High-Flying Software Framework (HSF)
API 参考手册
V1.4x

版本 1.4x
2015 年 5 月

更新记录：

修改时间	作者	修改	版本
2013.8.26	Jim	初稿	V1.0
2013.9.13	Jim	增加定时器 API,和串口接收 API	V1.03
2013.10.16	Jim	1， 增加 用户文件操作接口 2， 增加 hfsys_get_time API 函数 3， 整理部分说明 4， 更新 hfgpio_configure_fpin_interrupt	V1.13
2013.10.28	Jim	1,添加 hfnet_httpd_set_get_nvram_callback websever 回调函数 2,添加 wps 功能,通过 AT+WPS 进入 wps.	V1.15
2013.11.19	Jim	1,支持 SmartLink 功能 2，添加 hfuflash_xx 接口； 3 , 支持 2MB flash	V1.16
2013.11.24	Jim	1,添加 nvm 接口。 2，添加 pwm 接口 3,添加线程软件看门狗接口 4,添加 hftimer_change_period 接口 5,添加 hfsys_get_reset_reason 函数	V1.17
2013.12.03	Jim	1， 定时器支持硬件定时器; 2， 解决 udp 无法接收广播包 Bug.	V1.17
2013.12.05	Jim	1,添加获取定时器计数器接口	V1.17
2014.01.07	Jim	1， 添加 ADC 接口 2， 添加 hfmem_realloc 接口 3， 添加 socketa,socketb 获取详细详细接口	V1.17
2014.01.11	Cyrus	1,增加 httpd_callback_register，httpd_callback_cancel 接口	V1.17
2014.02.18	Jim	1,增加支持 UART1,完善串口 API 函数; 2,添加 HFSYS_RESET_REASON_WPS_OK 重启原因	V1.17

		3,优化 hfuflash_write 函数，当传入保存数据的 buffer 在 ROM 的时候返回- HF_E_INVAL;	
2014.05.20	Ke	1,添加 hfnet_socketa_close_client_by_fd，通过 socket fd 关闭某个客户端 2,添加 smartlink 抓包函数，hfsmtlk_register，hfsmtlk_set_filter，hfsmtlk_finished_ok	V1.40

目录

1 结构定义

1.1 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*/
};
```

2 API 函数说明

2.1 libc 函数

HSF 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.

注：在 Keil MDK 系统中不能直接调用 libc 中的内存管理函数，否则链接将不通过，内存管理函数当前只提供 3 个函数，参考 hfmem_malloc,hfmem_free,hfmem_realloc.

2.2 系统函数

2.2.1 hfsys_switch_run_mode

切换系统运行模式。

```
int hfsys_switch_run_mode(int mode);
```

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_RUN_GPIO,
        HFSYS_STATE_RUN_PWM,
        HFSYS_STATE_MAX_VALUE
};
```

HFSYS_STATE_RUN_THROUGH：透传模式
HFSYS_STATE_RUN_CMD：命令模式
HFSYS_STATE_RUN_GPIO:GPIO 模式

返回值：

HF_SUCCESS:成功，否则失败，请查看 HSF 错误码;

要求：

所在头文件：hfsys.h
所在库：libKernel.a
HSF 版本要求：V1.0 以上
硬件：LPBXX

2.2.2 hfsys_get_run_mode

获取系统当前运行模式

```
int hfsys_get_run_mode();
```

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_RUN_GPIO,
	HFSYS_STATE_RUN_PWM,
	HFSYS_STATE_MAX_VALUE
};
```

备注：

例子：

要求：

所在头文件：hfsys.h
所在库：libKernel.a
HSF 版本要求：V1.0 以上
硬件：LPBXX

2.2.3 hfmem_malloc

动态分配内存

