编译链下载

```
    通过网盘分享的文件：gcc-arm-8.2-2018.08-x86_64-arm-linux-gnueabihf.tar.gz
    链接: https://pan.baidu.com/s/1jUTzLkzLU4vplmRgemzRhw?pwd=6vag 提取码:
6vag
    --来自百度网盘超级会员v6的分享
```

- 概述
  - 需要将编译链压缩包放在当前目录
  - 交叉编译链以压缩包的形式提供，make编译的时候会自动解压到当前目录

- 路径
  - gcc-arm-8.2-2018.08-x86_64-arm-linux-gnueabihf.tar.gz

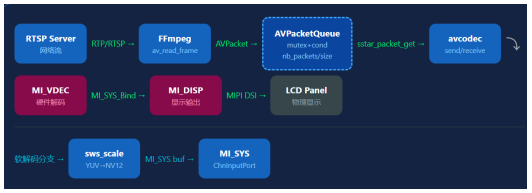
1.3 SDK目录结构

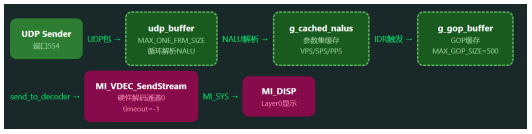
```
.
├── app                     # ---------------- LVGL 源码
│   ├── lv_drivers          # ---------------- lvgl 芯片平台代码
│   ├── lvgl                # ---------------- lvgl 源码
│   ├── objs                # ---------------- 编译过程生成 .o 文件
│   ├── output              # ---------------- 生成 app 路径
│   ├── picture             # ---------------- OSD 图标
│   └── yunzhuo             # ---------------- UI 源码
├── firmware                # ---------------- 生成固件
├── VideoPlayer             # ---------------- 视频播放器
├── include                 # ---------------- 依赖第三方头文件
│   ├── drivers
│   ├── hal                 # ---------------- hal层头文件
│   └── isp
├── package                 # ---------------- ota包
│   ├── bin
│   ├── customer
│   ├── etc
│   ├── lib                 # ---------------- 第三方库
│   └── usr
```

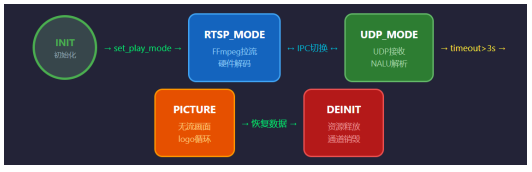
1.4 系统

- 系统框图
  - 

- app
  - 功能模块
    - ui_video：视频主界面，包括视频显示、切换、OSD、云台控制、WIFI、亮度设置
    - ui_menu：设置界面，包括视频设置、遥控设置、射频设置、WIFI设置、系统升级
    - ui_factory：工厂模式界面，包括触摸、显示、老化、背光、WIFI测试
- VideoPlayer
  - 主要功能

        - RTSP解码
        - UDP解码
        - 屏幕息屏控制
        - 启动界面显示
    - 功能模块
        - SsPlayeEs：显示模块、解码器模块、显示模块、RTSP/UDP/无码流检测模块/UDS通讯模块初始化
        - sstaradec\sstardisp\sstarvdec：VDEC通道管理、DISP显示配置、MI_SYS绑定
        - sstarvdec_picture\sstarvdec_rtsp\sstarvdec_udp：RTSP拉流/UDP接收/图片显示
        - video_ipc\video_monitor：Socket通信协议、200ms轮询调度、超时检测、模式切换
    - 数据流向
        - RTSP
        - 

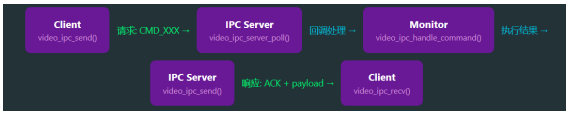
        - UDP
        - 

        - 播放模式状态机
        - 

- IPC模块
    - 概述
        - 采用UDC的短连接的通讯方式，用于 app 和 VideoPlayer 进行进程间通讯
    - 命令类型枚举

```
/* 命令类型 */
typedef enum {
    CMD_SET_PLAYMODE = 1,       /* 切换播放模式 */
    CMD_SET_DISP_STATUS,        /* 亮屏/息屏 (0=off, 1=on) */
```

```
    CMD_REQUEST_RTSP_SWITCH,    /* RTSP 码流切换 */
    CMD_GET_STATUS,             /* 查询当前状态 */
    CMD_ACK,                    /* 响应确认 */
    CMD_HEARTBEAT,              /* 心跳保活 */
} VideoCmdType_e;
```

- 通讯方式
  - 

1.5 app 编译

- 编译指令
  - make clean;make app

1.6 OTA 包编译

- 编译指令
  - make update_picture/image
- 生成固件路径
  - firmware

1.7 启动后台应用程序

- 存放文件
  - 将自定义后台进程和第三方库存放在 package/customer 中

```
customer/bin：存放需要启动的程序
customer/lib：存放自定义动态库
```

- 修改启动脚本：package/etc/init.d/S99start_app

```
...

start() {
    ...

    run_app "monitor_core.sh"

    run_app "/customer/bin/xxx"

    ...
```

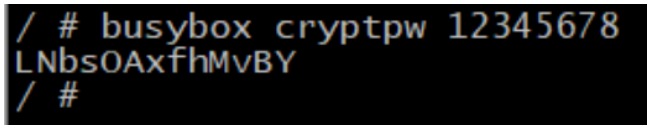
```
    }

    ...
```

- 添加动态库地址：package/etc/profile

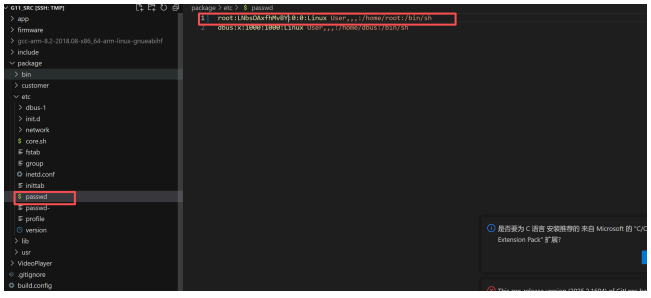
```
    #!/bin/sh
    export PATH=/bin:/sbin:/usr/bin:/usr/sbin
    export LD_LIBRARY_PATH=/lib:/config/lib:/customer/lib
    mkdir -p /dev/pts
    mount -t devpts devpts /dev/pts
    ulimit -c unlimited
    echo "/customer/core.%e.%p.%t" > /proc/sys/kernel/core_pattern
```

1.8 修改telnet密码

- 生成加密的密文：busybox cryptpw xxx



- 修改 etc/passwd 文件

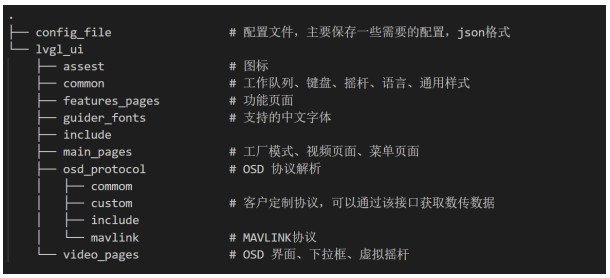


2. UI

2.1 概述

lvgl

- 版本：v8.2
- 分辨率：1920x1080
- 字体
  - 菜单栏：myFont.c，思源黑体
  - OSD：LiberationSans-Regular.ttf，字库

- 目录结构
  - 

```
.
├── config_file                # 配置文件，主要保存一些需要的配置，json格式
└── lvgl_ui
    ├── assest                 # 图标
    ├── common                 # 工作队列、键盘、摇杆、语言、通用样式
    ├── features_pages         # 功能页面
    ├── guider_fonts           # 支持的中文字体
    ├── include
    ├── main_pages             # 工厂模式、视频页面、菜单页面
    ├── osd_protocol           # OSD 协议解析
    │   ├── commom
    │   ├── custom             # 客户定制协议，可以通过该接口获取数传数据
    │   ├── include
    │   └── mavlink            # MAVLINK协议
    └── video_pages            # OSD 界面、下拉框、虚拟摇杆
```

2.2 私有协议开发

实例代码

```cpp
    int buf[64] = {0};
    osd_protocol_init(OSD_PROTOCOL_CUSTOM);
    osd_protocol_data_write(buf , 64);
    while(1) {
        osd_protocol_data_read(buf);
    }

```

2.3 UI 开发

更换图标

- 修改 app/yunzhuo/lvgl_ui/assest 相关的文件
- 在 app/yunzhuo/lvgl_ui/video_pages/Drones_demo.c 中修改加载图片

更换字体

- 菜单栏字体
  - 使用 LvglFontTool 工具转换并替换 app/yunzhuo/lvgl_ui/guider_fonts/myFont.c 文件
- OSD字体
  - 从网上下载 ttf 字库，然后加载字库

```
    static void init_freetype_font(void)
    {
        /*Create a font*/
        /*FreeType uses C standard file system, so no driver letter is
required.*/
        // font_info.name = "/tmp/nfs/SSD202/arial.ttf";
        font_info.name = "/customer/assest/arial.ttf";
```

```
        font_info.weight = 36;
        font_info.style =
FT_FONT_STYLE_BOLD;//FT_FONT_STYLE_BOLD;//FT_FONT_STYLE_NORMAL;
        font_info.mem = NULL;
        if(!lv_ft_font_init(&font_info)) {
            LV_LOG_ERROR("create failed.");
        }
    }

    create_label(parent, text, x, y, font_info.font,
lv_color_hex(0xf0f4f5), 8);
```

