API

HF-LPT26X SDK API 参考手册

目录

1. LINUX 标准 C 函数

HSF-LPT262 兼容标准 c 库的函数，例如内存管理，字符串，时间，标准输入输出等，有关函数的说明请参考标准 c 库函数说明。

mbedtls_compatible_cyassl.h 头文件和例子代码 ssltest.c 文件。

2. 系统错误码定义

API 函数返回值（特别说明除外）规定，成功 HF_SUCCESS，或者>0、失败<0。 错误码为 4Bytes 有符号整数，返回值为错误码的负数。 31-24bit 为模块索引，23-8 保留，7-0，为具体的错误码。

```
#define MOD_ERROR_START(x) ((x << 16) | 0)
/* Create Module index */
#define MOD_GENERIC 0
/** HTTPD module index */
#define MOD_HTTPDE 1
/** HTTP-CLIENT module index */
#define MOD_HTTPC 2
/** WPS module index */
#define MOD_WPS 3
/** WLAN module index */
#define MOD_WLAN 4
/** USB module index */
#define MOD_USB 5
/*0x70~0x7f user define index*/
#define MOD_USER_DEFINE (0x70)
/* Globally unique success code */
#define HF_SUCCESS 0
enum hf_errno {
/* First Generic Error codes */
   HF_GEN_E_BASE = MOD_ERROR_START(MOD_GENERIC),
   HF_FAIL,
   HF_E_PERM, /* Operation not permitted */
   HF_E_NOENT, /* No such file or directory */
   HF_E_SRCH, /* No such process */
   HF_E_INTR, /* Interrupted system call */
   HF_E_IO, /* I/O error */
   HF_E_NXIO, /* No such device or address */
   HF_E_2BIG, /* Argument list too long */
   HF_E_NOEXEC, /* Exec format error */
   HF_E_BADF, /* Bad file number */
   HF_E_CHILD, /* No child processes */
   HF_E_AGAIN, /* Try again */
   HF_E_NOMEM, /* Out of memory */
   HF_E_ACCES, /* Permission denied */
```

```
    HF_E_FAULT, /* Bad address */
    HF_E_NOTBLK, /* Block device required */
    HF_E_BUSY, /* Device or resource busy */
    HF_E_EXIST, /* File exists */
    HF_E_XDEV, /* Cross-device link */
    HF_E_NODEV, /* No such device */
    HF_E_NOTDIR, /* Not a directory */
    HF_E_ISDIR, /* Is a directory */
    HF_E_INVAL, /* Invalid argument */
    HF_E_NFILE, /* File table overflow */
    HF_E_MFILE, /* Too many open files */
    HF_E_NOTTY, /* Not a typewriter */
    HF_E_TXTBSY, /* Text file busy */
    HF_E_FBIG, /* File too large */
    HF_E_NOSPC, /* No space left on device */
    HF_E_SPIPE, /* Illegal seek */
    HF_E_ROFS, /* Read-only file system */
    HF_E_MLINK, /* Too many links */
    HF_E_PIPE, /* Broken pipe */
    HF_E_DOM, /* Math argument out of domain of func */
    HF_E_RANGE, /* Math result not representable */
    HF_E_DEADLK, /*Resource deadlock would occur*/
};
```

头文件:

hferrno.h

3. AT 命令 API

● hfat_get_words

函数原型:

int HSF_API hfat_get_words((char *str,char *words[],int size);

说明:

获取 AT 命令响应的每一个参数值

参数:

str：指向 AT 命令请求或者响应;对应的 RAM 地址一定可读写；

words：保存每一个参数值；

size：words 的个数

返回值:

<=0：str 对应的字符串不是正确的 AT 命令或者非法响应;

>0：对应字符串中包含 Word 的个数；

备注:

AT 命令以”,”,”=”,” “,” \r\n”分隔；

头文件:

hfat.h

● hfat_send_cmd

函数原型:

int HSF_API hfat_send_cmd(char *cmd_line,int cmd_len,char *rsp,int len) ;

说明:

发送 AT 命令，结果返回到指定的 buffer

参数:

cmd_line: 包含 AT 命令字符串；

格式为 AT+CMD_NAME[=][arg，]...[argn];

cmd_len: cmd_line 的长度，包括结束符；

rsp: 保存 AT 命令执行结果的 buffer;

