API

HF-LPX30 API 参考手册

2018-02-07

目录

1. LINUX 标准 C 函数

HSF-LPX30 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。

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

说明:

获取 AT 命令或者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：word 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；

例子:

Example/attest.c

头文件:

hfat.h

● hfat_send_cmd

函数原型:

int 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。

例子:

参考 example 下的 attest.c

头文件:

hfat.h

● hfat_enable_uart_session

函数原型:

int 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;

例子:

无

头文件:

hfat.h

4. DEBUG API

● HF_Debug

函数原型:

void 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。

例子:

无

头文件:

hfdebug.h

● hfdbg_get_level

函数原型:

int hfdbg_get_level();

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

例子:

无

头文件:

hfdebug.h

● hfdbg_set_level

函数原型:

void 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。

例子:

无

头文件:

hfdebug.h

5. GPIO 控制 API

● hfgpio_configure_fpin

函数原型:

int 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。

例子:

gpiotest.c

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

例子:

gpiotest.c

头文件:

hfgpio.h

● hfgpio_fpin_add_feature

函数原型:

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

例子:

gpiotest.c

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

例子:

gpiotest.c

头文件:

hfgpio.h

● hfgpio_fpin_is_high

函数原型:

int 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

例子:

参考 example 下的 gpiotest.c

头文件:

hfgpio.h

● hfgpio_fset_out_high

函数原型:

int 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: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);

例子:

gpiotest.c

头文件:

hfgpio.h

● hfgpio_fset_out_low

函数原型:

int 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);

例子:

gpiotest.c

头文件:

hfgpio.h

● hfgpio_pwm_disable

函数原型:

int hfgpio_pwm_disable(int fid);

说明:

关闭 PWM 输出

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，HF_FAIL: 设置失败；HF_E_ACCES:对应的 PIN 脚没有 F_PWM 属性,不能配置成 PWM 模式

备注:

无

例子:

attest.c

头文件:

hfgpio.h

● hfgpio_pwm_enable

函数原型:

int hfgpio_pwm_enable(int fid, int freq, int hrate);

说明:

打开 PWM 输出

参数:

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

freq: 输出频率，范围 200-1000000，GPIO3 仅支持范围 2000-1000000

hrate: 输出占空比，范围 1-99

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 脚没有 F_PWM 属性,不能配置成 PWM 模式

备注:

无

例子:

attest.c

头文件:

hfgpio.h

● hfgpio_adc_enable

函数原型:

int hfgpio_adc_enable(int fid);

说明:

打开 ADC 功能

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，HF_FAIL:设置失败；HF_E_ACCES:对应的 PIN 脚没有 F_ADC 属性,不能配置成 ADC 模式

备注:

无

例子:

attest.c

头文件:

hfgpio.h

● hfgpio_adc_get_value

函数原型:

int hfgpio_adc_get_value(int fid);

说明:

读取 ADC 电平值

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:

ADC 电平，范围 0-1023

备注:

ADC为10位采样，在调用这个函数之前一定先调用hfgpio_adc_enable把ADC打开，采样值是相对应 2V 而言的，即 0V 的返回值为 0，2V 的返回值为 1023

例子:

attest.c

头文件:

hfgpio.h

6. WIFI API

● hfsmtlk_start

函数原型:

int HSF_API hfsmtlk_start(void);

说明:

启动 smartlink

参数:

无

返回值:

成功返回 HF_SUCCESS,否则失败

备注:

调用这个函数后程序马上软重启。

例子:

无

头文件:

hfsmtlk.h

● hfsmtlk_stop

函数原型:

int HSF_API hfsmtlk_stop(void);

说明:

停止 smartlink.