```
void  *hfmem_malloc(size_t size);
```

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：

这个函数是线程安全的，如果开发多线程应用这个函数，而不要使用 libc 中的 malloc，它不是线程安全函数,在 LPB100 系列中直接调用 libc 中的内存管理函数程序链接不成功。

例子：

要求：

所在头文件：hfsys.h
所在库：libKernel.a
HSF 版本要求：V1.0 以上
硬件：LPBXX

2.2.4 hfmem_free

释放由 hfsys_malloc 分配的内存
void HSF_API hfmem_free(void *pv)；

参数：

pv:指向要释放内存地址;

返回值：

无

备注：

这个函数是线程安全的，如果开发多线程应用这个函数，而不要使用 libc 中的 free，它不是线程安全函数。

例子：

无

要求：

所在头文件：hfsys.h
所在库：libKernel.a
HSF 版本要求：V1.0 以上
硬件：LPBXX

2.2.5 hfmem_ realloc

重新分配内存
void HSF_API *hfmem_realloc(void *pv,size_t size)；

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

要求：

所在头文件：hfsys.h
所在库：libKernel.a
HSF 版本要求：V1.17 以上
硬件：LPBXX

2.2.6 hfsys_reset

重启系统,IO 电平不保持
void HSF_API hfsys_reset(void);

参数：

无

返回值：

无

备注：

无

例子：

无

要求：

所在头文件：hfsys.h
所在库：libKernel.a
HSF 版本要求：V1.0 以上
硬件：LPBXX

2.2.7 hfsys_softreset

软重启系统，IO 电平保持
void HSF_API hfsys_softreset(void);

参数：

无

返回值：

无

备注：

无

例子：

无

要求：

所在头文件：hfsys.h
所在库：libKernel.a
HSF 版本要求：V1.17 以上

硬件：LPBXX

2.2.8 hfsys_reload

系统恢复成出厂设置
void HSF_API hfsys_reload()；

参数：

无

返回值：

无

备注：

调用这个函数之后，建议马上调用 hfsys_reset 重启系统;

例子：

无

要求：

所在头文件：hfsys.h
所在库：libKernel.a
HSF 版本要求：V1.0 以上
硬件：LPBXX

2.2.9 hfsys_get_time

获取系统从启动到现在所花的时间（毫秒）

uint32_t HSF_API hfsys_get_time (void);

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

要求：

所在头文件：hfsys.h
所在库：libKernel.a
HSF 版本要求：V1.0 以上
硬件：LPBXX

2.2.10 hfsys_nvm_read

从 NVM 里面读数据

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

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，LPB 提供了 100Bytes 的 NVM，如果模块断电 NVM 的数据会被清除.

例子：

无

要求：

所在头文件：hfsys.h
所在库：libKernel.a
HSF 版本要求：V1.17 以上
硬件：LPBXX

2.2.11 hfsys_nvm_write

向 NVM 里面写数据

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

参数：

nvm_addr:NVM 的地址，可以为(0-99);
buf: 保存写入到 NVM 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，LPB 提供了 100Bytes 的 NVM，如果模块断电 NVM 的数据会被清除.

例子：

无

要求：

所在头文件：hfsys.h
所在库：libKernel.a
HSF 版本要求：V1.17 以上
硬件：LPBXX

2.2.12 hfsys_get_reset_reason

获取模块重启的原因

uint32_t HSF_API hfsys_get_reset_reason (void);

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
HFSYS_RESET_REASON_ERESET	模块是由于硬件看门狗和外部 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
HFSYS_RESET_REASON_WPS_OK	模块由于 WPS 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：

参考 example 中的 callbacktest

要求：

所在头文件：hfsys.h

所在库：libKernel.a
HSF 版本要求：V1.17 以上
硬件：LPBXX

2.2.13 hfsys_register_system_event

注册系统事件回调

```
int HSF_API hfsys_register_system_event(
                                    hfsys_event_callback_t p_callback
                                    );
```

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：

无

要求：

所在头文件：hfsys.h
所在库：libKernel.a
HSF 版本要求：V1.17 以上
硬件：LPBXX

2.3 定时器 API

LPB 软件定时器精度为 1ms,LPB100 软件定时器精度为 10ms，如果需要比较严格的定时器，请使用硬件定时器，硬件定时器为精度为微秒 us。定时器里面不能做有延时的长时间的操作，不能执行函数里面本来用到 timer 的 API,否则会卡死 timer。

2.3.1 hftimer_create

创建一个定时器

```
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 );
```

参数：

name: 定时器的名称

period:定时器触发的周期，以 ms 为单位;
如果 flags 设置为 HFTIMER_FLAG_HARDWARE_TIMER 单位为μ s 微秒.

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 或者 HFTIMER_FLAG_HARDWARE_TIMER,如果要创建的定时器是硬件定时器，请把 flags 设置为 HFTIMER_FLAG_HARDWARE_TIMER。

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.

如果要创建硬件定时器,flags 制定 HFTIMER_FLAG_HARDWARE_TIMER,只能创建一个硬件定时器。当为硬件定时器的时候，周期的单位为微秒，由于硬件原因定时器可能不是很准确，要根据时间情况微调(大概在%1-%2 的误差)。硬件定时器只有在 V1.17 和之后的版本才能够支持.

例子：

参考 example/time

要求：

所在头文件：hftimer.h
所在库：libKernel.a
HSF 版本要求：V1.03 以上
硬件：LPBXX

2.3.2 hftimer_delete

销毁一个定时器
void HSF_API hftimer_delete(hftimer_handle_t htimer);

参数：

htimer:要删除的定时器，由 hftimer_create 创建；

返回值：

无

例子：

要求：

所在头文件：hftimer.h
所在库：libKernel.a
HSF 版本：HSF V1.03 以上

2.3.3 hftimer_start

启动一个定时器

int HSF_API hftimer_start(hftimer_handle_t htimer);

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：

例子：

参考 hftimer_create

要求：

所在头文件：hftimer.h
所在库：libKernel.a
HSF 版本：HSF V1.03 以上

2.3.4 hftimer_stop

停止一个定时器

void HSF_API hftimer_stop(hftimer_handle_t htimer);

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;

例子：

要求：

所在头文件：hftimer.h
所在库：libKernel.a
HSF 版本：HSF V1.03 以上

2.3.5 hftimer_get_timer_id

获取定时器的 ID
uint32_t HSF_API hftimer_get_timer_id(hftimer_handle_t htimer);

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 hftimer_create

要求：

所在头文件：hftimer.h
所在库：libKernel.a
HSF 版本：HSF V1.03 以上

2.3.6 hftimer_change_period

修改定时器的周期