Len: rsp 的长度；

返回值:

HF_success: 设置成功，HF_FAIL: 执行失败

备注:

函数执行和通过串口发送 AT 命令一样，当前不支持”AT+H“和”AT+WSCAN“;wifi 扫描可以参考 hfwifi_scan，AT 命令执行结果保存在 rsp 中，rsp 是一个字符串，具体格式请参考串口 AT 命令集帮助文档；通过这个函数可以获取设置系统配置。

注意这个函数放送不了通过 user_define_at_cmds_table 扩展的 AT 命令，因为自己扩展的 AT 命令可以直接调用，不需要在通过发送 AT 命令实现，如果用户通过 user_define_at_cmds_table 扩展了已经存在的 AT 命令例如”AT+VER “，

如果在程序中发送 hfat_send_cmd(“AT+VER\r\n”,sizeof(“AT+VER\r\n”),rsp,64);

返回的将是自带的 AT+VER 而不是自己扩展的。

头文件:

hfat.h

● hfat_enable_uart_session

函数原型:

int HSF_API hfat_enable_uart_session(char enable);

说明:

使能/关闭+++透传模式切换到命令模式;

参数:

enable：1-使能，0-关闭;

返回值:

HF_success: 设置成功，HF_FAIL: 执行失败;

备注:

无;

头文件:

hfat.h

4. DEBUG API

● HF_Debug

函数原型:

void HSF_API HF_Debug(int debug_level，const char *format，...);

说明:

输出调试信息到串口

参数:

Debug_level: 调试等级，可以为

#define DEBUG_LEVEL_LOW　1

#define DEBUG_LEVEL_MID　2

#define DEBUG_LEVEL_HI　　3

或者其他更大值，配合 hfdbg_set_level 设置的调式等级可以只输出设置的等级以上的 log 信息，log 信息输出需要先使能。

Format: 格式化输出，和 printf 一样，内容最多 250 字节，若内容超过此值，请调用多次进行打印。

返回值:

无

备注:

AT+NDBGL=X,Y 可使能 debug 信息输出，X 代表调试等级(0:关闭)，Y 代表串口号(0：串口 0，1:串口 1)，推荐调试信息输出到串口 1（串口 1 引脚请详见各模块手册），串口 0 用于正常交互通讯。程序发布后要动态打开调试，就可以用 AT+NDBGL 命令打开，不需要调试的时候用 AT+NDBGL=0 关闭。

头文件:

hfuart.h

● hfdbg_get_level

函数原型:

int HSF_API hfdbg_get_level(void);

说明:

获取当前设置的调试等级

参数:

无

返回值:

返回当前设置的调试等级

备注:

无

头文件:

hfuart.h

● hfdbg_set_level

函数原型：

void HSF_API hfdbg_set_level (int debug_level);

说明：

设置调试信息输出等级，或者关闭调试信息输出

参数：

debug_level: 调试级别，可以为

0：关闭 debug 信息输出

#define DEBUG_LEVEL_LOW 1

#define DEBUG_LEVEL_MID 2

#define DEBUG_LEVEL_HI 3

返回值：

无

备注：

推荐使用串口 AT+NDBGL 命令动态使能或关闭 debug 信息输出，这样需要查看 log 的时候可以随时查看，而不需要修改程序。

头文件：

hfuart.h

5. GPIO 控制 API

● hfgpio_configure_fpin

函数原型:

int HSF_API hfgpio_configure_fpin (int fid,int flag);

说明:

根据 fid(功能码)，配置对应的 PIN 脚

参数:

fid：功能码

```
enum HF_GPIO_FUNC_E
{
    //////fix//////////////////////
    HFGPIO_F_JTAG_TCK=0,
    HFGPIO_F_JTAG_TDO=1,
    HFGPIO_F_JTAG_TDI,
    HFGPIO_F_JTAG_TMS,
    HFGPIO_F_USBDP,
    HFGPIO_F_USBDM,
    HFGPIO_F_UART0_TX,
    HFGPIO_F_UART0_RTS,
    HFGPIO_F_UART0_RX,
    HFGPIO_F_UART0_CTS,
    HFGPIO_F_SPI_MISO,
    HFGPIO_F_SPI_CLK,
    HFGPIO_F_SPI_CS,
    HFGPIO_F_SPI_MOSI,
    HFGPIO_F_UART1_TX,
    HFGPIO_F_UART1_RTS,
    HFGPIO_F_UART1_RX,
    HFGPIO_F_UART1_CTS,
    ////////////////////////////////
    HFGPIO_F_NLINK,
    HFGPIO_F_NREADY,
    HFGPIO_F_NRELOAD,
    HFGPIO_F_SLEEP_RQ,
    HFGPIO_F_SLEEP_ON,
    HFGPIO_F_WPS,
    HFGPIO_F_IR,
    HFGPIO_F_RESERVE2,
    HFGPIO_F_RESERVE3,
    HFGPIO_F_RESERVE4,
    HFGPIO_F_RESERVE5,
    HFGPIO_F_USER_DEFINE
```

};

也可以为用户自定义功能吗，用户自定义功能码从 HFGPIO_F_USER_DEFINE 开始.

flags:PIN 脚属性，可以为下面一个或者多个值进行”|”运算

HFPIO_DEFAULT	默认
HFM_IO_TYPE_INPUT	输入模式
HFM_IO_OUTPUT_0	输出为低电平
HFM_IO_OUTPUT_1	输出为高电平

返回值:

HF_SUCCESS: 设置成功，HF_E_INVAL: fid 非法,或者它对应的 PIN 脚非法,

HF_E_ACCES: 对应的 PIN 不具备要设置的属性(flags)，例如 HFGPIO_F_JTAG_TCK 对应的 PIN 脚是一个外设 PIN 脚，不是 GPIO 脚，不能配置 HFPIO_DEFAULT 以外的任何属性.

备注:

在设置之前，先要清楚功能码对应的 PIN 脚的属性，每个 PIN 脚的属性请查看相关数据手册，如果给一个 PIN 配置它不具备的属性，将返回 HF_E_ACCES 错误。

头文件:

hfgpio.h

● hfgpio_fconfigure_get

函数原型:

int HSF_API hfgpio_fconfigure_get(int fid);

说明:

获取功能码对应的 PIN 脚对应的属性值;

参数:

fid: 功能码，参考 HF_GPIO_FUNC_E,也可以为用户自定义功能吗。

返回值:

成功返回 PIN 对应的属性值，属性值可以参考 hfgpio_configure_fpin，HF_E_INVAL: fid 非法,或者它对应的 PIN 脚非法

备注:

无

头文件:

hfgpio.h

● hfgpio_fpin_add_feature

函数原型:

int HSF_API hfgpio_fpin_add_feature(int fid,int flags);

说明:

对功能码对应的 PIN 脚添加属性值;

参数:

fid: 功能码，参考 HF_GPIO_FUNC_E,也可以为用户自定义功能吗;

flags: 参考 hfgpio_configure_fpin flags;

返回值:

HF_SUCCESS:设置成功， HF_E_INVAL: fid 非法,或者它对应的 PIN 脚非法

备注:

无

头文件:

hfgpio.h

● hfgpio_fpin_clear_feature

函数原型:

int HSF_API hfgpio_fpin_clear_feature (int fid,int flags);

说明:

清除功能码对应的 PIN 脚的一个或者多个属性值;

参数:

fid: 功能码，参考 HF_GPIO_FUNC_E,也可以为用户自定义功能吗;

flags: 参考 hfgpio_configure_fpin flags;

返回值:

HF_SUCCESS:设置成功， HF_E_INVAL: fid 非法,或者它对应的 PIN 脚非法

备注:

无

头文件:

hfgpio.h

● hfgpio_fpin_is_high

函数原型:

int HSF_API hfgpio_fpin_is_high(int fid);

说明:

判断功能码对应的 PIN 脚是否为高电平;

参数:

fid: 功能码，参考 HF_GPIO_FUNC_E,也可以为用户自定义功能吗,fid 对应的 PIN 脚一定具有 F_GPO 或者 F_GPI 属性。

返回值:

如果对应的 PIN 脚为低电平返回 0，如果为高电平返回 1;如果小于 0 说明 fid 对应的 PIN 脚非法.