---

3. WiFi HAL (hal_wifi.h)

3.1 概述

- WiFi HAL 提供对嵌入式设备WiFi功能的统一抽象，基于单例模式设计，适用于单WiFi接口场景。底层通过 `wpa_supplicant` 进行802.11协议栈管理，支持扫描、连接、状态监控等完整功能。

3.2 功能描述

- WiFi 总开关：启用/禁用 WiFi 硬件
- 已保存网络：显示已保存的 AP，支持点击连接、长按管理（断开/删除/重输密码），最多保存 3 个网络
- 扫描网络：显示附近可用 AP，点击后输入密码连接
- 状态实时反馈：连接中、已连接、密码错误、网络不存在等

3.3 常量定义

常量	值	说明
HAL_WIFI_SSID_MAX_LEN	128	SSID最大长度（不含终止符）
HAL_WIFI_BSSID_MAX_LEN	18	BSSID字符串长度（如 "aa:bb:cc:dd:ee:ff"）
HAL_WIFI_PSK_MAX_LEN	64	密码/PSK最大长度
HAL_WIFI_MAX_SCAN_RESULTS	50	单次扫描最大返回结果数
HAL_WIFI_MAX_SAVED_NETWORKS	20	最大已保存网络数
HAL_WIFI_IFACE_NAME_LEN	16	网络接口名称最大长度

3.4 枚举类型

hal_wifi_security_t — WiFi安全模式

枚举值	说明
HAL_WIFI_SECURITY_NONE	开放网络，无密码
HAL_WIFI_SECURITY_WEP	WEP（已废弃，不推荐）
HAL_WIFI_SECURITY_WPA_PSK	WPA-PSK (TKIP)
HAL_WIFI_SECURITY_WPA2_PSK	WPA2-PSK (AES/CCMP)
HAL_WIFI_SECURITY_WPA3_SAE	WPA3-SAE
HAL_WIFI_SECURITY_WPA2_ENTERPRISE	WPA2-Enterprise (802.1X)
HAL_WIFI_SECURITY_WPA3_ENTERPRISE	WPA3-Enterprise
HAL_WIFI_SECURITY_UNKNOWN	未知/自动检测

hal_wifi_state_t — WiFi连接状态

状态机流转：`IDLE -- SCANNING -- CONNECTING -- AUTHENTICATING -- CONNECTED`

枚举值	说明
HAL_WIFI_STATE_IDLE	空闲，未进行任何操作
HAL_WIFI_STATE_SCANNING	正在扫描
HAL_WIFI_STATE_CONNECTING	正在连接（已发送关联请求）
HAL_WIFI_STATE_CONNECTED	已连接，IP获取成功
HAL_WIFI_STATE_DISCONNECTED	已断开
HAL_WIFI_STATE_CONNECTION_FAILED	连接失败（认证/关联失败）
HAL_WIFI_STATE_AUTHENTICATING	正在认证（WPA/WPA2握手）
HAL_WIFI_STATE_NETWORK_NOT_FOUND	AP不存在

hal_wifi_disconnect_reason_t — 断开原因

枚举值	说明
HAL_WIFI_DISCONNECT_REASON_UNKNOWN	未知原因
HAL_WIFI_DISCONNECT_REASON_USER_REQUEST	用户主动断开
HAL_WIFI_DISCONNECT_REASON_AUTH_FAILED	密码错误或认证失败
HAL_WIFI_DISCONNECT_REASON_ASSOC_FAILED	关联失败（AP拒绝）
HAL_WIFI_DISCONNECT_REASON_SIGNAL_LOST	信号丢失/漫游
HAL_WIFI_DISCONNECT_REASON_NETWORK_NOT_FOUND	未找到指定SSID
HAL_WIFI_DISCONNECT_REASON_DHCP_FAILED	DHCP获取IP失败

枚举值	说明
HAL_WIFI_DISCONNECT_REASON_SUPPLICANT_STOPPED	wpa_supplicant异常停止

3.5 数据结构

hal_wifi_scan_result_t — 扫描结果条目

字段	类型	说明
bssid	char[18]	BSSID，如 "aa:bb:cc:dd:ee:ff"
ssid	char[129]	SSID字符串，以'\0'结尾
signal_level_dbm	int16_t	信号强度，单位dBm（如-45）
frequency_mhz	uint16_t	信道频率，单位MHz（如2412）
security	hal_wifi_security_t	安全类型
max_rate	uint32_t	最大速率，单位Mbps
is_hidden	bool	true = 隐藏SSID

hal_wifi_saved_network_t — 已保存的网络配置

字段	类型	说明
network_id	int	wpa_supplicant网络ID
ssid	char[129]	SSID
security	hal_wifi_security_t	安全类型
auto_connect	bool	true = 开机自动连接
priority	int	连接优先级（数值越大越优先）

hal_wifi_connection_info_t — 当前连接信息

字段	类型	说明
ssid	char[129]	已连接SSID
bssid	char[18]	已连接BSSID
security	hal_wifi_security_t	当前使用的安全类型
rssi_dbm	int16_t	当前信号强度dBm
tx_rate	uint32_t	当前发送速率，单位Kbps
rx_rate	uint32_t	当前接收速率，单位Kbps
local_ip	char[16]	本机IPv4地址

字段	类型	说明
gateway	char[16]	网关地址
dns1	char[16]	主DNS
dns2	char[16]	备DNS

hal_wifi_config_t — WiFi配置参数

字段	类型	说明
iface_name	char[16]	网络接口名，如 "wlan0"
country_code	char[3]	国家码，如 "CN"、"US"、"JP"
bgscan_enabled	bool	后台扫描（漫游优化）
scan_interval_ms	uint32_t	后台扫描间隔，单位毫秒
connect_timeout_ms	uint32_t	连接超时时间，单位毫秒
disconnect_timeout_ms	uint32_t	断开超时时间，单位毫秒

hal_wifi_callbacks_t — 回调函数集合

字段	类型	说明
on_scan_done	hal_wifi_scan_done_cb	扫描完成
on_state_changed	hal_wifi_state_changed_cb	状态变化
on_connected	hal_wifi_connected_cb	连接成功
on_disconnected	hal_wifi_disconnected_cb	断开连接
on_rssi_changed	hal_wifi_rssi_changed_cb	信号强度变化
on_error	hal_wifi_error_cb	错误

3.6 API接口

初始化与配置

- 初始化WiFi HAL层，设置默认配置（iface="wlan0", country="CN"）。必须在任何其他API之前调用。重复调用返回0。

```
int hal_wifi_init(void);
```

- 反初始化WiFi HAL层：如果WiFi仍在启用状态，会先自动调用 `hal_wifi_disable()`

```
void hal_wifi_deinit(void);
```

- 注册/更新全局回调函数：所有回调均为可选，不需要的回调设为NULL即可。

```
int hal_wifi_register_callbacks(const hal_wifi_callbacks_t *cbs, void
*user_data);
```

启用/禁用

- 启用WiFi：启动流程：创建wpa_daemon实例 → 注册底层事件监听 → 应用国家码配置。耗时约500ms~2s。

```
int hal_wifi_enable(void);
```

- 禁用WiFi：关闭流程：停止RSSI轮询 → 断开连接 → 停止wpa_daemon → 清空扫描队列 → 清空连接信息。

```
int hal_wifi_disable(void);
```

扫描

- 启动WiFi扫描（异步，带独立回调）：支持多模块并发请求，扫描完成后所有请求的回调都会被调用（结果广播）。

```
int hal_wifi_start_scan_ex(hal_wifi_scan_done_cb cb, void *user_data);
```

- 获取最近一次扫描结果：复制内部缓存到调用者提供的缓冲区。建议在 `on_scan_done` 回调中调用。

```
int hal_wifi_get_scan_results(hal_wifi_scan_result_t *results, int
max_count);
```

连接管理

- 连接WiFi网络：向wpa_supplicant添加网络配置并触发连接。save=true时，网络会持久化到 `/etc/wpa_supplicant.conf`

```
int hal_wifi_connect(const char *ssid, const char *psk, hal_wifi_security_t
security,
```

```
                        const char *bssid, bool save);
```

- 断开当前连接：发送DISCONNECT命令，停止RSSI轮询。断开原因固定为 `HAL_WIFI_DISCONNECT_REASON_USER_REQUEST`

```
int hal_wifi_disconnect(void);
```

- 连接已保存在wpa_supplicant中的网络（通过network_id）。

```
int hal_wifi_connect_saved(int network_id);
```

状态查询

- 获取当前连接状态：线程安全，可在任意线程调用。

```
hal_wifi_state_t hal_wifi_get_state(void);
```

- 获取当前连接信息：仅在 `HAL_WIFI_STATE_CONNECTED` 状态下有效。

```
int hal_wifi_get_connection_info(hal_wifi_connection_info_t *info);
```

- 获取当前信号强度：已连接时返回缓存值（2秒更新），未连接时返回-999。

```
int16_t hal_wifi_get_rssi(void);
```

网络管理

- 列出wpa_supplicant已保存的网络：包括当前连接的网络。

```
int hal_wifi_list_saved_networks(hal_wifi_saved_network_t *networks, int
max_count);
```