```
void HSF_API hftimer_change_period(
                                hftimer_handle_t htimer,
                                int32_t new_period
                                );
```

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.

例子：

参考 example timer

要求：

所在头文件：hftimer.h
所在库：libKernel.a
HSF 版本：HSF V1.17 以上

2.3.7 hftimer_get_counter

获取硬件定时器从启动到现在的 CLK 计数

```
void HSF_API hftimer_get_counter (hftimer_handle_t htimer);
```

参数：

htimer:指向由 hftimer_create 创建的硬件定时器.

返回值：

返回 定时器从启动到当前 CLK 计数，LPB100 当前的频率为 48MHZ,一个

CLK 为 1/48 us,定时器从启动到现在经历的时间为 counter/48 us.如果返回值为 0 说明定时器的时间到了。

备注：

如果程序里面需要比较精确的时间，可以用这个函数加硬件定时器来实现.

例子：

参考 example timer

要求：

所在头文件：hftimer.h
所在库：libKernel.a
HSF 版本：HSF V1.17 以上

2.4 多线程 API

2.4.1 hfthread_create

```
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：

```
#include <hsf.h>

//线程入口函数
void test_thread_func(void *arg)
{
        while(1)
        {
                msleep(1000);//线程休眠 1s
                HF_debug(DEBUG_LEVEL,"thread is running\n");
        }
}

int app_main(void)
{
        If(hfthread_create(test_thread_func,"TEST_THREAD",256,NULL,
HFTHREAD_PRIORITIES_LOW,NULL,NULL)!=HF_SUCCESS)
        {
                HF_debug(DEBUG_LEVEL,"create thread fail\n");
                return 0;
        }

        return 0;
}
```

要求：

所在头文件：hfthread.h
所在库：libKernel.a
HSF 版本要求：V1.0 以上
硬件：LPBXX

2.4.2 hfthread_delay

把当前线程暂停 ms 毫秒。
void hf_thread_delay(uint32_t ms);

参数：

ms ,指定要暂停的时间(单位为毫秒);

返回值：

这个函数没有返回值

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，如果要求 sleep 比较准确的时间，请使用 **hfthread_delay** (hftimer_get_timer_adjust_factor()*ms), msleep 函数也用有这个限制。

要求：

所在头文件：hfthread.h
所在库：libKernel.a
HSF 版本：HSF V1.0 以上

2.4.3 hfthread_destroy

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：

函数没有返回值

要求：

所在头文件：hfthread.h
所在库：libKernel.a
HSF 版本：HSF V1.0 以上

2.4.4 hfthread_enable_softwatchdog

使能线程的软件看门狗。

```
int HSF_API hfthread_enable_softwatchdog(
                                    hfthread_hande_t thread,
                                    uint32_t time
                                    );
```

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：

HF_SUCCESS:成功，否则失败，请查看 HSF 错误码;

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 LPB 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 thread.

要求：

所在头文件：hfthread.h
所在库：libKernel.a
HSF 版本：HSF V1.7 以上

2.4.5 hfthread_disable_softwatchdog

禁用线程的软件看门狗。

```
int HSF_API hfthread_disable_softwatchdog(
                                    hfthread_hande_t thread,
                                    );
```

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：

HF_SUCCESS:成功，否则失败，请查看 HSF 错误码;

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 thread.

要求：

所在头文件：hfthread.h
所在库：libKernel.a
HSF 版本：HSF V1.7 以上

2.4.6 hfthread_reset_softwatchdog

复位线程的软件看门狗(喂狗)。

```
int HSF_API hfthread_disable_softwatchdog(
                                    hfthread_hande_t thread,
                                    );
```

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：

HF_SUCCESS:成功，否则失败，请查看 HSF 错误码;

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 thread.