备注:

无

头文件:

hfgpio.h

● hfgpio_fset_out_high

函数原型:

int HSF_API hfgpio_fset_out_high(int fid);

说明:

把功能码对应的 PIN 脚，设置为输出高电平

参数:

fid: 功能码，参考 HF_GPIO_FUNC_E，也可以为用户自定义功能吗。

返回值:

HF_SUCCESS:设置成功，HF_E_INVAL: fid 非法,或者它对应的 PIN 脚非法,

HF_FAIL: 设置失败，HF_E_ACCES: 对应的 PIN 属性不支持输出

备注:

这个函数等价于 hfgpio_configure_fpin(fid,HFM_IO_OUTPUT_1|HFPIO_DEFAULT);

头文件:

hfgpio.h

● hfgpio_fset_out_low

函数原型:

int HSF_API hfgpio_fset_out_low(int fid);

说明:

把功能码对应的 PIN 脚设置为输出低电平;

参数:

fid: 功能码，参考 HF_GPIO_FUNC_E,也可以为用户自定义功能码。

返回值:

HF_SUCCESS:设置成功，HF_E_INVAL: fid 非法,或者它对应的 PIN 脚非法

备注:

这个函数等价于 hfgpio_configure_fpin(fid,HFM_IO_OUTPUT_0|HFPIO_DEFAULT);

头文件:

hfgpio.h

● hfwifi_scan_ex

函数原型:

int HSF_API hfwifi_scan_ex(hfwifi_scan_callback_ex_t p_callback,void *ctx,unsigned char ch, unsigned char scan_time);

说明:

扫描附近的存在的 AP。

参数:

hfwifi_scan_callback_t: 设备扫描到周围的 AP 的时候，通过这个回调告诉用户这个 AP 的具体信息。

typedef int (*hfwifi_scan_callback_t)(PWIFI_SCAN_RESULT_ITEM);

typedef struct _WIFI_SCAN_RESULT_ITEM

{

```
    uint8_t auth; //认证方式
    uint8_t encry;//加密方式
    uint8_t channel;//工作信道
    uint8_t rssi;//信号强度
    char ssid[32+1];//AP 的 SSID
    uint8_t mac[6];//AP 的 mac 地址
    int rssi_dbm;//信号强度的 dBm 值
    int sco;
}WIFI_SCAN_RESULT_ITEM,*PWIFI_SCAN_RESULT_ITEM;
#define WSCAN_AUTH_OPEN 0
#define WSCAN_AUTH_SHARED 1
#define WSCAN_AUTH_WPAPSK 2
#define WSCAN_AUTH_WPA2PSK 3
#define WSCAN_AUTH_WPAPSKWPA2PSK 4
#define WSCAN_ENC_NONE 0
#define WSCAN_ENC_WEP 1
#define WSCAN_ENC_TKIP 2
#define WSCAN_ENC_AES 3
#define WSCAN_ENC_TKIPAES 4
```

ctx:回调函数的参数；

ch:要扫描的 Wi-Fi 信道

scan_time：扫描时间

返回值:

成功返回 HF_SUCCESS,否则失败。

备注:

扫描过程中函数不会退出，函数退出说明扫描结束。

头文件:

hfwifi.h

● hfwifi_sta_is_connected

函数原型:

Int HSF_API hfwifi_sta_is_connected(void);

说明:

判断 STA 模式下 WiFi 是否连接成功。

参数:

无

返回值:

如果成功连接 WiFi 返回 1，其他返回 0。

备注:

无

头文件:

hfnet.h

● hfwifi_transform_rssi

函数原型:

int HSF_API hfwifi_transform_rssi(int rssi_dbm);

说明:

将 dBm 信号格式转换为百分比形式。

参数:

rssi_dbm: 信号强度的 dBm 值，是负数

返回值:

信号的百分比强度;

备注:

转换公式约等于 rssi=(rssi_dbm+95)*2;

头文件:

hfwifi.h

● hfwifi_sta_get_current_rssi

函数原型:

int HSF_API hfwifi_sta_get_current_rssi(int *dBm);