- 删除已保存的网络：删除当前连接的网络会导致断开。

```
int hal_wifi_remove_network(int network_id);
```

- 设置网络的自动连接属性：底层通过 `wpa_cli ENABLE_NETWORK/DISABLE_NETWORK` 实现。

```
int hal_wifi_set_auto_connect(int network_id, bool enable);
```

辅助接口

- 获取MAC地址：读取 /sys/class/net/<iface>/address。格式："aa:bb:cc:dd:ee:ff"

```
int hal_wifi_get_mac_address(char *mac_buf);
```

- 获取IP地址信息：仅在 HAL_WIFI_STATE_CONNECTED 状态下有效。当前仅支持IPv4。

```
int hal_wifi_get_ip_info(char *ip_buf, char *gateway_buf, char *dns_buf);
```

3.7 实例代码

- 初始化与事件注册

```
    void my_wifi_page_create(lv_obj_t *parent) {
        // 1. 注册 WiFi 回调（全局只需一次）
        static hal_wifi_callbacks_t cbs = {
            .on_scan_done = my_scan_done,
            .on_state_changed = my_state_changed,
            .on_connected = my_connected,
            .on_disconnected = my_disconnected,
        };
        hal_wifi_register_callbacks(&cbs, NULL);

        // 2. 创建开关
        lv_obj_t *sw = lv_switch_create(parent);
        lv_obj_add_event_cb(sw, on_enable_switch, LV_EVENT_VALUE_CHANGED,
NULL);

        // 3. 创建列表容器
        lv_obj_t *list = lv_obj_create(parent);
        // ... 设置样式 ...

        // 4. 启动 WiFi（如果已启用）
        if (hal_wifi_get_state() != HAL_WIFI_STATE_IDLE) {
            lv_obj_add_state(sw, LV_STATE_CHECKED);
            hal_wifi_start_scan_ex(my_scan_done, NULL);
        }

        // 5. 创建 UI 刷新定时器
        lv_timer_create(ui_refresh_timer_cb, 1000, NULL);
    }
```

4. 无线射频 HAL (hal_wireless.h)

4.1 概述

无线射频HAL提供对射频模块（地面端/天空端）的控制接口，包括频段管理、功率控制、信道配置、对频等功能。

4.2 数据结构

hal_freqinfo_t — 对频信息

字段	类型	说明
rssi	int	信号质量
self_snr	int	地面端信噪比
self_gain_a	int	地面端天线A增益
self_gain_b	int	地面端天线B增益
self_tx_mcs	int	地面端MCS
self_tx_power	int	地面端功率
peer_snr	int	天空端信噪比
peer_gain_a	int	天空端天线A增益
peer_gain_b	int	天空端天线B增益
peer_tx_mcs	int	天空端MCS
peer_tx_power	int	天空端功率

hal_bandinfo_t — 频段信息

字段	类型	说明
band_mode	uint8_t	频段模式：0=手动，1=自动
work_band	uint8_t	工作频段

hal_powerinfo_t — 功率信息

字段	类型	说明
pwr_auto	uint8_t	自动功率控制
min	uint8_t	最小功率
max	uint8_t	最大功率
pwr_value	uint8_t	当前功率值

hal_chaninfo_t — 信道信息

字段	类型	说明
chn_mode	uint8_t	信道模式
chn_num	uint8_t	信道数量
work_chan	uint8_t	工作信道
freq[32]	uint32_t	频率列表
power[32]	int32_t	功率列表

hal_rf_version_t — 射频版本信息

字段	类型	说明
soft_ver	char[32]	软件版本
hardware_ver	char[32]	硬件版本
firmware_ver	char[32]	固件版本
compile_time	char[32]	编译时间

4.3 API接口

- 初始化射频模块：初始化底层射频硬件，建立与对端设备的通信链路。

```
int hal_wireless_init(void);
```

- 反初始化射频模块：释放射频资源，关闭通信链路。

```
int hal_wireless_deinit(void);
```

- 获取对频信息：包含信号质量、信噪比、天线增益等本端和对端状态。

```
int hal_wireless_get_freq(hal_freqinfo_t *info);
```

- 设置对端设备波特率：修改对端设备的UART通信波特率。

```
int hal_wireless_set_baudrate(uint8_t uart_id, uint32_t baudrate);
```

- 获取对端设备波特率。

```
int hal_wireless_get_baudrate(uint8_t uart_id, uint32_t *baudrate);
```

- 获取频段模式：包含频段模式（手动/自动）和工作频段。

```
int hal_wireless_getbandmode(hal_bandinfo_t *info);
```

- 设置频段模式：配置射频工作频段和自动/手动切换模式。

```
int hal_wireless_setbandmode(hal_bandinfo_t *info);
```

- 获取功率信息：包含自动功率控制、最小/最大功率、当前功率值。

```
int hal_wireless_get_power(hal_powerinfo_t* info);
```

- 设置本端功率：配置射频发射功率参数。

```
int hal_wireless_set_power(hal_powerinfo_t* info);
```

- 设置信道工作模式：包含信道数量、工作信道、频率列表和功率列表。

```
int hal_wireless_set_chninfo(hal_chaninfo_t* info);
```

- 获取信道信息。

```
int hal_wireless_get_chninfo(hal_chaninfo_t* info);
```

- 设置本地射频开关：enable: 0=关闭射频，1=打开射频。

```
int hal_wireless_setrfenable(uint32_t enable);
```

- 获取本地射频开关状态。

```
int hal_wireless_getrfenable(int32_t *enable);
```

- 开始对频：启动与对端设备的频率配对过程。

```
int hal_wireless_startpair(void);
```

- 停止对频：终止正在进行的频率配对过程。

```
int hal_wireless_stoppair(void);
```

- 设置下载模式：需要每4秒调用1次以保持下载模式。

```
int hal_wireless_downloadmode(void);
```

- 设置上传模式：需要每4秒调用1次以保持上传模式。

```
int hal_wireless_uploadmode(void);
```

- 获取对频状态：status: 0=未连接，>0=已连接。

```
int hal_wireless_getpairstatus(uint32_t *status);
```

- 获取本地射频版本信息：包含软件版本、硬件版本、固件版本和编译时间。

```
int hal_wireless_get_local_version(hal_rf_version_t* ver);
```

- 重启射频模块：升级射频成功后需要调用重启该模块

```
int hal_wireless_reboot(void);
```

---

5. MCU状态 (mcu_status.h)

5.1 概述

- MCU状态模块提供对MCU（微控制器）推送的状态数据的访问，包括电压、电量、电源状态和充电状态

5.2 数据结构

mcu_status_t — MCU状态数据

字段	类型	说明
vbat	uint16_t	电压，单位mV，例: 3740mV
bat_duty	uint8_t	电量百分比 0-100
power_state	uint8_t	电源状态: 0=关机，1=准备关机，2=息屏，3=亮屏开机
charge_state	uint8_t	充电状态: 0=没有充电，1=正在充电，2=已充满

5.3 API接口

- 获取MCU当前状态：线程安全，可在任意线程调用。数据由接收线程通过 mcu_status_update_heartbeat() 更新。

```
int mcu_status_get(mcu_status_t* status);
```

- 更新MCU心跳状态（内部接口）：由接收线程调用，更新全局MCU状态缓存。上层应用不应直接调用此函数。

```
void mcu_status_update_heartbeat(mcu_status_t* status);
```

5.4 注意

- 如果检测获取MCU关机状态，没有执行hal_videoplayer_set_disp_status进行息屏，会导致出现残影

---

6. 通用HAL (hal_common.h)

6.1 概述

通用HAL提供固件版本管理、远程固件信息获取、系统状态监控等通用功能。

6.2 数据结构

hal_remote_fw_info_t — 远程固件信息

字段	类型	说明
version	char[128]	远程固件版本号
url	char[512]	固件下载地址
brief	char[2048]	版本说明（中文）
briefEn	char[2048]	版本说明（英文）
versionName	char[128]	固件文件名

hal_hardware_info_t — 硬件信息

字段	类型	说明
brightness	int	亮度

6.3 API接口

- 硬件模块初始化。

```
int hal_hardware_init(hal_hardware_info_t* info);
```

- 负责检测系统状态的线程：启动系统监控线程。

```
int hal_system_status_monitor_init(void);
```

- 停止重启检测线程（用于程序退出时清理）。

```
void hal_reboot_monitor_deinit(void);
```

- 下载远程JSON并解析指定固件的版本与下载地址。

```
int hal_remote_firmware_info_get(const char *target_name,
hal_remote_fw_info_t *info);
```

- 读取当前设备固件版本（/customer/version.txt 或编译宏回退）。

```
int hal_device_current_version_get(char *out_version, int out_size);
```

- 读取当前MCU固件版本。

```
int hal_mcu_current_version_get(char *out_version, int out_size);
```

- 使用libcurl下载文件到指定路径。

```
int hal_firmware_download(const char *url, const char *save_path);
```

---

7. GPIO HAL (hal_gpio.h)

7.1 概述

GPIO HAL 封装了Linux sysfs GPIO接口，提供自动export、方向设置、电平读写和边沿触发配置功能。

7.2 API接口

- 封装GPIO初始化接口：功能：自动检查并export GPIO（如未export）、根据模式设置输入/输出方向、输出模式时设置初始电平。out_flag: 0=输入模式, 1=输出模式。value: 输出模式时的初始值(0=低电平, 1=高电平)

```
int hal_gpio_init(unsigned int gpio, unsigned int out_flag, unsigned int value);
```