参数:

无

返回值:

成功返回 HF_SUCCESS,否则失败

备注:

无

例子:

无

头文件:

hfsmtlk.h

● hfwifi_scan

函数原型:

int HSF_API hfwifi_scan(hfwifi_scan_callback_t p_callback);

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

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。

例子:

wifitest.c

头文件:

hfwifi.h

● hfwifi_scan_ex

函数原型:

int HSF_API hfwifi_scan_ex(hfwifi_scan_callback_ex_t p_callback,void *ctx);

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；

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。

例子:

wifitest.c

头文件:

hfwifi.h

● hfwifi_sta_is_connected

函数原型:

Int 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。

例子：

wifitest.c

头文件：

hfwifi.h

● hfwifi_transform_rssi

函数原型：

int hfwifi_transform_rssi(int rssi_dbm);

说明：

将 dBm 信号格式转换为百分比形式。

参数：

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=(dBm+95)*2;

例子：

wifitest.c

头文件：

hfwifi.h

● hfwifi_sta_get_current_rssi

函数原型：

int 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；

例子:

wifitest.c

头文件:

hfwifi.h

● hfwifi_sta_get_current_bssid

函数原型:

int hfwifi_sta_get_current_bssid(uint8_t *bssid);

说明:

获取当前连接路由器的 BSSID。

参数:

bssid：保存连接路由器的 bssid；

返回值:

成功返回 HF_SUCCESS,未连接路由器返回-1。

备注:

无；

例子:

wifitest.c

头文件:

hfwifi.h

● hfwifi_read_sta_mac_address

函数原型：

int 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。

例子：

wifitest.c

头文件：

hfwifi.h

7. 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

例子：

无

头文件：

hfuart.h

8. 定时器 API

● hftimer_start

函数原型:

int HSF_API hftimer_start(hftimer_handle_t htimer);

说明:

启动一个定时器

参数:

htimer:由 hftimer_create 创建;

返回值:

成功返回 HF_SUCCESS,否则返回 HF_FAIL;

备注:

不允许在硬件中断中使用;

例子:

参考 example 下的 timertest.c

头文件:

hftimer.h

● hftimer_create

函数原型:

hftimer_handle_t HSF_API hftimer_create(const char *name, int32_t period, bool auto_reload, uint32_t timer_id, hf_timer_callback p_callback, uint32_t flags);

说明:

创建一个定时器

参数:

name: 定时器的名称

period:定时器触发的周期，以 ms 为单位;

auto_reload: 指定自动还是手动，如果为 true，只需要调用一次 hftimer_start 一次，定时器触发后,不需要再次调用 hftimer_start;如果为 false,触发后要再次触发要再次调用 hftimer_start.

timer_id: 指定一个唯一 ID，代表这个定时器，当多个定时器使用一个回调函数的时候可以用这个值来区分定时器;

flags: 当前可以为 0。

返回值:

函数执行成功，放回指向一个定时器对象的指针,否则返回 NULL;

备注:

定时器创建后，不会马上启动，直到调用 hftimer_start 定时器才会启动。如果制定定时器为手动，定时器触发后要想再次触发要重新调用 hftimer_start,如果是自动不需要，定时器会在下一个额周期自动触发；**注意：不支持在硬件中断中使用 hftimer 函数，hftimer 回调函数中不能使用 hfsniffer 相关函数。**

例子:

timertest.c

头文件:

hftimer.h

● hftimer_change_period

函数原型:

void HSF_API hftimer_change_period(hftimer_handle_t htimer,int32_t new_period);

说明:

修改定时器的周期

参数:

htimer:由 hftimer_create 创建;

new_period: 新的周期，单位 ms.如果创建的定时器为硬件定时器，单位为微秒

返回值:

无

备注:

修改定时器的周期，调用这个函数后，定时器将以新的周期运行；不允许在硬件中断中使用。

例子:

参考 example 下的 timertest.c

头文件:

hftimer.h

● hftimer_delete

函数原型:

void HSF_API hftimer_delete(hftimer_handle_t htimer);

说明:

销毁一个定时器

参数:

htimer:要删除的定时器，由 hftimer_create 创建;

返回值:

无

备注:

不允许在硬件中断中使用。

例子:

参考 example 下的 timertest.c

头文件:

hftimer.h

● hftimer_get_timer_id

函数原型:

uint32_t HSF_API hftimer_get_timer_id(hftimer_handle_t htimer);

说明:

获取定时器的 ID

参数:

htimer:由 hftimer_create 创建;

返回值:

成功返回定时器的 ID，由 hftimer_create 指定.失败返回 HF_FAIL;

备注:

这个函数一般在定时器回调的时候调用，又来区分多个 timer 使用一个回调函数的情况。

例子:

参考 example 下的 timertest.c

头文件:

hftimer.h

● hftimer_stop

函数原型:

void HSF_API hftimer_stop(hftimer_handle_t htimer);

说明:

停止一个定时器

参数:

htimer:由 hftimer_create 创建;

返回值:

无

备注:

调用这个函数后，定时器将不再触发，直到再次调用 hftimer_start；不允许在硬件中断中使用。

例子:

参考 example 下的 timertest.c

头文件:

hftimer.h

9. 多任务 API

● hfthread_create

函数原型：

int 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 ：输入参数：线程的入口函数,typedef void (*PHFTHREAD_START_ROUTINE)(void *);

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

parameters: 输入参数,传给线程入口函数的参数;

uxpriority: 输入参数，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;

例子：

参考 example 下的 threadtest.c

头文件：

hfthread.h

hfthread_delay

函数原型:

void hf_thread_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);

返回值:

没有返回值

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;

例子:

参考 example 下的 threadtest.c

头文件:

hfthread.h

hfthread_destroy

函数原型:

void 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:

没有返回值

备注:

此函数用于销毁线程本身时资源不会立刻释放;

例子:

参考 example 下的 threadtest.c

头文件:

hfthread.h

● hfthread_enable_softwatchdog

函数原型:

int HSF_API hfthread_enable_softwatchdog(hfthread_hande_t thread,uint32_t time);

说明:

使能线程的软件看门狗。

参数:

thread: 指向线程的指针,有 hfthread_create 返回,这个参数可以为 NULL,当为 NULL,使能当前线程的软件看门狗;

time:软件看门狗超时时间，单位秒;

返回值:

成功返回 HF_SUCCESS，HF_FAIL 表示失败

备注:

线程看门狗，可以检查线程卡死，如果看门狗使能，线程在规定的时间内没有调用 hfthread_reset_softwatchdog，那么模块会软复位。这个函数可以多次调用，可以动态修改超时时间，调用的时候系统会先把线程软件看门狗复位。线程看门狗默认为禁用，只有调用这个函数线程看门狗才起作用。

例子:

参考 example 下的 threadtest.c

头文件:

hfthread.h

● hfthread_disable_softwatchdog

函数原型:

int HSF_API hfthread_disable_softwatchdog(hfthread_hande_t thread);

说明:

禁用线程的软件看门狗。

参数:

thread: 指向线程的指针,有 hfthread_create 返回,这个参数可以为 NULL,当为 NULL,禁用当前线程的软件看门狗;

返回值:

成功返回 HF_SUCCESS，HF_FAIL 表示失败

备注:

在线程运行过程中如果某一个操作时间太长（或者等待某个信号量时间太长）大于超时时间，可以先禁用软件看门狗，防止因为操作时间太长而导致看门狗生效，导致模块重启，在操作完成后在使能看门狗;

例子:

参考 example 下的 threadtest.c

头文件:

hfthread.h

● hfthread_reset_softwatchdog

函数原型:

int HSF_API hfthread_reset_softwatchdog(hfthread_handle_t thread);

说明:

复位线程的软件看门狗(喂狗)。

参数:

thread: 指向线程的指针,有 hfthread_create 返回,这个参数可以为 NULL,当为 NULL,复位当前线程的软件看门狗;

返回值:

成功返回 HF_SUCCESS，HF_FAIL 表示失败

备注:

当看门狗使能后，线程一定要在规定的时间内调用这个函数，来做喂狗操作，当看门狗时间超时，模块会进行软重启操作;