要求：

所在头文件：hfthread.h
所在库：libKernel.a
HSF 版本：HSF V1.7 以上

2.4.7 hfthread_mutext_new

```
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

HF_SUCCESS:成功，否则失败，请查看 HSF 错误码;

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：

要求：

所在头文件：hfthread.h
所在库：libKernel.a
HSF 版本：HSF V1.0 以上

2.4.8 hfthread_mutext_free

void hfthread_mutext_free(hfthread_mutex_t mutex);

说明：销毁由 hfthread_mutext_new 创建的线程；

参数：

mutex:指向要销毁的互斥体;

返回值：

函数没有返回值;

例子：

参考 hfthread_create

要求：

所在头文件：hfthread.h
所在库：libKernel.a
HSF 版本：HSF V1.0 以上

2.4.9 hfthread_mutext_unlock

释放互斥体;

void hfthread_mutext_unlock(hfthread_mutex_t mutex);

参数：

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

返回值：

函数没有返回值；

例子：

参考 hfthread_create

要求：

所在头文件：hfthread.h
所在库：libKernel.a
HSF 版本：HSF V1.0 以上

2.4.10 hfthread_mutext_lock

int hfthread_mutext_lock (hfthread_mutex_t mutex);

参数：

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

返回值：

HF_SUCCESS 成功； HF_FAIL 可能发生死锁，其它请参考 HSF 错误码定义

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：

要求：

所在头文件：hfthread.h
所在库：libKernel.a
HSF 版本：HSF V1.0 以上

2.4.11 hfthread_mutext_trylock

检查 mutext 是否 lock.

int HSF_API hfthread_mutext_trylock(hfthread_mutex_t mutex)

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.

注：

例子：

要求：

所在头文件：hfthread.h
所在库：libKernel.a
HSF 版本：HSF V1.17 以上

2.5 网络 API

2.5.1 hfnet_ping

向目标地址发送 PING 包，检查 IP 地址是否可达。

int hfnet_ping(const char* ip_address);

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，否则失败,具体失败原因请参考 HSF 错误码

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：

要求：

所在头文件：hfnet.h
所在库：libKernel.a
HSF 版本：HSF V1.0 以上

2.5.1 hfnet_gethostbyname

获取域名的 IP 地址。

参数：

返回值：

成功返回 HF_SUCCESS，HF_FAIL 表示失败

备注：

例子：

要求：

所在头文件：hfnet.h
所在库：libKernel.a
HSF 版本：HSF V1.0 以上

2.5.1 hfnet_start_httpd

启动 httpd,一个小型的 web server。

int hfnet_start_httpd(uint32_t uxpriority);

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：

要求：

所在头文件：hfnet.h
所在库：libKernel.a
HSF 版本：HSF V1.0 以上

2.5.2 hfnet_httpd_set_get_nvram_callback

设置 webserver 获取设置模块参数回调。

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

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：

参考 SDK example file .

要求：

所在头文件：hfnet.h

所在库：libKernel.a
HSF 版本：HSF V1.15 以上

2.5.3 hfnet_start_socketa

启动 HSF 自带 socketa 服务。
int hfnet_start_socketa(uint32_t uxpriority,hfnet_callback_t p_callback);

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：

下面例子实现当 socktea(工作在 TCP 服务器模式)服务接收到网络发送过来的数据的时候，把接收的长度加入到 buffer 的最后 2 个 Bytes;

```
int socketa_recv_callback( uint32_t event,void *data,uint32_t len,uint32_t buf_len)
{
	uint32_t cid;
	hfnet_socketa_client_t client;
	uint8_t *p_data=(uint8_t*)data+len;
	cid = p_data[0]|p_data[1]<<8|p_data[2]<<16|p_data[3]<<24;
	hfnet_socketa_get_client(cid,& client);
	u_printf("recv socketa event= %d fd= %d\n",event,client.fd);

	if(event== HFNET_SOCKETA_DATA_READY)
```

```
        {
        If(buf_len>len+2)
                return len;
        data[len]=len&0xFF;
        data[len+1]=(len&0x00FF)>>8;
        return len+2;
        }
        return len;
}
int USER_FUNC app_main (void)
{
        if(hfnet_start_socketa(HFTHREAD_PRIORITIES_LOW,
                                (hfnet_callback_t)socketa_recv_callback)!=HF_SUCCESS)
        {
                HF_Debug(DEBUG_WARN,"start socketb fail\n");
        }
        ……
}
```

要求：

所在头文件：hfnet.h
所在库：libKernel.a
HSF 版本：HSF V1.0 以上

2.5.4 hfnet_start_socketb

启动 HSF 自带 socketb 服务。