- 获取GPIO电平：value: 0=低电平, 1=高电平。

```
int hal_gpio_get_value(unsigned int gpio, unsigned int *value);
```

- 设置引脚触发方式（边沿触发）：edge: "none", "rising", "falling", "both"

```
int hal_gpio_set_edge(unsigned int gpio, char *edge);
```

- GPIO反初始化：释放GPIO资源，取消export。

```
int hal_gpio_deinit(unsigned int gpio);
```

---

8. PWM背光 (hal_pwm.h)

8.1 概述

PWM HAL 封装了Linux sysfs PWM接口，用于控制背光亮度。支持自动export、周期/占空比配置、使能/禁用和亮度百分比设置。

8.2 常量定义

常量	值	说明
MAX_DUTY_CYCLE	78	最大占空比百分比，目前亮度不允许超过78
MIN_DUTY_CYCLE	5	最小占空比百分比

8.3 API接口

- 函数实现

- PWM背光初始化接口：period_ns: PWM周期，单位纳秒（如1000000=1KHz）。duty_ns: 初始占空比，单位纳秒（如500000=50%）。

```
int hal_pwm_init(unsigned int pwm_chip, unsigned int pwm_channel,
                 unsigned int period_ns, unsigned int duty_ns);
```

- 设置PWM背光亮度（占空比）：duty_ns范围: 0 ~ period_ns。

```
int hal_pwm_set_duty(unsigned int pwm_chip, unsigned int pwm_channel,
                     unsigned int duty_ns);
```

- 设置PWM背光亮度百分比：brightness: 亮度百分比 (0 ~ 100)

```
int hal_pwm_set_brightness(unsigned int pwm_chip, unsigned int
pwm_channel,
                           unsigned int brightness);
```

- 使能/禁用PWM输出：enable: 0=禁用, 1=使能。

```
int hal_pwm_enable(unsigned int pwm_chip, unsigned int pwm_channel,
                   unsigned int enable);
```

- 获取当前亮度百分比。

```
unsigned int hal_pwm_get_brightness(void);
```

- PWM背光反初始化：禁用PWM输出，unexport PWM。

```
int hal_pwm_deinit(unsigned int pwm_chip, unsigned int pwm_channel);
```

- 息屏/亮屏控制：enable: 0=息屏，1=亮屏。

```
int hal_pwm_screen_off_on(int enable);
```

---

9. 遥控器调参 (hal_rc_tuner.h)

9.1 概述

遥控器调参模块提供与MCU（遥控器主控）通信的接口，用于配置通道、手型、死区、按键、拨轮等参数，以及获取版本、电量、序列号等信息。

9.2 常量定义

常量	值	说明
RUDDER_CHANNEL_COUNT	16	协议获取的通道数量是16个，但是我们只有11个有效
UI_DISPLAY_CHANNEL_COUNT	11	UI显示通道数
AUX1_INDEX	8	AUX1通道索引
AUX2_INDEX	9	AUX2通道索引

9.3 枚举类型

hal_rc_error_t — 错误码

枚举值	值	说明
HAL_RC_OK	0	成功
HAL_RC_ERROR_NOT_SUPPORTED	-1	不支持
HAL_RC_ERROR_PARAM	-2	参数错误
HAL_RC_ERROR_PROTOCOL	-3	协议错误

TELECTRL_HAND_SETTING — 手型设置

枚举值	说明
TELECTRL_HAND_SETTING_JAPAN	日本手
TELECTRL_HAND_SETTING_AMERICA	美国手
TELECTRL_HAND_SETTING_JAPAN_REVERSE	日本手反向
TELECTRL_HAND_SETTING_AMERICA_REVERSE	美国手反向

9.4 数据结构

hal_chncfg_t — 通道配置

字段	类型	说明
chn	uint8_t	通道编号
minpulse	uint8_t	最小脉冲宽度
midpulse	uint8_t	中位脉冲宽度
maxpulse	uint8_t	最大脉冲宽度

字段	类型	说明
direction	uint8_t	方向

hal_mixedcfg_t — 混控/死区配置

字段	类型	说明
mixctrl_enable	uint8_t	混控使能
deadzone_mode	uint8_t	死区模式

hal_keycfg_t — 按键配置

字段	类型	说明
key_voice	uint8_t	按键声音
key_keep	uint8_t	按键保持

hal_keylockcfg_t — 按键锁定配置

字段	类型	说明
H_lock	uint8_t	H键锁定
B1_lock	uint8_t	B1键锁定
B2_lock	uint8_t	B2键锁定

hal_savecfg_t — 按键保存配置

字段	类型	说明
b1_save	uint8_t	B1保存状态
b2_save	uint8_t	B2保存状态
h_save	uint8_t	H保存状态

hal_wheelcfg_t — 拨轮配置

字段	类型	说明
wheel1_mode	uint8_t	拨轮1模式
wheel2_mode	uint8_t	拨轮2模式

hal_coachcfg_t — 教练模式配置

字段	类型	说明
coach_mode	uint8_t	教练模式使能
sw	uint8_t	开关状态，sw='P'：PPM输出模式；sw='M'：教练控主机模式；sw='S'：学员控从机模式

hal_joystickcfg_t — 摇杆配置

字段	类型	说明
chn	uint8_t	通道编号
value	uint16_t	摇杆值

hal_mcu_status_t — MCU状态

字段	类型	说明
vbat	uint16_t	电压
bat_duty	uint8_t	电量百分比 0-100
power_state	uint8_t	电源状态
charge_state	uint8_t	充电状态

hal_beep_vibrate_t — 蜂鸣器/马达设置

字段	类型	说明
beep	uint8_t	蜂鸣器：0=无，1=短，2=长
vibrate	uint8_t	马达：0=无，1=短，2=长

hal_mcu_version_t — MCU版本

字段	类型	说明
soft_ver	char[8]	软件版本
hard_ver	char[8]	硬件版本

hal_sn_t — 序列号

字段	类型	说明
sn	int8_t[16]	16字节原始SN数据

9.5 API接口

初始化与状态

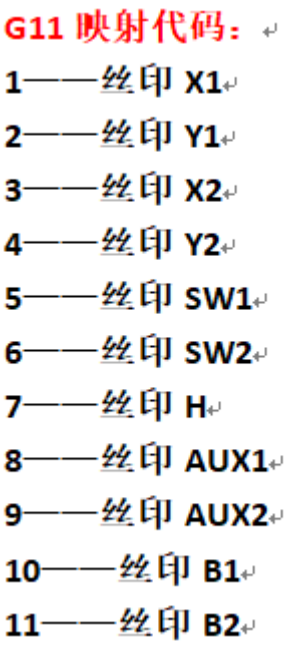
- 函数实现
  - 初始化遥控器调参模块：建立与MCU的通信通道，初始化内部状态。

```
int hal_rc_tuner_init(void);
```

  - 获取MCU状态（电压、电量、电源状态、充电状态）：封装了 `mcu_status_get()`，提供遥控器调参模块专用的状态获取接口。

```
int hal_get_mcu_status(hal_mcu_status_t* status);
```

遥控调参

- 硬件物理通道
  - 

- 概述
  - 目前G11拥有11个物理通道，但是协议获取是16个通道，需要按照过滤前11个通道的数据
  - 设置遥控调参的最大值最小值需要除以10，例如设置范围是1050~1950，则发送105和195，反之获取到的值要乘以10
  - min_rudder 无用
- 函数实现
  - 获取遥控调参：data指向 `hal_chncfg_t` 结构体，接收16通道的配置信息

```
int hal_rc_tuner_get_chn(void *data);
```

- 设置遥控调参：data指向 `hal_chncfg_t` 结构体，负责硬件和软件通道映射

```
int hal_rc_tuner_set_chn(void *data);
```

- 示例代码

```
    static void update_adjust_page(void) {
        hal_chncfg_t cfg = {0};
        int ret = 0;
        ret = hal_rc_tuner_get_chn(&cfg);
        if(ret < 0) {
            printf("hal_rc_tuner_get_chn error\n");
            return ;
        }

        for (int i = 0; i < UI_DISPLAY_CHANNEL_COUNT; i++) {
            lv_obj_t* slider = g_telectrl_obj_mgr.rudder_info.sliders[i];
            int max_value = cfg.chn_cfg[i].maxpulse*10;
            int min_value = cfg.chn_cfg[i].minpulse*10;
            int config_channel = cfg.chn_cfg[i].chn;

            printf("lv_slider_set_range[%d]: %d-%d \n", i, min_value,
max_value);
        }

    }

    static void save_adjust_page(void) {
        hal_chncfg_t cfg = {0};

        for (int i = 0; i < UI_DISPLAY_CHANNEL_COUNT; i++) {
            if (g_telectrl_obj_mgr.telectrl_adjust[i].reverse != NULL) {
                cfg.chn_cfg[i].direction = lv_obj_has_state(
                    g_telectrl_obj_mgr.telectrl_adjust[i].reverse,
                    LV_STATE_CHECKED) ? 0 : 1;
            }

            if (g_telectrl_obj_mgr.telectrl_adjust[i].channel != NULL) {
                cfg.chn_cfg[i].chn = lv_dropdown_get_selected(
                    g_telectrl_obj_mgr.telectrl_adjust[i].channel) + 1;
            }

            if (g_telectrl_obj_mgr.telectrl_adjust[i].min_rudder != NULL) {
                const char* min = lv_textarea_get_text(
                    g_telectrl_obj_mgr.telectrl_adjust[i].min_rudder);
                cfg.chn_cfg[i].minpulse = atoi(min) / 10;
```

```
            }

            if (g_telectrl_obj_mgr.telectrl_adjust[i].max_rudder != NULL) {
                const char* max = lv_textarea_get_text(
                    g_telectrl_obj_mgr.telectrl_adjust[i].max_rudder);
                cfg.chn_cfg[i].maxpulse = atoi(max) / 10;
            }
        }

        // 发送配置到遥控器
        hal_rc_tuner_set_chn(&cfg);
    }
```