例子:

参考 example 下的 threadtest.c

头文件:

hfthread.h

● hfthread_mutext_new

函数原型:

int 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;

例子:

参考 example 下的 threadtest.c

头文件:

hfthread.h

● hfthread_mutext_free

函数原型:

void 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:

没有返回值

备注:

;

例子:

参考 example 下的 threadtest.c

头文件:

hfthread.h

● hfthread_mutext_unlock

函数原型:

void 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:

没有返回值

备注:

;

例子:

参考 example 下的 threadtest.c

头文件:

hfthread.h

● hfthread_mutext_lock

函数原型:

int 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;

例子:

参考 example 下的 threadtest.c

头文件:

hfthread.h

● hfthread_mutext_trylock

函数原型:

int 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 mutext lock 返回 0,否则 mutext 没有 lock

备注:

;

例子:

参考 example 下的 threadtest.c

头文件:

hfthread.h

10. 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。

此函数和 LPB100 的用法完全不同。

例子:

头文件:

hfnet.h

● hfnet_start_uart

函数原型:

int hfnet_start_uart(uint32_t uxpriority,hfnet_callback_t p_uart_callback);

说明:

启动 HSF 自带 uart 串口收发控制服务。

参数:

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

p_uart_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,回调函数的定义和参数请参考 hfnet_start_socketa;

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_uart_callback 不为 NULL,先调用 p_uart_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。
在透传模式下，用户可以通过这个回调函数和 socketa,socketb 服务的回调，实现数据的加解密，或者二次处理；在命令模式下，用户可以通过回调实现自定义 AT 命令名称和格式;

例子:

参考 example 下的 callbacktest.c

头文件:

hfnet.h

● hfnet_start_socketa

函数原型:

int hfnet_start_socketa(uint32_t uxpriority,hfnet_callback_t p_callback);

说明:

启动 HSF 自带 socketa 服务。

参数:

uxpriority: socketa 服务优先级，请参考 hfthread_create 参数 uxpriority;

p_callback：回调函数，可选，如果不需要回调把这个值设置为 NULL,当 socketa 服务接收到数据包或者状态发送变化的时候触发;

int socketa_recv_callback_t(uint32_t event,void *data,uint32_t len,uint32_t buf_len);

event:事情 ID ；

data:指向接收数据的 buffer,用户可以在回调函数中修改 buffer 里面的值；当工作在 UDP 模式的时候 data+len 之后 6 个 bytes 放置的为发送端的 4Bytes ip 地址和 2 Bytes 端口号，如果是 socketa 工作在 TCP 服务器端模式，data+len 后面 4 个 Bytes 为客户端的 cid，可以通过 hfnet_socketa_get_client 或者详细信息。

len:接收到数据的长度;

buf_len:data 指向的 buffer 的实际长度,这个值大于等于 len;
回调函数返回值，为用户处理过数据的长度，如果用户不对数据进行修改，只是读，放回值应该等于 len;

返回值:

成功返回 HF_SUCCESS，HF_FAIL 表示失败

备注:

当 socketa 服务接收到网络发过来的数据的时候，调用 p_callback，再把 p_callback 处理的值发到串口，用户可以利用 p_callback 对接收的数据进行解析，或者二次处理，例如加密解密等,把处理的数据返回给 socketa 服务。

例子:

参考 example 下的 callbacktest.c

头文件:

hfnet.h

● hfnet_start_socketb

函数原型:

int hfnet_start_socketb(uint32_t uxpriority,hfnet_callback_t p_callback);

说明:

启动 HSF 自带 socketb 服务。

参数:

uxpriority:socketb 服务对应的线程的优先级;请参考 hfthread_create 参数 uxpriority
p_callback:可选，不使用回调传 NULL,请参考 hfnet_start_socketa

返回值:

成功返回 HF_SUCCESS，HF_FAIL 表示失败

备注:

无

例子：

callbacktest.c

头文件：

hfnet.h

● hfnet_ping

函数原型：

int hfnet_ping(const char* ip_address);