说明:

获取当前连接路由器的信号强度;

参数:

dBm: 信号强度的 dBm 值，如果未连接值为-100

返回值:

信号强度的百分比值，如果未连接返回-1。

备注:

无

头文件:

hfwifi.h

● hfwifi_read_sta_mac_address

函数原型:

int HSF_API hfwifi_read_sta_mac_address(uint8_t *mac);

说明:

获取模块的 MAC 地址;

参数:

mac: 保存 MAC 地址;

返回值:

成功返回 HF_SUCCESS,否则失败。

备注：

获取的 MAC 格式为 6 个 8bit 的数字，如果获取到的值为 0xac,0xcf,0x88,0x88,0x88,0x88 则表示这个模块没有经过出厂校验，无法使用 WiFi 功能。

头文件：

hfsys.h

6. 串口 API

● hfuart_send

函数原型：

int HSF_API hfuart_send(hfuart_handle_t huart,char *data,uint32_t bytes, uint32_t timeouts);

说明：

发送数据到串口

参数：

huart: 串口设备对象，可选 HFUART0 或者 HFUART1（串口 0 或者串口 1）

data: 要发送的数据的缓存区

bytes: 发送数据的长度

timeouts: 超时时间，暂时无效值，默认填 0 即可

返回值：

成功返回为实际发送的数据，失败返回错误码；

备注：

无

头文件：

hfuart.h

7. 多任务 API

● hfthread_create

函数原型:

int HSF_API hfthread_create(PHFTHREAD_START_ROUTINE routine,const char * const name, uint16_t stack_depth, void *parameters, uint32_t uxpriority,hfthread_hande_t *created_thread, uint32_t *stack_buffer);

说明:

创建一个线程。

参数:

routine：函数指针类型，指向线程的启动函数

stack_depth: 线程堆栈深度，深度以 4Bytes 为一个单元，stack_size = stack_depth*4;

parameters: 传给线程入口函数的参数;

uxpriority: 线程优先级，HSF 线程优先级有：

HFTHREAD_PRIORITIES_LOW: 优先级低

HFTHREAD_PRIORITIES_MID：优先级一般

HFTHREAD_PRIORITIES_NORMAL: 优先级高

HFTHREAD_PRIORITIES_HIGH：优先级最高

用户线程一般使用 HFTHREAD_PRIORITIES_MID, HFTHREAD_PRIORITIES_LOW;

created_thread: 可选，函数执行成功，返回指向创建线程的指针;如果为空，不返回；

stack_buffer: 保留以后使用

返回值:

HF_SUCCESS: 成功，否则失败，请查看 HSF 错误码

备注:

为了稳定行，用户线程建议用 HFTHREAD_PRIORITIES_LOW 和 HFTHREAD_PRIORITIES_MID 两个优先级，最好不要使用 HFTHREAD_PRIORITIES_NORMAL 和它以上的优先级，除非线程大部分时间都在休眠，处理事件很少;

头文件:

hfthread.h

● hfthread_delay

函数原型:

void HSF_API hfthread_delay(uint32_t ms);

说明:

把当前线程暂停 ms 毫秒。

参数:

ms：指定要暂停的时间(单位为毫秒,最低 10ms);

返回值:

无

备注:

这个函数真正使线程休眠的时候可能会和实际时间有误差;

头文件:

hfthread.h

- **hfthread_destroy**

函数原型:

void HSF_API hfthread_destroy(hfthread_hande_t thread);

说明:

销毁由 hfthread_create 创建线程。

参数:

thread: 指向要销毁的线程,如果为 NULL,销毁当前线程;

返回值:

无

备注:

此函数用于销毁线程本身时资源不会立刻释放;

头文件:

hfthread.h

- **hfthread_mutext_new**

函数原型:

int HSF_API hfthread_mutext_new(hfthread_mutex_t *mutex);

说明:

创建一个线程互斥体。

参数:

mutex: 函数执行成功后，返回指向创建的互斥体;

返回值:

成功返回 HF_SUCCESS，HF_FAIL 表示失败

备注:

当不再使用创建的互斥体的时候，请使用 hfthread_mutext_free 释放资源;