```
int hfnet_start_socketb(uint32_t uxpriority,hfnet_callback_t p_callback);
```

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：

例子：

请参考 hfnet_start_socketa；

要求：

所在头文件：hfnet.h
所在库：libKernel.a
HSF 版本：HSF V1.0 以上

2.5.5 hfnet_start_uart

启动 HSF 自带 uart 串口收发控制服务。

int hfnet_start_uart(uint32_t uxpriority,hfnet_callback_t p_uart_callback);

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_socketb;

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：

要求：

所在头文件：hfnet.h
所在库：libKernel.a
HSF 版本：HSF V1.0 以上

2.5.6 hfnet_socketa_send

向 SOCKETA 发送数据

```
int  HSF_API hfnet_socketa_send(
                        char *data,
                        uint32_t len,
                        uint32_t timeouts)
```

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：

无

要求：

所在头文件：hfnet.h
所在库：libKernel.a
HSF 版本：HSF V1.03 以上

2.5.7 hfnet_socketb_send

向 SOCKETB 发送数据

```
int  HSF_API hfnet_socketb_send(
                    char *data,
                    uint32_t len,
                    uint32_t timeouts)
```

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：

无

要求：

所在头文件：hfnet.h
所在库：libKernel.a
HSF 版本：HSF V1.03 以上

2.5.8 hfnet_set_udp_broadcast_port_valid

设置 UDP 可以接收广播的端口范围;

```
int  HSF_API hfnet_set_udp_broadcast_port_valid (
                    uint16_t start_port,
                    uint16_t end_port)
```

参数：

start_port:开始端口号;
end_port:结束端口号;

返回值：

成功返回 HF_SUCCESS,否则返回-HF_E_INVAL;

备注：

LPB100 中默认会过滤掉网络中的广播包来减清系统负担，因此如果用户创建的 socket 需要接收广播包的话，要通过这个函数设置你要监听端口范围.

例子：

参考 example 中的 threadtest

要求：

所在头文件：hfnet.h
所在库：libKernel.a
HSF 版本：HSF V1.17 以上

2.5.9 hfnet_socketa_fd

获取 socketa 的标准 socket 描述符;

```
int HSF_API hfnet_socketa_fd(void);
```

参数：

无

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：

参考 example 中的 netcallback

要求：

所在头文件：hfnet.h
所在库：libKernel.a
HSF 版本：HSF V1.17 以上

2.5.10 hfnet_socketa_get_client

获取 socketa 工作在 TCP 服务器模式的时候，连上的客服端信息;

```
int HSF_API hfnet_socketa_get_client(int cid,phfnet_socketa_client_t p_client);
```

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 的事情回调返回。

例子：

参考 example 中的 netcallback

要求：

所在头文件：hfnet.h
所在库：libKernel.a
HSF 版本：HSF V1.17 以上

2.5.11 hfnet_socketb_fd

获取 socketb 的标准 socket 描述符;

```
int HSF_API hfnet_socketb_fd(void);
```

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：

参考 example 中的 netcallback

要求：

所在头文件：hfnet.h
所在库：libKernel.a
HSF 版本：HSF V1.17 以上

2.5.12 hfhttpd_url_callback_register

设置基于 webserver url 的数据处理回调。