手型设置

- 函数实现
  - 获取手型设置（日本手/美国手）：data为 `TELECTRL_HAND_SETTING` 枚举值。

    ```
    int hal_rc_tuner_get_hand(void *data);
    ```

  - 设置手型（日本手/美国手）：data为 `TELECTRL_HAND_SETTING` 枚举值。

    ```
    int hal_rc_tuner_set_hand(void *data);
    ```

- 示例代码

```
static void save_hand_page(void) {
    int cfg = TELECTRL_HAND_SETTING_JAPAN;
    printf("save_hand_page: cfg=%d\n", cfg);
    hal_rc_tuner_set_hand(&cfg);
}

static void update_hand_page(void) {
    hal_rc_tuner_get_hand(&cfg);
    printf("update_hand_page: hand_type=%d\n", cfg);
}
```

死区配置

- 函数实现
  - 获取死区设置：data指向 `hal_mixedcfg_t` 结构体，包含混控使能和死区模式。

```
int hal_rc_tuner_get_dead(void *data);
```

- 设置死区：data指向 `hal_mixedcfg_t` 结构体。

```
int hal_rc_tuner_set_dead(void *data);
```

- 示例代码

```
static void update_mixctrl_page(void) {
    hal_mixedcfg_t cfg;
    hal_rc_tuner_get_dead(&cfg);
}

static void save_mixctrl_page(void) {
    hal_mixedcfg_t cfg = {0};
    printf("save_mixctrl_page\n");
    hal_rc_tuner_set_dead(&cfg);
}
```

按键锁定

- 概述
  - 设置 B1、B2、H 按键是否为锁定的方式
- 函数实现
  - 获取按键锁定状态：data指向 `hal_keylockcfg_t` 结构体，包含H键、B1键、B2键的锁定状态。

```
int hal_rc_tuner_get_key_lock(void *data);
```

  - 设置按键锁定状态：data指向 `hal_keylockcfg_t` 结构体。

```
int hal_rc_tuner_set_key_lock(void *data);
```

- 示例代码

```
static void save_key_lock_page(void) {
    hal_keylockcfg_t cfg = {0};
    hal_rc_tuner_set_key_lock(&cfg);
}
```

```
    static void update_key_lock_page(void) {
        hal_keylockcfg_t cfg;
        hal_rc_tuner_get_key_lock(&cfg);
    }
```

教练模式

- 概述
  - 教练模式，mode='P'：PPM输出模式；mode='M'：教练控主机模式；mode='S'：学员控从机模式
  - 教练开关的按键，7=H, 10=B1, 11=B2
- 函数实现
  - 获取教练模式设置：data指向 `hal_coachcfg_t` 结构体，包含教练模式使能和开关状态。

    ```
    int hal_rc_tuner_get_coach(void *data);
    ```

  - 设置教练模式：data指向 `hal_coachcfg_t` 结构体。

    ```
    int hal_rc_tuner_set_coach(void *data);
    ```

- 示例代码

```
    static void update_coach_page(void) {
        hal_coachcfg_t cfg;
        hal_rc_tuner_get_coach(&cfg);
    }
    static void save_coach_page(void) {
        hal_coachcfg_t cfg = {0};

        // 根据选中的模式设置 coach_mode
        if (g_telectrl_obj_mgr.coach_ui.ppm_mode != NULL &&
            lv_obj_has_state(g_telectrl_obj_mgr.coach_ui.ppm_mode,
LV_STATE_CHECKED)) {
            cfg.coach_mode = 'P';
        } else if (g_telectrl_obj_mgr.coach_ui.coach_host != NULL &&
                lv_obj_has_state(g_telectrl_obj_mgr.coach_ui.coach_host,
LV_STATE_CHECKED)) {
            cfg.coach_mode = 'M';
        } else if (g_telectrl_obj_mgr.coach_ui.coach_slaver != NULL &&
                lv_obj_has_state(g_telectrl_obj_mgr.coach_ui.coach_slaver,
LV_STATE_CHECKED)) {
            cfg.coach_mode = 'S';
```

```
        }

        // 获取教练开关选择
        if (g_telectrl_obj_mgr.coach_ui.coach_switch != NULL) {
            int sw_index =
lv_dropdown_get_selected(g_telectrl_obj_mgr.coach_ui.coach_switch);
            cfg.sw = (sw_index == 0) ? 7 :   // H
                     (sw_index == 1) ? 10 :  // B1
                     (sw_index == 2) ? 11 :  // B2
                     7;
        }

        printf("save_coach_page\n");
        hal_rc_tuner_set_coach(&cfg);
    }

    static void update_coach_page(void) {
        hal_coachcfg_t cfg;
        hal_rc_tuner_get_coach(&cfg);
    }
```

按键保存

- 概述
  - 设置B1、B2、H的按键掉电之后是否保存
- 函数实现
  - 获取按键保存设置：data指向 `hal_savecfg_t` 结构体，包含B1、B2、H键的保存状态。

    ```
    int hal_rc_tuner_get_key_save(void *data);
    ```

  - 设置按键保存：data指向 `hal_savecfg_t` 结构体。

    ```
    int hal_rc_tuner_set_key_save(void *data);
    ```

- 示例代码

```
    static void update_keysave_page(void) {
        hal_savecfg_t cfg;
        hal_rc_tuner_get_key_save(&cfg);
    }
    static void save_keysave_page(void) {
        hal_savecfg_t cfg = {0};
        hal_rc_tuner_set_key_save(&cfg);
```

```
    }
```

按键配置

- 概述
  - 设置按键声音以及按键保持
- 函数实现
  - 获取按键配置：data指向 `hal_keycfg_t` 结构体，包含按键声音和保持设置。

```
int hal_rc_tuner_get_key(void *data);
```

  - 设置按键配置：data指向 `hal_keycfg_t` 结构体。

```
int hal_rc_tuner_set_key(void *data);
```

舵量查看

- 概述
  - 主要获取摇杆的舵量值，源码中是使用一个线程异步获取舵量值并上报给UI进行显示
- 函数实现
  - 获取摇杆配置：data指向 `hal_joystickcfg_t` 结构体，包含16通道摇杆值。

```
int hal_rc_tuner_get_rudder(void *data);
```

- 示例代码

```
    static void *_rudder_refresh_thread(void *param) {
        (void)param;
        hal_joystickcfg_t cfg;

        printf("[INFO] Rudder refresh thread started (TID: %lu)\n", (unsigned
long)pthread_self());

        while (g_rudder_refresh_running) {
            if (hal_rc_tuner_get_rudder(&cfg) == 0) {
                pthread_mutex_lock(&g_rudder_cache.lock);
                for (int i = 0; i < RUDDER_CHANNEL_COUNT; i++) {
                    g_rudder_cache.values[i] = cfg.rudder[i].value;
```

```
                    g_rudder_cache.valid[i] = 1;
                }
                g_rudder_cache.data_ready = 1;
                pthread_mutex_unlock(&g_rudder_cache.lock);
            }

            usleep(200 * 1000);
        }

        printf("[INFO] Rudder refresh thread terminated\n");
        return NULL;
    }
```

版本信息

- 概述
  - 获取MCU的版本信息
- 函数实现
  - 获取MCU版本信息：data指向 `hal_mcu_version_t` 结构体，包含软件版本和硬件版本。

```
int hal_rc_tuner_get_version(void *data);
```

- 示例代码

```
    hal_mcu_version_t version;
    if (hal_rc_tuner_get_version(&version) == 0) {
        strncpy(hard_ver, version.hard_ver, sizeof(hard_ver) - 1);
        strncpy(local_version, version.soft_ver, sizeof(local_version) - 1);
        local_ok = true;
    }
```

序列号

- 概述
  - 获取序列号，-1代表结尾
- 函数实现
  - 获取设备序列号：data指向 `hal_sn_t` 结构体，接收16字节原始SN数据。 `c int hal_rc_tuner_get_sn(void *data);`
- 示例代码

```
    hal_sn_t sn = {0};
    for (int i = 0; i < 16; i++) {
        sn.sn[i] = -1;
    }
    hal_rc_tuner_get_sn(&sn);
    char sn_hex_str[33] = {0};
    char sn_txt[17] = {0};
    int txt_pos = 0;

    for (int i = 0; i < 16 && sn.sn[i] != -1; i++) {
        uint8_t b = (uint8_t)sn.sn[i];

        // sn_hex: 十六进制字符串
        sprintf(&sn_hex_str[i * 2], "%02X", b);

        // sn_txt: 只显示 ASCII 可打印字符 (0x20-0x7E)
        if (b >= 0x20 && b <= 0x7E) {
            sn_txt[txt_pos++] = (char)b;
        }
    }

    if (g_system_setting_obj_mgr.info.sn_hex != NULL) {
        lv_label_set_text(g_system_setting_obj_mgr.info.sn_hex,
                          sn_hex_str[0] ? sn_hex_str : "N/A");
    }
    if (g_system_setting_obj_mgr.info.sn_txt != NULL) {
        lv_label_set_text(g_system_setting_obj_mgr.info.sn_txt,
                          sn_txt[0] ? sn_txt : "N/A");
    }
```