头文件:

hfthread.h

- **hfthread_mutext_free**

函数原型:

void HSF_API hfthread_mutext_free(hfthread_mutex_t mutex);

说明:

销毁由 hfthread_mutext_new 创建的线程互斥体。

参数:

mutex: 指向要销毁的线程互斥体;

返回值:

无

备注:

无

头文件:

hfthread.h

- **hfthread_mutext_unlock**

函数原型:

void HSF_API hfthread_mutext_unlock(hfthread_mutex_t mutex);

说明:

释放线程互斥体。

参数:

mutex: 指向一个互斥体对象，由 hfthread_mutext_new 创建;

返回值:

无

备注:

无

头文件:

hfthread.h

- **hfthread_mutext_lock**

函数原型:

int HSF_API hfthread_mutext_lock (hfthread_mutex_t mutex);

说明:

获取线程互斥体。

参数:

mutex: 指向一个互斥体对象，由 hfthread_mutext_new 创建;

返回值:

成功返回 HF_SUCCESS，HF_FAIL 表示失败

备注:

hfthread_mutext_lock 和 hfthread_mutex_unlock 是成对出现的，如果调用的 hfthread_mutex_lock，没有调用 hfthread_mutex_unlock 再次调用 hfthread_mutex_lock 的时候就会发生死锁;

头文件:

hfthread.h

- **hfthread_mutext_trylock**

函数原型：

int HSF_API hfthread_mutext_trylock(hfthread_mutex_t mutex);

说明：

检查线程互斥体是否 lock。

参数：

mutex: 指向一个互斥体对象，由 hfthread_mutext_new 创建;

返回值：

如果返回 0mutext lock,否则 mutext 没有 lock

备注：

无

头文件：

hfthread.h

8. 网络 API

● hfnet_wifi_is_active

函数原型:

int HSF_API hfnet_wifi_is_active(void);

说明:

判断 WiFi 驱动是否初始化成功。

参数:

无

返回值:

成功返回 1，失败返回 0

备注:

STA 模式下当连接路由器成功后才会返回 1，STA 模式下只有当连接到路由器后才允许建立 socket 等后续网络通讯，未连接到路由器时，lwip 没有初始化，不允许创建网络 socket 等相关功能，也可以去掉这个判断，但进行网络通讯的时候必须等到 HFE_DHCP_OK 的系统事件后创建。

AP 模式下不影响，会直接跳过走后续流程。

头文件:

hfnet.h

● hfnet_start_uart

函数原型:

int HSF_API hfnet_start_uart(uint32_t uxpriority,hfnet_callback_t p_callback);

说明:

启动 UART0，并注册回调函数

参数:

uxpriority: uart 服务对应的线程的优先级;请参考 hfthread_create 参数 uxpriority

p_callback: 串口回调函数，可选，如果不需要请设置为 NULL,当串口收到数据的时候调用

返回值:

成功返回 HF_SUCCESS，HF_FAIL 表示失败

备注:

当串口接收数据的时候,如果 p_callback 不为 NULL,先调用 p_callback,如果工作在透传模式，把接收的数据发给 socketa,socketb 服务（如果这两个服务器存在），如果工作在命令模式把接收到的命令交给命令解析程序。

头文件:

hfnet.h

● 标准 socket API

HSF 采用 lwip 协议栈，兼容标准 socket 接口，例如 socket,recv,select,sendto,ioctl

等；如果源代码中使用标准 socket 函数，只需要导入头文件 hsf.h 和 hfnet.h 就可以了，标准 socket 的使用方法请参考相关手册。

UDP socket 接接收广播数据请使用 setsockopt 函数打开 SO_BROADCAST。

9. 系统函数

● hfmem_free

函数原型:

void HSF_API hfmem_free(void *pv);

说明:

释放由 hfsys_malloc 分配的内存

参数:

pv: 指向要释放内存地址;

返回值:

无

备注:

不要使用 libc 中的 free 函数.