```
int HSF_API  hfhttpd_url_callback_register(
                                hfhttpd_url_callback_t callback,
                                int flag
```

);

参数：

callback:用户参数解析回调函数；

设置回调函数的类型为：

int hfhttpd_url_callback (char *url, char *rsp);

其中 url 为去掉 ip 地址的 url，rsp 为需要返回的数据，最大支持 1400 byte；回调函数返回-1 则 webserver 处理；返回 0，webserver 将不解析此次 http 请求,回调函数自己处理这个 http 请求。

flag：0 为无需认证，1 为需要认证；

返回值：

成功返回 HF_SUCCESS，HF_FAIL 表示失败；

备注：

经过初步解析的 url 为去掉 ip 地址的 url；如果浏览器输入：10.10.100.254/abcd，则回调函数得到的 url 为/abcd。

Example：

参考 example 中的 url_callback

要求：

所在头文件:hfnet.h

所在库：libKernel.a

HSF 版本：HSF V1.17 以上

2.5.13 hfhttpd_url_callback_cancel

注销 url 处理回调函数

int HSF_API hfhttpd_url_callback_cancel(void)；

参数：

无

返回值：

成功返回 HF_SUCCESS，HF_FAIL 表示失败；

备注：

无

Example：

无

要求：

所在头文件:hfnet.h
所在库：libKernel.a
HSF 版本：HSF V1.17 以上

2.5.14 hfnet_socketa_close_client_by_fd

通过 socket fd 关闭某个客户端

int HSF_API hfnet_socketa_close_client_by_fd(int sockfd)

参数：

无

返回值：

成功返回 0，-1 表示失败；

备注：

无

Example：

无

要求：

所在头文件:hfnet.h
所在库：libKernel.a
HSF 版本：HSF V1.40 以上

2.5.15 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.

注：由于系统限制，在使用 lwip 建立 socket 的时候，建立 socket 和接收数据的一定要在同一个线程,发送不需要在同一个线程.不然将接收不到数据. 对应 UDP 要接收广播包，为了性能我们模块对广播包进行过滤，如果要接收广播包请参考 hfnet_set_udp_broadcast_port_valid 函数。

2.6 GPIO 控制 API

2.6.1 hfgpio_configure_fpin

根据 fid(功能码)，配置对应的 PIN 脚

int hfgpio_configure_fpin(int fid,int flags);

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_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

HFPIO_PULLUP ：内部上拉
HFPIO_PULLDOWN：内部下拉
HFPIO_IT_LOW_LEVEL :中断低电平触发
HFPIO_IT_HIGH_LEVEL:中断高电平触发
HFPIO_IT_FALL_EDGE :中断下降沿触发
HFPIO_IT_RISE_EDGE : 中断上升沿触发
HFPIO_IT_EDGE :中断边沿触发

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：

要求：

所在头文件：在 hfgpio.h 中声明
所在库：libKernel.a
HSF 版本：HSF V1.0 以上
硬件：LPBXX

2.6.1 hfgpio_fconfigure_get

获取功能码对应的 PIN 脚对应的属性值;

int HSF_API hfgpio_fconfigure_get(int fid);

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：

无

要求：

所在头文件：在 hfgpio.h 中声明
所在库：libKernel.a
HSF 版本：HSF V1.16 以上
硬件：LPBXX

2.6.2 hfgpio_fpin_ add_feature

对功能码对应的 PIN 脚添加属性值;

int HSF_API hfgpio_fpin_add_feature(int fid,int flags);;

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：

无

要求：

所在头文件：在 hfgpio.h 中声明
所在库：libKernel.a
HSF 版本：HSF V1.16 以上
硬件：LPBXX

2.6.3 hfgpio_fpin_ clear_feature

清除功能码对应的 PIN 脚的一个或者多个属性值;

int HSF_API hfgpio_fpin_clear_feature (int fid,int flags);;

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：

无

要求：

所在头文件：在 hfgpio.h 中声明
所在库：libKernel.a
HSF 版本：HSF V1.16 以上
硬件：LPBXX

2.6.4 hfgpio_fset_out_high

把功能码对应的 PIN 脚，设置为输出高电平

int hfgpio_fset_out_high(int fid);

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 不能做输入脚

备注：

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

例子：

参考 example/gpio

要求：

所在头文件：在 hfgpio.h 中声明
所在库：libKernel.a
HSF 版本：HSF V1.0 以上
硬件：LPBXX

2.6.5 hfgpio_fset_out_low

把功能码对应的 PIN 脚设置为输出低电平;

int hfgpio_fset_out_low(int fid);

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

备注：

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

例子：

参考 hfgpio_fset_out_high;

要求：

所在头文件：在 hfgpio.h 中声明
所在库：libKernel.a
HSF 版本：HSF V1.0 以上
硬件：LPBXX

2.6.6 hfgpio_fpin_is_high

判断功能码对应的 PIN 脚是否为高电平;

int hfgpio_fpin_is_high(int fid);

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：

例子：

参考 example gpio;

要求：

所在头文件：在 hfgpio.h 中声明
所在库：libKernel.a
HSF 版本：HSF V1.0 以上
硬件：LPBXX

2.6.1 hfgpio_configure_fpin_interrupt

配置功能码对应 PIN 脚为中断输入脚，并指定中断入口函数，以及中断触发模式