---

10. 云台控制 (hal_top.h)

10.1 概述

- 需要使用云卓科技自带的云台才能生效
- 云台控制HAL通过UDP协议与云台设备通信，提供俯仰/航向角度控制、速度控制、拍照录像、数码变焦等功能。

10.2 错误码

枚举值	值	说明
HAL_TOP_OK	0	成功
HAL_TOP_ERR_INIT	-1	初始化错误
HAL_TOP_ERR_PARAM	-2	参数错误
HAL_TOP_ERR_SEND	-3	发送错误

枚举值	值	说明
HAL_TOP_ERR_BUSY	-4	设备忙
HAL_TOP_ERR_NO_MEM	-5	内存不足
HAL_TOP_ERR_TIMEOUT	-6	查询超时

10.3 枚举类型

hal_top_record_status_t — 录像状态

枚举值	值	说明
HAL_TOP_RECORD_STATUS_UNKNOWN	-1	未知/未查询
HAL_TOP_RECORD_STATUS_STOPPED	0	未录像
HAL_TOP_RECORD_STATUS_RECORDING	1	正在录像

hal_top_dzm_action_t — 数码变焦动作

枚举值	值	说明
TOP_DZM_1X	0x01	1x变焦
TOP_DZM_2X	0x02	2x变焦
TOP_DZM_3X	0x03	3x变焦
TOP_DZM_4X	0x04	4x变焦
TOP_DZM_ZOOM_IN	0x0A	Zoom+ 放大
TOP_DZM_ZOOM_OUT	0x0B	Zoom- 缩小
TOP_DZM_TELE	0x0C	长焦
TOP_DZM_WIDE	0x0D	短焦

10.4 数据结构

hal_top_dzm_info_t — 数码变焦查询结果

字段	类型	说明
value	uint8_t	原始步伐值 X0X1
is_tele	bool	true=长焦, false=短焦 (value < 70短焦, >=70长焦)
zoom_ratio	float	估算的变焦倍数（仅参考）

10.5 API接口

初始化

- HAL层初始化。建立与云台设备的UDP通信通道。

```
int hal_top_init(uint16_t local_port, const char *remote_ip, uint16_t
remote_port);
```

- HAL层反初始化。关闭UDP通信通道。

```
void hal_top_deinit(void);
```

角度模式控制

- 设置云台俯仰角度：angle_deg范围 [-90.0, +90.0]，正值向上。speed_deg_per_sec范围 (0, 63.5]

```
int hal_top_set_pitch_angle(int angle_deg, int speed_deg_per_sec);
```

- 设置云台航向角度：angle_deg范围 [-90.0, +90.0]，正值向右。

```
int hal_top_set_yaw_angle(int angle_deg, int speed_deg_per_sec);
```

- 联合设置云台航向+俯仰角度。

```
int hal_top_set_yaw_pitch_angle(int yaw_deg, int pitch_deg,
                                int yaw_speed_deg_per_sec,
                                int pitch_speed_deg_per_sec);
```

速度模式控制

- 设置云台俯仰转动速度：speed_deg_per_sec范围 [-63.5, +63.5]，正值向上转，负值向下转，0停止。速度模式：持续转动，不指定目标角度。

```
int hal_top_set_pitch_speed(int speed_deg_per_sec);
```

- 设置云台航向转动速度：speed_deg_per_sec范围 [-63.5, +63.5]，正值向右转（顺时针），负值向左转（逆时针），0停止。

```
int hal_top_set_yaw_speed(int speed_deg_per_sec);
```

- 联合设置云台航向+俯仰转动速度：yaw_speed: 右为正。pitch_speed: 上为正。

```
int hal_top_set_yaw_pitch_speed(int yaw_speed_deg_per_sec, int
pitch_speed_deg_per_sec);
```

- 停止云台所有轴转动（发送0速度）。

```
int hal_top_stop_gimbal(void);
```

拍照录像控制

- 拍照

```
int hal_top_capture(void);
```

- 开始录像

```
int hal_top_record_start(void);
```

- 停止录像

```
int hal_top_record_stop(void);
```

- 录像状态翻转（开始/停止切换）

```
int hal_top_record_toggle(void);
```

- 发送录像状态查询命令：异步查询，云台响应后通过回调更新内部状态。

```
int hal_top_record_query(void);
```

- 获取当前缓存的录像状态。

```
hal_top_record_status_t hal_top_get_record_status(void);
```

数码变焦控制

- 设置数码变焦：action为变焦动作，如 `HAL_TOP_DZM_ZOOM_IN`（放大一步）或 `HAL_TOP_DZM_2X`（切换到2x）。

```
int hal_top_set_dzm(hal_top_dzm_action_t action);
```

- 查询当前数码变焦值：异步查询，内部通过回调接收响应后填充info。

```
int hal_top_get_dzm(hal_top_dzm_info_t *info);
```

---

11. 视频播放控制接口 (hal_videoplayer.h)

11.1 概述

- 视频播放器HAL提供视频播放模式控制、RTSP流切换、显示状态管理等功能。支持RTSP、UDP、图片等多种播放模式。
- 主要用于和 VideoPlayer 进行通信，VideoPlayer 主要负责解码、显示功能

11.2 枚举类型

PlayMode_e — 播放模式

枚举值	值	说明
PLAY_MODE_NONE	0	无播放
PLAY_MODE_RTSP	1	RTSP模式
PLAY_MODE_UDP	2	UDP模式
PLAY_MODE_PICTURE	3	图片模式

PowerState_e — 电源状态

枚举值	说明
POWER_STATE_OFF	关机
POWER_STATE_READY	准备关机
POWER_STATE_SCREEN_OFF	息屏
POWER_STATE_SCREEN_ON	亮屏

11.3 数据结构

VideoRtsp_t — RTSP专用结构体

字段	类型	说明
rtsp_url1	char[64]	RTSP主URL
rtsp_url2	char[64]	RTSP备URL

PlayModePayload_t — 播放模式载荷

字段	类型	说明
mode	PlayMode_e	播放模式
rtsp_url1	char[128]	RTSP主URL
rtsp_url2	char[128]	RTSP备URL
udp_port	uint32_t	UDP端口

DispStatusPayload_t — 显示状态载荷

字段	类型	说明
status	int	0=off, 1=on

RtsSwitchPayload_t — 流切换载荷

字段	类型	说明
stream_index	int	0 or 1

AckPayload_t — 应答载荷

字段	类型	说明
code	int	0=success, -1=error
msg	char[64]	应答消息

StatusPayload_t — 状态载荷

字段	类型	说明
current_mode	PlayMode_e	当前模式
disp_status	int	显示状态
stream_active	int	流活动状态
current_rtsp_idx	int	当前RTSP索引
rtsp_url1	char[128]	RTSP主URL
rtsp_url2	char[128]	RTSP备URL

11.4 API接口

- 概述
  - 这些接口都可能导致阻塞，在调用的时候最好使用异步的方式，否则会导致UI卡住
- 函数实现
  - 设置视频播放模式：mode: RTSP/UDP/图片/无。priv为模式相关的私有参数（如RTSP URL、UDP端口等）。切换播放模式时会自动停止当前播放

```
int hal_videoplayer_set_playmode(PlayMode_e mode, void* priv);
```

    - mode为PLAY_MODE_RTSP，priv的结构为VideoRtsp_t，表示拉取的rtsp码流地址
    - mode为PLAY_MODE_UDP，priv的结构为int表示监听端口
    - mode为PLAY_MODE_PICTURE，priv无效，表示显示启动页面
  - 请求切换RTSP视频流：在双RTSP URL模式下切换主/备视频源。

```
int hal_videoplayer_request_rtsp_switch(void);
```

  - 获取当前视频流状态：status包含当前模式、显示状态、活动流索引、RTSP URL等。

```
int hal_videoplayer_get_stream_status(StatusPayload_t* status);
```

  - 设置显示状态（亮屏/息屏）：status: 0=关闭显示，1=打开显示。仅控制视频显示，不关闭底层播放器。

```
int hal_videoplayer_set_disp_status(int status);
```

    - 目前主要用于重启时候做屏幕的反初始化，防止重新上电会出现残影
- 示例代码
  - 配置视频设置

```
    int config_video_ctrl_start(void) {
        int video_port = 0;
        video_mode_t video_mode = VIDEO_MODE_RAW;

        hal_videoplayer_set_playmode(PLAY_MODE_PICTURE, NULL);
        usleep(100*1000);
```

```
        config_low_latency_get(&video_port, &video_mode);

        if (video_mode == VIDEO_MODE_RAW) {
            unsigned int udp_port = (unsigned int)video_port;

            return hal_videoplayer_set_playmode(
                PLAY_MODE_UDP,
                (void *)&udp_port
            );

        } else if (video_mode == VIDEO_MODE_RTP) {
            VideoRtsp_t rtsp_info;

            memset(&rtsp_info, 0, sizeof(rtsp_info));
            config_video_rtsp_get(
                rtsp_info.rtsp_url1,
                rtsp_info.rtsp_url2
            );

            return hal_videoplayer_set_playmode(
                PLAY_MODE_RTSP,
                (void *)&rtsp_info
            );
        }

        printf("Unsupported video mode: %d\n", video_mode);
        return -1;
    }
```