头文件:

hfsys.h

● hfmem_malloc

函数原型:

void HSF_API *hfmem_malloc(size_t size)

说明:

动态分配内存

参数:

size: 分配内存的大小

返回值:

如果为 NULL,说明系统没有空闲的内存；成功返回内存的地址;

备注:

不要使用 libc 中的 malloc 函数

头文件:

hfsys.h

● hfsys_get_run_mode

函数原型:

int HSF_API hfsys_get_run_mode()

说明:

获取系统当前运行模式

参数:

无

返回值:

返回当前运行的模式,运行模式可以为下面的值:

```
enum HFSYS_RUN_MODE_E
{
   HFSYS_STATE_RUN_THROUGH=0,
   HFSYS_STATE_RUN_CMD=1,
   HFSYS_STATE_MAX_VALUE
};
```

头文件:

hfsys.h

● hfsys_get_time

函数原型:

uint32_t HSF_API hfsys_get_time (void);

说明:

获取系统从启动到现在所花的时间（毫秒）

参数:

无

返回值:

返回系统运行到现在所花的毫秒数

备注:

无

头文件:

hfsys.h

● hfsys_reset

函数原型:

void HSF_API hfsys_reset(void);

说明:

重启系统,IO 电平不保持

参数:

无

返回值:

无

备注:

无

头文件:

hfsys.h

● hfsys_switch_run_mode

函数原型:

int HSF_API hfsys_switch_run_mode(int mode);

说明:

切换系统运行模式

参数:

mode: 要切换的运行模式，系统当前支持的运行模式有

```
enum HFSYS_RUN_MODE_E
{
    HFSYS_STATE_RUN_THROUGH=0,
    HFSYS_STATE_RUN_CMD=1,
    HFSYS_STATE_MAX_VALUE
};
```

HFSYS_STATE_RUN_THROUGH：透传模式

HFSYS_STATE_RUN_CMD：命令模式

返回值:

HF_SUCCESS:成功，否则失败;

头文件:

hfsys.h

10. 用户 FLASH API

● hfuflash_size

函数原型:

```
int HSF_API hfuflash_size(void);
```

说明:

获取用户 flash 大小，单位字节

参数:

无

返回值:

返回用户 flash 大小，单位字节;

备注:

用户 flash 为物理 flash 的某一块特定的区域，用户只能通过 API 操作这一块区域，API 操作地址为用户 flash 的逻辑地址，我们不需要关心它的实际地址，不同型号模块对应的此区域大小可能不同。

头文件:

hfflash.h

● hfuflash_erase_page

函数原型:

```
int HSF_API hfuflash_erase_page(uint32_t addr, int pages);
```

说明:

擦写用户 flash 的页

参数:

addr: 用户 flash 逻辑地址,不是 flash 物理地址;

pages : 要擦除的 flash 页数;

返回值:

成功返回 HF_SUCCESS,失败返回 HF_FAIL;

备注:

用户 flash 为物理 flash 的某一块固定大小的区域,用户只能通过 API 操作这一块区域,API 操作地址为用户 flash 的逻辑地址，我们不需要关心它的实际地址。

头文件:

hfflash.h

- **hfuflash_read**

函数原型:

int HSF_API hfuflash_read(uint32_t addr, char *data, int len);

说明:

从用户文件中读数据

参数:

addr: 用户 flash 的逻辑地址(0- HFUFLASH_SIZE-2);

data : 从 flash 的数据的缓存区读取数据;

len : 缓存区的大小;

返回值:

小于零失败，否则返回实际从 flash 读到的 Bytes 数;

备注:

无

头文件:

hfflash.h

- **hfuflash_write**

函数原型:

int HSF_API hfuflash_write(uint32_t addr, char *data, int len);

说明:

向用户文件中写数据

参数:

addr: 用户 flash 的逻辑地址(0- HFUFLASH_SIZE-2);

data : 保存要写到 flash 中的数据的缓存区;

len : 缓存区的大小;

返回值:

如果小于零失败，否则返回实际写入到 flash 的 Bytes 数;

备注:

在对 flash 写之前，如果写的地址已经写入了数据，一定要先进行擦写动作。

头文件:

hfflash.h

11. OTA 升级 API

模块通过后文的 API 接口把固件下载到 OTA 升级备份区，如果校验文件正确，内部树标记位，下次重启的时候会把备份程序覆盖到 CODE 运行区并清除标记，完成固件更新动作，详细分区信息参见 SDK 用户文档。