```
int hfgpio_configure_fpin_interrupt(
                    int fid,
                    uint32_t flags,
                    hfgpio_interrupt_func handle,
                    int enable);
```

参数：

fid:要配置的功能码,系统固定功能码可以参考 HF_GPIO_FUNC_E,也可以为用户自定义的功能码；

flags:

设置中断触发模式，中断模式可以为：

HFPIO_IT_LOW_LEVEL:低电平触发
HFPIO_IT_HIGH_LEVEL:高电平触发
HFPIO_IT_FALL_EDGE:下降沿触发
HFPIO_IT_RISE_EDGE:上升沿触发
HFPIO_IT_EDGE:边沿触发

除了设置中断模式,flags可以为可以为其它值做逻辑或操作，具体可以参看hfgpio_configure_fpin 中的flags;

handle:中断入口函数,函数类型
void interrupt_hande(uint32_t,uint32_t);

enable:使能中断，1,配置完成后中断使能，0,配置完成后中断不使能，直到调用 hfgpio_fenable_interrupt(fid),中断才生效;

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 不能做中断脚;

备注：

例子：

参考 example/gpio

要求：

所在头文件：在 hfgpio.h 中声明
所在库：libKernel.a
HSF 版本：HSF V1.0 以上
硬件：LPBXX

2.6.2 hfgpio_fenable_interrupt

使能功能码对应 PIN 中断

int hfgpio_fenable_interrupt(int fid);

参数：

fid:要配置的功能码,系统固定功能码可以参考 HF_GPIO_FUNC_E,也可以为用户自定义的功能码;

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 不能做中断脚;

备注：

调用这个函数之前，一定要先调用 hfgpio_configure_fpin_interrupt 配置中断。

例子：

要求：

所在头文件：在 hfgpio.h 中声明
所在库：libKernel.a
HSF 版本：HSF V1.0 以上
硬件：LPBXX

2.6.3 hfgpio_fdisable_interrupt

禁止功能码对应 PIN 中断

int hfgpio_fdisable_interrupt(int fid);

参数：

fid:要配置的功能码,系统固定功能码可以参考 HF_GPIO_FUNC_E,也可以为用户自定义的功能码；

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 不能做中断脚;

备注：

调用这个函数之前，一定要先调用 hfgpio_configure_fpin_interrupt 配置中断。

例子：

要求：

所在头文件：在 hfgpio.h 中声明
所在库：libKernel.a
HSF 版本：HSF V1.0 以上
硬件：LPBXX

2.6.4 hfgpio_pwm_enable

使能功能码对应 PIN 脚的 PWM 功能.

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

参数：

fid:要配置的功能码,系统固定功能码可以参考 HF_GPIO_FUNC_E,也可以为用户自定义的功能码；

freq: PWM 的频率，LPB 中 pwm 频率是由 12MHZ 分频而来.

hrate:PWM 中高电平的保持百分比，可以为(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：

LPBxx 模组中 PWM 的频率是由 12MHZ 分频而来,功能码对应的 PIN 脚一定具有 F_PWM 属性。

例子：

无

要求：

所在头文件：在 hfgpio.h 中声明

所在库：libKernel.a

HSF 版本：HSF V1.17 以上

硬件：LPBXX

2.6.5 hfgpio_pwm_disable

禁用功能码对应 PIN 脚的 PWM 功能.

```
int HSF_API hfgpio_pwm_disable(int fid);
```

参数：

fid:要配置的功能码,系统固定功能码可以参考 HF_GPIO_FUNC_E,也可以为用户自定义的功能码；

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：

LPBxx 模组中 PWM 的频率是由 12MHZ 分频而来。

例子：

无

要求：

所在头文件：在 hfgpio.h 中声明
所在库：libKernel.a
HSF 版本：HSF V1.17 以上
硬件：LPBXX

2.6.6 hfgpio_adc_enable

使能功能码对应 PIN 脚的 ADC 功能.

int HSF_API hfgpio_adc_enable(int fid);

参数：

fid:要配置的功能码,可以为系统固定功能码,参考 HF_GPIO_FUNC_E,也可以为用户自定义的功能码；

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：

LPB100 的 ADC 为 12 位

例子：

无

要求：

所在头文件：在 hfgpio.h 中声明
所在库：libKernel.a
HSF 版本：HSF V1.17 以上
硬件：LPBXX

2.6.7 hfgpio_adc_get_value

获取功能码对应 PIN 脚的采样值.

int HSF_API hfgpio_adc_get_value(int fid);

参数：

fid:要配置的功能码,可以为系统固定功能码,参考 HF_GPIO_FUNC_E,也可以为用户自定义的功能码；

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：

LPB100 的 ADC 为 12 位,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 3.3V 而已的。

例子：

无

要求：

所在头文件：在 hfgpio.h 中声明
所在库：libKernel.a
HSF 版本：HSF V1.17 以上
硬件：LPBXX

2.7 串口 API

2.7.1 Hfuart_open

打开串口设备

```
hfuart_handle_t HSF_API  hfuart_open(int uart_no);
```

参数：

uart_no:串口号，当前只能为0,1;

返回值：

成功返回指向串口设备的指针；否则返回 NULL

备注：

串口 API hfuart_open 和 hfuart_recv 只能在同一个线程，否则 hfuart_recv 将收不到数据,在使用 uart 之前一定要先调用 hfuart_open;

例子：

无

2.7.2 hfuart_close

打开串口设备

```
hfuart_handle_t HSF_API  hfuart_open(int uart_no);
```

参数：

uart_no:串口号，当前只能为0,1;

返回值：

成功返回 HF_SUCCESS,否则 HF_FAIL;

备注：

在不使用串口的时候调用 hfuart_close 释放资源;

例子：

无

2.7.3 hfuart_send

发送数据到串口

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

参数：

huart:串口设备对象，由hfuart_open返回

data: 要发送的数据的缓存区

bytes:发送数据的长度

timeouts:超时时间

返回值：

成功返回为实际发送的数据，失败返回错误码；

备注：

例子：

参考 example/socket

2.7.4 hfuart_recv

从串口中接收数据