- 切换 rtsp 码流

```
    static void do_rtsp_switch(void *data, int *result)
    {
        (void)data;
        struct timespec ts_start, ts_end;
        clock_gettime(CLOCK_MONOTONIC, &ts_start);
        // force_black_screen_start();
        hal_videoplayer_request_rtsp_switch();
        clock_gettime(CLOCK_MONOTONIC, &ts_end);
        printf("[HAL] connect took %ld ms\n", (ts_end.tv_sec -
ts_start.tv_sec) * 1000 + (ts_end.tv_nsec - ts_start.tv_nsec) /
1000000);
        *result = 0;
    }
```

- 获取 VideoPlayer 的一些状态信息

```
    static void *stream_status_thread(void *arg)
    {
        (void)arg;

        while (g_stream_status_running) {
            StatusPayload_t status = {0};
            int ret = hal_videoplayer_get_stream_status(&status);

            pthread_mutex_lock(&g_stream_cache.lock);
            g_stream_cache.valid = (ret >= 0);
            g_stream_cache.stream_active = (ret >= 0) ?
(status.stream_active != 0) : 0;
            g_stream_cache.data_ready = 1;
            pthread_mutex_unlock(&g_stream_cache.lock);

            usleep(500000);  /* 500ms */
        }

        return NULL;
    }
```

- 关闭屏幕

```
    const char *target_dir = "/customer/update";
    const char *target_file = "/customer/update/update.tgz";

    char mkdir_cmd[128];
    snprintf(mkdir_cmd, sizeof(mkdir_cmd), "mkdir -p %s", target_dir);
    system(mkdir_cmd);

    char cp_cmd[512];
    snprintf(cp_cmd, sizeof(cp_cmd), "cp \"%s\" %s",
g_selected_upgrade_file, target_file);
    int copy_ret = system(cp_cmd);
    if (copy_ret != 0) {
        printf("Failed to copy file to %s\n", target_file);
        return;
    }

    printf("File copied successfully to %s\n", target_file);
    system("sync");

    lv_msgbox_t *info_box = lv_msgbox_create(NULL,
_(STR_COMMON_PROMPT),
                                             _(STR_SYS_UPGRADE_PREPARE),
NULL, false);
    lv_obj_center(info_box);
```

```
        printf("Rebooting system...\n");
        hal_videoplayer_set_disp_status(0);
        usleep(200 * 1000);
        sync();
        system("reboot");
        break;
```

---

12. 升级功能

12.1 本地固件升级操作文档(SOC升级为例)

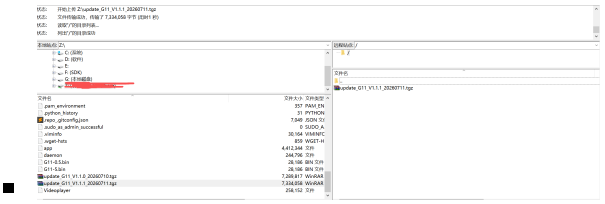
升级前准备

项目	说明
所需工具	FileZilla FTP客户端、固件文件
网络环境	确保PC与遥控器设备处于同一局域网（IP地址：`192.168.144.110`）
固件文件	`update_G11_V1.1.1_20260711.tgz`

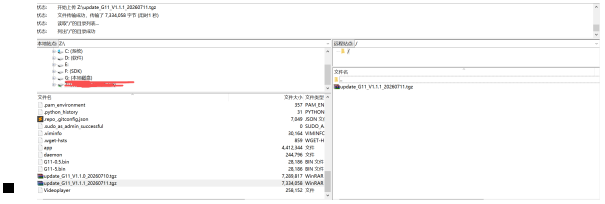
> ⚠ **注意**：升级前请确保设备电量充足，升级过程中**不要退出程序、断开连接或关闭电源**，否则可能导致设备变砖。

上传固件文件到设备

- 步骤1：连接FTP服务器
  - 打开 **FileZilla** 客户端
  - 在顶部连接栏输入以下信息：
    - **主机(H)**：`192.168.144.110`
    - **用户名(U)**：`root`
    - **密码(W)**：输入设备密码（G11目前没有）
    - **端口(P)**：留空（默认21端口）
  - 点击 **快速连接(Q)** 按钮
- 步骤2：输入密码确认
  - 弹出密码输入对话框时，输入密码
  - 勾选 **"在FileZilla关闭前记住密码"**（可选）
  - 点击 **确定(O)**



- 步骤3：上传固件文件
  - 连接成功后，左侧显示**本地站点**，右侧显示**远程站点**（设备根目录 `/`）
  - 在左侧本地文件夹中找到固件文件：
    - `update_G11_V1.1.1_20260711.tgz`
  - 将固件文件**拖拽**到右侧远程站点的根目录 `/` 下
  - 等待传输完成

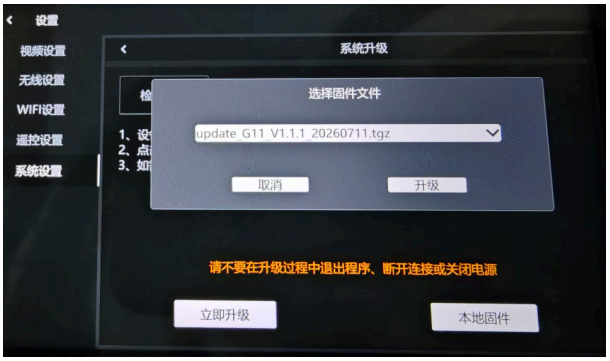


设备端执行本地升级

- 进入升级界面
  - 在遥控器设备上，进入 **设置 → 系统设置 → 检查升级**
  - 找到 **"系统升级"** 选项
  - 点击 **"本地固件"** 按钮

选择固件文件

- 弹出 **"选择固件文件"** 对话框
- 系统会自动列出已上传的固件文件：
- 确认文件名正确后，点击 **"升级"** 按钮



开始升级

- 点击 **"立即升级"** 开始固件刷写
- 等待升级进度完成，设备会自动重启

升级后验证

检查项	验证方法
版本号	进入设置查看当前固件版本
功能测试	测试遥控器各功能是否正常
连接稳定性	检查与接收端的通信是否正常

常见问题排查

问题	可能原因	解决方法
FTP连接失败	IP地址错误或网络不通	检查设备IP，确认PC与设备同网段
传输中断	网络不稳定	重新上传，确保网络稳定
找不到固件文件	文件未上传到根目录	确认文件上传到 / 目录下
升级失败	固件文件损坏或不匹配	重新下载正确版本的固件文件

12.2 远程升级

- 概述
  - 代码中`UI_SYS_SETTING_DEVICE_UPGRADE_PAGE`app固件升级，调用`hal_remote_firmware_info_get`获取升级json文件，然后在调用`hal_firmware_download`下载固件，最后进入升级流程
  - `hal_remote_firmware_info_get` 是自己封装的解析远程固件信息的接口，如果需要自己维护固件，需要`hal_firmware_download`下载自己的文件，然后解析下载
- 示例代码

```
    static void upgrade_remote_file_event_cb(lv_event_t * e) {

        // 2. 获取远程固件信息（HAL 层使用 libcurl 下载 JSON 并解析）
        hal_remote_fw_info_t fw_info = {0};
        if (hal_remote_firmware_info_get(target_name, &fw_info) != 0 ||
fw_info.url[0] == '\0') {
            ...
            return;
        }

        ...

        // 4. 下载固件（HAL 层使用 libcurl，不调用 system）
```

```
        ...

        if (hal_firmware_download(fw_info.url, firmware_path) != 0) {
            if(text_label) lv_label_set_text(text_label,
_(STR_SYS_UPGRADE_FAIL));
            return;
        }

        // 5. 执行对应升级
        int ret = 0;
        switch(g_current_page_index) {
            case UI_SYS_SETTING_DEVICE_UPGRADE_PAGE: {
                if(text_label) lv_label_set_text(text_label,
_(STR_SYS_UPGRADE_PREPARE));
                hal_videoplayer_set_disp_status(0);
                usleep(200 * 1000);
                sync();
                system("reboot");
                break;
            }
            case UI_SYS_SETTING_MCU_UPGRADE_PAGE: {
                ret = hal_rc_tuner_upgrade(firmware_path);
                if(text_label) {
                    lv_label_set_text(text_label, ret < 0 ?
_(STR_SYS_UPGRADE_FAIL) : _(STR_SYS_UPGRADE_SUCCESS));
                }
                break;
            }
            case UI_SYS_SETTING_LOCAL_RF_UPGRADE_PAGE: {
                char cmd[512];
                snprintf(cmd, sizeof(cmd), "update_RF.sh LDEV \"%s\"",
firmware_path);
                printf("Executing: %s\n", cmd);
                ret = system(cmd);
                printf("update_RF.sh LDEV executed, ret=%d\n", ret);

                if (ret == 0) {
                    printf("Rebooting wireless module after remote RF
upgrade...\n");
                    hal_wireless_reboot();
                }

                if(text_label) {
                    lv_label_set_text(text_label, ret != 0 ?
_(STR_SYS_UPGRADE_FAIL) : _(STR_SYS_UPGRADE_SUCCESS));
                }
                break;
            }
            case UI_SYS_SETTING_RECV_RF_UPGRADE_PAGE: {
                ret = 0;
                if(text_label) {
                    lv_label_set_text(text_label, ret < 0 ?
_(STR_SYS_UPGRADE_FAIL) : _(STR_SYS_UPGRADE_SUCCESS));
                }
```

```
                break;
            }
            default:
                break;
        }

        ...
    }
```