● hfupdate_start

函数原型:

int HSF_API hfupdate_start(HFUPDATE_TYPE_E type);

说明:

开始升级，初始化 OTA 备份区域 Flash

参数:

type: 升级类型

```
typedef enum HFUPDATE_TYPE
{
   HFUPDATE_SW=0,//升级软件
}HFUPDATE_TYPE_E;
```

返回值:

成功返回 HF_SUCCESS,否则失败

备注:

当前只支持 HFUPDATE_SW，在开始下载升级文件之前先调用这个函数进行初始化，即擦除 OTA 升级备份区域。

头文件:

hfupdate.h

● hfupdate_write_file

函数原型:

int HSF_API hfupdate_write_file(HFUPDATE_TYPE_E type ,uint32_t offset,uint8_t *data,int len);

说明:

把升级文件数据写到升级区.

参数:

type: 升级类型，应为 HFUPDATE_SW

offset: 升级文件的偏移量，从 0 开始，实际地址是 OTA 备份区

data: 要写入的升级文件数据

len: 升级文件数据的长度,

返回值:

大于零成功，为实际写入的长度，否则失败。

备注:

无

头文件:

hfupdate.h

● hfupdate_complete

函数原型:

int HSF_API hfupdate_complete(HFUPDATE_TYPE_E type,uint32_t file_total_len);

说明:

用于升级文件数据下载完成后，进行更新操作

参数:

type: 升级类型，应为 HFUPDATE_SW

file_total_len: 升级文件的长度

返回值:

成功返回 HF_SUCCESS,否则失败

备注:

当升级文件全下载完成后调用这个函数检查文件合法性,如果合法,则写入升级标记位，下次重启的时候执行更新覆盖动作，如果非法，则不执行升级标记位写入动作。

头文件:

hfupdate.h

12. 加解密 API

● hfcrypto_aes_ecb_encrypt

函数原型:

int HSF_API hfcrypto_aes_ecb_encrypt(const unsigned char *key, unsigned char *data, int data_len);

说明:

AES ECB 模式（128Bit）加密；

参数:

key: 秘钥

data: 待加密数据，保存加密后的数据；

len: 加密数据长度，取 16 整数倍加密；

返回值:

加密后数据长度

备注:

无

头文件:

hfcrypto.h

● hfcrypto_aes_ecb_decrypt

函数原型:

int HSF_API hfcrypto_aes_ecb_decrypt(const unsigned char *key, unsigned char *data, int data_len);

说明:

AES ECB 模式（128Bit）解密；

参数:

key: 秘钥

data: 待解密数据，保存解密后的数据；

len: 解密数据长度，取 16 整数倍解密；

返回值:

解密后数据长度

备注:

无

头文件:

hfcrypto.h

● hfcrypto_aes_cbc_encrypt

函数原型:

int HSF_API hfcrypto_aes_cbc_encrypt(const unsigned char *key, const

unsigned char *iv, unsigned char *data, int data_len);

说明:

AES CBC 模式（128Bit）加密；

参数:

key: 秘钥

Iv: AES 的初始化向量

data: 待加密数据，保存加密后的数据；

len: 加密数据长度，取 16 整数倍加密；

返回值:

加密后数据长度

备注:

无

头文件:

hfcrypto.h

- **hfcrypto_aes_cbc_decrypt**

函数原型:

int HSF_API hfcrypto_aes_cbc_decrypt(const unsigned char *key, const unsigned char *iv, unsigned char *data, int data_len);

说明:

AES CBC 模式（128Bit）解密；

参数:

key: 秘钥

Iv: AES 的初始化向量

data: 待解密数据，保存解密后的数据；

len: 解密数据长度，取 16 整数倍解密；

返回值:

解密后数据长度

备注:

无

头文件:

hfcrypto.h

- **hfcrypto_md5**

函数原型:

int HSF_API hfcrypto_md5(const unsigned char *data, int len, unsigned char *digest);

说明:

MD5 值计算；

参数:

data: 待计算数据;

len: 待计算数据长度;

digest: 保存计算后的 MD5 值;

返回值:

MD5 值长度

备注:

无

头文件:

hfcrypto.h