说明：

向目标地址发送 PING 包，检查 IP 地址是否可达。

参数：

ip_address:要检查的目标 IP 地址的字符串，地址格式为 xxx.xxx.xxx.xxx,如果要 ping 域名请先调用 hfnet_gethostbyname 来获取域名的 ip 地址;

返回值：

成功返回 HF_SUCCESS，HF_FAIL 表示失败。

备注：

如果网络不通，DNS 服务器设置错误或者要查询的都会导致失败。

例子：

头文件：

hfnet.h

● hfnet_gethostbyname

函数原型：

int hfnet_gethostbyname(const char *name, ip_addr_t *addr);

说明：

获取域名的 IP 地址。

参数：

name: 域名;
addr：IP 地址

返回值:

成功返回 HF_SUCCESS，HF_FAIL 表示失败。

备注:

例子:

头文件:

hfnet.h

● hfnet_start_httpd

函数原型:

int hfnet_start_httpd(uint32_t uxpriority);

说明:

启动 httpd,一个小型的 web server。

参数:

uxpriority：httpd 服务优先级，请参考 hfthread_create 参数 uxpriority;

返回值:

成功返回 HF_SUCCESS，HF_FAIL 表示失败。

备注:

如果应用程序需要支持网页接口，请在程序启动的时候调用这个函数。

例子:

头文件:

hfnet.h

● hfnet_httpd_set_get_nvram_callback

函数原型：

```
void HSF_API hfnet_httpd_set_get_nvram_callback(
        hfhttpd_nvset_callback_t p_set,
        hfhttpd_nvget_callback_t p_get);
```

说明：

设置 webserver 获取设置模块参数回调。

参数：

p_set：可选参数，如果不需要扩展 WEB 设置参数接口，请设置为 NULL,否则指向设置的入口函数；

设置回调函数的类型为：

int hfhttpd_nvset_callback(char * cfg_name,int name_len,char* value,int val_len);

其中 cfg_name 为对应的配置的名称，name_len 为 cfg_name 的长度，value 为配置对应的值，val_len 为 value 的长度；

p_get: 可选参数，如果不需要扩展 WEB 获取参数接口，请设置为 NULL,否则指向获取参数的入口函数；

读取参数的回调函数类型：

int hfhttpd_nvget_callback(char *cfg_name,int name_len,char *value,int val_len);

cfg_name 要读取参数的名称注意 cfg_name 不一定包含字符串结束符,,name_len:cfg_name 的长度,value:保存 cfg_name 对应配置的值,val_len:value 对应的长度；

返回值：

无

备注：

例子：

头文件：

hfnet.h

● hfnet_socketa_send

函数原型：

int hfnet_socketa_send(char *data,uint32_t len,uint32_t timeouts);

说明：

向 SOCKETA 发送数据。

参数：

data:保存发送数据的缓存区;

len:发送缓存区的长度;

timeouts:发送超时时间，当前不可用;

返回值：

成功返回实际发送数据的长度，否则返回错误码

备注：

。

例子：

头文件：

hfnet.h

● hfnet_socketb_send

函数原型：

int hfnet_socketb_send(char *data,uint32_t len,uint32_t timeouts);

说明：

向 SOCKETB 发送数据。

参数：

data:保存发送数据的缓存区;

len:发送缓存区的长度;

timeouts:发送超时时间，当前不可用;

返回值：

成功返回实际发送数据的长度，否则返回错误码。

备注:

例子:

头文件:

hfnet.h

● hfnet_socketa_fd

函数原型:

int hfnet_socketa_fd(void);

说明:

获取 socketa 的标准 socket 描述符。

参数:

;

返回值:

成功返回 socketa 标准 socket 的描述符.否则返回小于 0。

备注:

如果 socketa 工作在服务器模式，返回的是监听的 socket fd。

例子:

头文件:

hfnet.h

● hfnet_socketa_get_client

函数原型:

int hfnet_socketa_get_client(int cid,phfnet_socketa_client_t p_client);