---

13. 调试方式

13.1 telnet

- 登录
  - 默认账号是root，密码为空，如果更改了passwd，则修改密码即可
  - 密码为空（直接回车）

13.2 nfs

虚拟机

- 虚拟机网络也要桥接到同一网卡，并配置为同一网段，如192.168.144.101
- 虚拟机设置nfs的目录为 /home/qxn/share(该目录以实际为准)
  - 在 /etc/exports 下添加挂载目录，我以 /home/qxn/share 为例

```
    qxn@ubuntu:/etc$  cat /etc/exports
    # /etc/exports: the access control list for filesystems which may
be exported
    #               to NFS clients.  See exports(5).
    #
    # Example for NFSv2 and NFSv3:
    # /srv/homes       hostname1(rw,sync,no_subtree_check)
hostname2(ro,sync,no_subtree_check)
    #
    # Example for NFSv4:
    # /srv/nfs4
gss/krb5i(rw,sync,fsid=0,crossmnt,no_subtree_check)
    # /srv/nfs4/homes  gss/krb5i(rw,sync,no_subtree_check)
    #

    /home/nfs *(rw,sync,no_root_squash)
    /home/qxn/share *(rw,sync,no_root_squash)
```

- 修改后重启nfs

```
sudo exportfs -ra
sudo systemctl restart nfs-server  # 或 service nfs-kernel-server restart
```

- 拷贝编译的程序到nfs目录

```
qxn@ubuntu:~/code/extern/G11_SRC$ cp package/bin/app /home/qxn/share
qxn@ubuntu:~/code/extern/G11_SRC$ ls /home/qxn/share
    app
```

设备

- 先 telnet 进入设备
- 设备挂载nfs

```
    mkdir /tmp/nfs;mount -t nfs -o nolock 192.168.144.101:/home/qxn/share
/tmp/nfs/;cd /tmp/nfs
```

- 运行可执行文件

```
killall app.sh app;/tmp/nfs/app
```

- 注意
  - 必须先杀掉守护脚本 `app.sh`，再杀 `app`，app.sh 是 watchdog 脚本，会监控 app 进程并在崩溃后自动重启。只杀 app 会被立即重新拉起

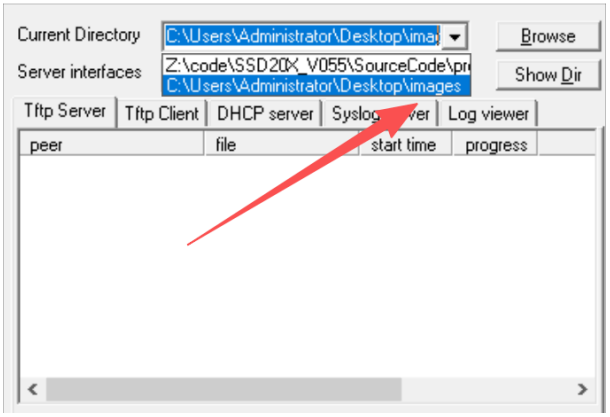
14. 救砖流程

14.1 异常现象

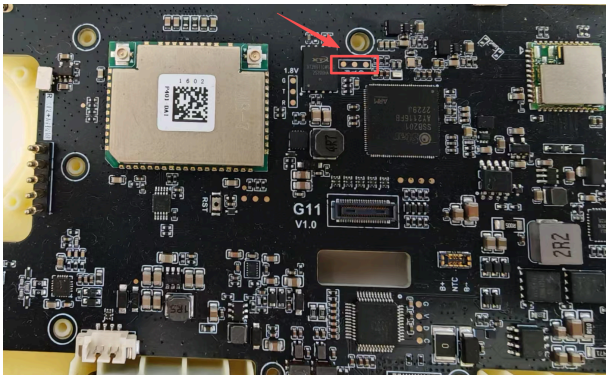
- 升级异常升级包，导致启动失败
- 异常操作导致文件系统的文件丢失，导致启动异常

14.2 软件设置

- 将电脑IP地址设置为192.168.144.100
- 将`image/g11_v1.1.2_image_20260714.tgz`下载到桌面并解压
- 启动tftp服务器并设置目录

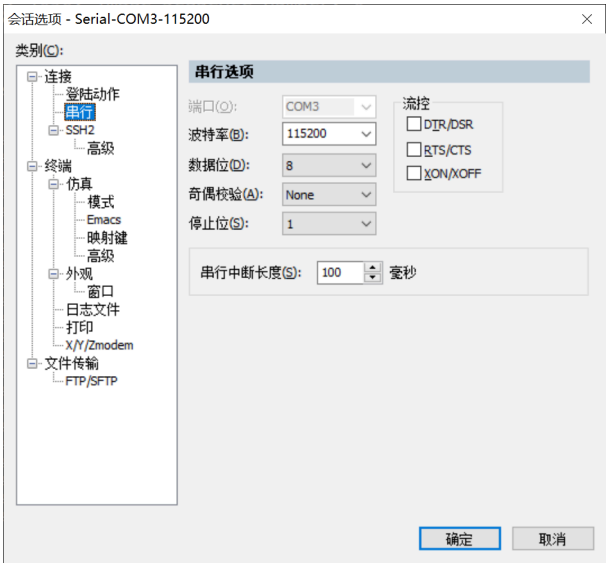
- Windows 推荐：Tftpd64 / Tftpd32
  - 

14.3 硬件

- 将串口线引出接入到串口工具
  - 

- 设备的TX要和串口工具的RX连接，设备的RX和串口工具的TX连接，GND直接相连

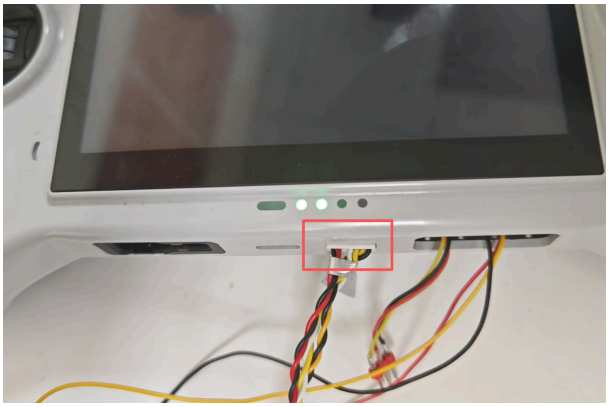
14.4 操作流程

- 设置终端的串口波特率为 115200 波特率，8 数据位，无校验，1 停止位，无流控
  - 

- 在终端工具中**持续按住回车不放**，然后给设备上电，看到`SigmaStar #`提示后松开回车

```
SPI 54M
Load UBOOT from SPINAND
 -Verify UBOOT CRC32 passed!
 -Decompress UBOOT XZ
  decomp_size=0x000a3dd4
Disable MMU and D-cache before jump to UBOOT U-Boot 2015.01 (Jul 03 2026 - 10:51:14)

Version: I2gbd2ee54
[WDT] Enalbe WATCHDOG 10s
      Watchdog enabled
I2C:   ready
DRAM:
WARNING: Caches not enabled
SPINAND: _MDrv_SPINAND_GET_INFO: Found SPINAND INFO
(0xB) (0x15) (0x0)
SPINAND: board_nand_init: CIS contains part info
128 MiB
MMC:   MStar SD/MMC: 0
In:    serial
Out:   serial
Err:   serial
Net:   MAC Address 00:30:1B:BA:02:DB
sstar_emac
SigmaStar #
SigmaStar #
SigmaStar #
SigmaStar #
SigmaStar #
SigmaStar #
SigmaStar #
```

- 将设备和电脑连接在同一个局域网下



- 输入 estar scripts_bin/[[rootfs.es

```
SigmaStar # ping 192.168.144.100
Using sstar_emac device
host 192.168.144.100 is alive
SigmaStar # estar scripts_bin/[[rootfs.es
Using sstar_emac device
TFTP from server 192.168.144.100; our IP address is 192.168.144.110
Filename 'scripts_bin/[[rootfs.es'.
Load address: 0x23a01f00
Loading: T #
         0 Bytes/s
done
Bytes transferred = 179 (b3 hex)

>> ubi part UBI
UBI: parsing mtd_dev string 'mtd=11'
UBI: attaching mtd1 to ubi0
UBI: scanning is finished
UBI: attached mtd1 (name "mtd=11", size 112 MiB) to ubi0
UBI: PEB size: 131072 bytes (128 KiB), LEB size: 126976 bytes
UBI: min./max. I/O unit sizes: 2048/2048, sub-page size 2048
UBI: VID header offset: 2048 (aligned 2048), data offset: 4096
UBI: good PEBs: 899, bad PEBs: 4, corrupted PEBs: 0
UBI: user volume: 4, internal volumes: 1, max. volumes count: 128
UBI: max/mean erase counter: 4/2, WL threshold: 4096, image sequence number: 0
UBI: available PEBs: 9, total reserved PEBs: 890, PEBs reserved for bad PEB handling: 16

>> tftp 0x21000000 rootfs.ubifs
Using sstar_emac device
TFTP from server 192.168.144.100; our IP address is 192.168.144.110
Filename 'rootfs.ubifs'.
Load address: 0x21000000
Loading: ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################
         ########
```

- 当看到 `Bytes transferred = 20951040 (13fb000 hex)` 表示升级成功
- 烧录成功后输入`reset`重启即可