说明:

获取 socketa 工作在 TCP 服务器模式的时候，连上的客服端信息。

参数:

cid: 客户 ID，可以为 0-4,当前 socketa 最多可以接 5 个用户;

p_client:不能为 NULL,指向客户信息结构的指针;

返回值:

成功返回 HF_SUCCESS,客户信息保存到 p_client 中,否则失败，如果 cid 对应的客户端不存在，返回失败。

备注:

这个函数只有 socketa 工作在 TCP 服务器模式的时候才有效。cid 由 socketa 的事件回调返回。

例子:

头文件:

hfnet.h

● hfnet_socketb_fd

函数原型:

int hfnet_socketb_fd(void);

说明:

获取 socketb 的标准 socket 描述符。

参数:

无。

返回值:

成功返回 socketb 标准 socket 的描述符.否则返回小于 0。

备注:

用户可以通过这个接口获取标准 socket 描述符，通过标准 lwip 函数来对 socketb 进行操作。

例子：

头文件：

hfnet.h

● hfnet_socketa_close_client_by_fd

函数原型：

int hfnet_socketa_close_client_by_fd(int sockfd);

说明：

通过 socket fd 关闭某个客户端。

参数：

sockfd：socket 描述符。

返回值：

成功返回 HF_SUCCESS，HF_FAIL 表示失败。

备注：

例子：

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。

11. 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.

例子:

无

头文件:

hfsys.h

● hfmem_malloc

函数原型:

void *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

● hfmem_realloc

函数原型:

void HSF_API *hfmem_realloc(void *pv,size_t size);

说明:

重新分配内存

参数:

pv:指向原先用 hfmem_malloc 分配地址的指针;

size:重新分配内存的大小

返回值:

无

备注:

请参考 libc 的 realloc，程序中不能直接调用 realloc 的函数，只能用这个 API。

例子:

无

头文件:

hfsys.h

● hfsys_get_memoryc

函数原型:

uint32_t HSF_API *hfsys_get_memory(void *pv,size_t siz);

说明:

重新分配内存

参数:

pv:指向原先用 hfmem_malloc 分配地址的指针;

size:重新分配内存的大小

返回值:

无

备注:

请参考 libc 的 realloc，程序中不能直接调用 realloc 的函数，只能用这个 API。

例子:

无

头文件:

hfsys.h

● hfsys_get_reset_reason

函数原型:

uint32_t HSF_API hfsys_get_reset_reason (void);

说明:

获取模块重启的原因

参数:

无

返回值:

返回模块重启的原因,可以是下面表中的一个或者多个（做或运算）

HFSYS_RESET_REASON_NORMAL	模块是由于断电再启动、硬件看门狗和外部 Reset 按键重启
HFSYS_RESET_REASON_IRESET0	模块是由于程序内部调用 hfsys_softreset 重启 (软件看门狗重启,或者程序段错误，内存访问错误)
HFSYS_RESET_REASON_IRESET1	模块是由于内部调用 hfsys_reset 重启
HFSYS_RESET_REASON_WPS	模块是由于 WPS 而重启
HFSYS_RESET_REASON_SMARTLINK_START	模块是由于 SmartLink 启动而重启

HFSYS_RESET_REASON_SMARTLINK_OK	模块是由于 SmartLink 配置成功而重启

备注：

一般在入口函数调用这个函数来判断一下，这次启动是重启，还是断电启动，以及重启的原因，根据不同的重启原因来进行恢复行的操作。

例子：

无

头文件：

hfsys.h

● hfsys_get_run_mode

函数原型：

int 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

例子:

无

头文件:

hfsys.h

● hfsys_nvm_read

函数原型:

int HSF_API hfsys_nvm_read(uint32_t nvm_addr, char* buf, uint32_t length);

说明:

从 NVM 里面读数据

参数:

nvm_addr:NVM 的地址，可以为(0-99);

buf: 保存从 NVM 读到数据的缓存区；

length: 长度和 nvm_addr 的和小于 100;

返回值:

成功返回 HF_SUCCESS，否则返回小于零.

备注:

当模块重启，软重启，NVM 的数据不会被清除，提供了 100Bytes 的 NVM，如果模块断电 NVM 的数据会被清除.

例子:

无

头文件:

hfsys,h

● hfsys_nvm_write

函数原型:

int HSF_API hfsys_nvm_write(uint32_t nvm_addr, char* buf, uint32_t length);

说明:

向 NVM 里面写数据

参数:

nvm_addr:NVM 的地址，可以为(0-99);

buf: 保存从 NVM 读到数据的缓存区;

length: 长度和 nvm_addr 的和小于 100;

返回值:

成功返回 HF_SUCCESS，否则返回小于零

备注:

当模块重启，软重启，NVM 的数据不会被清除，提供了 100Bytes 的 NVM，如果模块断电 NVM 的数据会被清除

例子:

无

头文件:

hfsys.h

● hfsys_register_system_event

函数原型:

int HSF_API hfsys_register_system_event(hfsys_event_callback_t p_callback);

说明:

注册系统事件回调

参数:

p_callback:指向用户制定的系统事情回调函数的地址;

返回值:

如果返回 HF_SUCCESS，系统按照默认动作处理这个事情， 否则返回小于零，这个时候系统不会对事情进行相应的处理

备注:

在回调函数中不能调用有延时的 API 函数，不能延时，处理后应该立刻返回，否则会影响系统正常运行。当前支持的系统事情有：

HFE_WIFI_STA_CONNECTED	当 STA 连接成功的时候触发
HFE_WIFI_STA_DISCONNECTED	当 STA 断开的时候触发
HFE_CONFIG_RELOAD	当系统执行 reload 的时候触发
HFE_DHCP_OK	当 STA 连接成功，并且 DHCP 拿到地址的时候触发
HFE_SMTLK_OK	当 SMTLK 配置拿到密码的时候触发，默认动作重启，如果回调返回不是 HF_SUCCESS，将不会重启，用户可以手动重启。

例子:

参考 example 下的 tcpclienttest.c

头文件:

hfsys.h

● hfsys_reload

函数原型:

void HSF_API hfsys_reload();

说明:

系统恢复成出厂设置

参数:

无

返回值:

无

备注:

无

例子:

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

例子:

无

头文件:

hfsys.h

● hfsys_softreset

函数原型:

void HSF_API hfsys_softreset(void);

说明:

软重启系统，IO 电平保持

参数:

无

返回值:

无

备注:

无

例子:

无

头文件:

hfsys.h

● hfsys_switch_run_mode

函数原型:

int 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

● hfconfig_file_data_read

函数原型:

int hfconfig_file_data_read(int offset, unsigned char *data, int len);

说明:

读取模块配置区参数;

参数:

offset: 配置区参数偏移量;

data: 保存读取到的参数;

len: 读取大小;

返回值:

成功返回 HF_SUCCESS,否则失败。

备注:

例子:

头文件:

hfconfig.h

● hfconfig_file_data_write

函数原型:

int hfconfig_file_data_write(int offset, unsigned char *data, int len);

说明:

设置模块配置区参数;

参数:

offset: 配置区参数偏移量;

data: 设置参数;

len: 设置大小;

返回值:

成功返回 HF_SUCCESS,否则失败。

备注:

例子:

头文件:

hfconfig.h

12. 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。

例子:

参考 example 下的 uflashtest.c

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。

例子:

参考 example 下的 uflashtest.c

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:

例子:

参考 example 下的 uflashtest.c

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。

data 地址不能是在程序区(ROM)，只能在 ram 不然调用这个函数会卡死或者程序会返回- HF_E_INVAL,下面代码是不允许的:

错误的写法 1：“ Test”放在 ROM 区；

```
hfuflash_write (Offset,"Test",4);
```

错误的写法 2：const 修饰的 初始化之后的变量放在程序区(ROM).

```
const uint8_t Data[] = "Test";
hfuflash_write (Offset,Offset,Data,4);
```

正确写法：

```
Uint8_t Data[]=” Test”；
hfuflash_write (Offset,Offset,Data,4);
```

例子：

参考 example 下的 uflashtest.c

头文件：

hfflash.h

13. 用户文件操作 API

● hffile_userbin_read

函数原型:

int HSF_API hffile_userbin_read(uint32_t offset,char *data,int len);

说明:

从用户文件中读数据

参数:

offset: 文件偏移量;

data：保存从文件读取到的数据的缓存区;

len：缓存区的大小;

返回值:

如果小于零失败，否则返回实际从文件读到的 Bytes 数;

例子:

无

头文件:

hffile.h

● hffile_userbin_size

函数原型:

int HSF_API hffile_userbin_size(void);

说明:

从用户文件读 bin 文件的大小。

参数:

无

返回值:

小于零失败，否则返回文件的大小;

备注:

无

例子：

无

头文件：

hffile.h

● hffile_userbin_write

函数原型：

int HSF_API hffile_userbin_write(uint32_t offset,char *data,int len);

说明：

把数据写入到用户文件

参数：

offset: 文件偏移量；

data：保存要写入到文件数据的缓存区；

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文件的 Bytes 数;

备注：

用户配置文件是一个固定大小的文件，文件保存在 flash 中，可以保存用户数据。用户配置文件有备份的功能，用户不需要当心在写的工程中断电，如果写的过程中断电，会自动恢复到写之前的内容。

例子：

无

头文件：

hffile.h

● hffile_userbin_zero

函数原型：

int HSF_API hffile_userbin_zero (void);

参数:

无

说明:

把整个文件的内容快速清零

返回值:

小于零失败，否则返回文件的大小;

备注:

调用这个函数能够非常快速的把整个文件内容清零；比通过 hffile_userbin_write 要快;

例子:

无

头文件:

hffile.h

14. 自动升级 API

● hfupdate_complete

函数原型:

int hfupdate_complete(HFUPDATE_TYPE_E type,uint32_t file_total_len);

说明:

升级完成

参数:

type:升级类型

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来执行升级动作。

例子:

参考 example 下的 updatetest.c

头文件:

hfupdate.h

● hfupdate_start

函数原型:

int hfupdate_start(HFUPDATE_TYPE_E type);

说明:

开始升级.

参数:

type:升级类型

typedef enum HFUPDATE_TYPE

{

HFUPDATE_SW=0,//升级软件

HFUPDATE_CONFIG=1,//升级默认配置，暂不支持
HFUPDATE_WIFIFW,//升级 WIFI 固件
HFUPDATE_WEB,//升级 web，暂不支持
}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。

例子:

参考 example 下的 updatetest.c

头文件:

hfupdate.h

● hfupdate_write_file

函数原型:

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

说明:

把升级文件数据写到升级区.

参数:

type:升级类型
offset:升级文件的偏移量
data:要写入的升级文件数据
len:升级文件数据的长度

返回值:

大于等于零成功，时间写入的长度，否则失败。

备注:

当前只支持 HFUPDATE_SW.

例子:

参考 example 下的 updatetest.c

头文件:

hfupdate.h

15. 加解密 API

● hfcrypto_aes_ecb_encrypt

函数原型:

int 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:

例子:

头文件:

hfcrypto.h

● hfcrypto_aes_ecb_decrypt

函数原型:

int 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：

例子：

头文件：

hfcrypto.h

● hfcrypto_aes_cbc_encrypt

函数原型：

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

说明：

AES CBC 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：

例子：

头文件:

hfcrypto.h

● hfcrypto_aes_cbc_decrypt

函数原型:

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

说明:

AES CBC 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:

例子:

头文件:

hfcrypto.h

● hfcrypto_md5

函数原型:

int 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：

例子：

头文件：

hfcrypto.h