```
int HSF_API hfuart_recv(
        hfuart_handle_t huart,char *recv,
        uint32_t bytes,
        uint32_t timeouts)
```

参数：

huart:串口设备对象，huart:串口设备对象，由hfuart_open返回
recv:保存接收到数据的缓存区；
bytes:接收缓存区长度
timeouts:接收超时时间,当采用 select 操作的时候，timeouts 一定为 0；

返回值：

成功返回实际接收的数据的长度，否则返回错误码;

备注：

如果使用了系统自带的串口透传和命令模式，请不要调用这个函数，可能导致串口透传和命令模式异常;可以使用 hfnet_start_uart 制定回调来获取串口的数据。

例子：

参考 example/socket

2.7.5 hfuart_select

支持串口的 select 模型

```
int HSF_API hfuart_select(
                int maxfdp1,
                fd_set *readset,
               fd_set *writeset,
               fd_set *exceptset,
               struct timeval *timeout)
```

参数：

huart:串口设备对象，huart:串口设备对象，由hfuart_open返回
readset:参考select；
writeset:参考 select
exceptset 参考 select:
timeouts:参考 select；

返回值：

参考 select

备注：

这个函数和标准的 select 用法一样，如果 fd_set 里面不包含 uart_fd，函数功能和 select 一样。fd_set 可以传入 socket 的 fd.

例子：

参考 example/socket

2.8 AT 命令 API

2.8.1 hfat_send_cmd

发送 AT 命令，结果返回到指定的 buffer.
int hfat_send_cmd(char *cmd_line,int cmd_len,char *rsp,int len);

参数：
cmd_line: 包含 AT 命令字符串，
格式为 AT+CMD_NAME[=][arg,]…[argn]<CR><CL>
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 hfuart_send(“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/at

要求：
所在头文件：在 hfgpio.h 中声明
所在库：libKernel.a
HSF 版本：HSF V1.0 以上
硬件：LPBXX

2.8.2 hfat_get_words

获取 AT 命令或者响应的每一个参数值

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

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：

要求：

所在头文件：在 hfgpio.h 中声明

所在库：libKernel.a

HSF 版本：HSF V1.0 以上

硬件：LPBXX

2.9 Debug API

2.9.1 HF_Debug

输出调式信息到串口

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

参数：

debug_level:调式等级，可以为

```
#define DEBUG_LEVEL_LOW          1
#define DEBUG_LEVEL_MID          2
#define DEBUG_LEVEL_HI                   3
```

可以通过hfdbg_set_level来设置调式等级;

format:格式化输出，和 printf 一样；

返回值：

无

备注：

对于没有单独调式串口的设备，调式信息会输出到 AT 命令对应的串口，因此在调式完成后一定要关闭调式；程序发布后要动态打开调式，可以用 AT+NDBGL=level 打开，不需要调式的时候用 AT+NDBGL=0 关闭.

例子：

无

要求：

所在头文件：hf_debug.h
所在库：libKernel.a
HSF 版本要求：V1.0 以上
硬件：LPBXX

2.9.2 hfdbg_get_level

获取当前的调式等级
int hfdbg_get_level ();

参数：

无

返回值：

返回当前的调式等级

备注：

无

例子：

无

要求：

所在头文件：hf_debug.h
所在库：libKernel.a
HSF 版本要求：V1.0 以上
硬件：LPBXX

2.9.3 hfdbg_set_level

设置调试级别，或者关闭调式
void hfdbg_set_level (int debug_level);

参数：

debug_level:调式级别，可以为

```
#define DEBUG_LEVEL_LOW          1
#define DEBUG_LEVEL_MID          2
#define DEBUG_LEVEL_HI                   3
```

返回值：

无

备注：

无

例子：

无

要求：

所在头文件：hf_debug.h
所在库：libKernel.a
HSF 版本要求：V1.0 以上
硬件：LPBXX

2.10用户文件操作 API

2.10.1 hffile_userbin_write

把数据写入到用户文件
int HSF_API hffile_userbin_write(uint32_t offset,char *data,int len);

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

要求：

所在头文件：hffile.h
所在库：libKernel.a
HSF 版本要求：V1.13 以上
硬件：LPBXX

2.10.2 hffile_userbin_read

从用户文件中读数据

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

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;

备注：

无

例子：

无

要求：

所在头文件：hffile.h

所在库：libKernel.a

HSF 版本要求：V1.13 以上

硬件：LPBXX

2.10.3 hffile_userbin_size

获取用户文件的大小

int HSF_API hffile_userbin_size(void);

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

要求：

所在头文件：hffile.h

所在库：libKernel.a

HSF 版本要求：V1.13 以上

硬件：LPBXX

2.10.4 hffile_userbin_zero

把整个文件的内容快速清零

int HSF_API hffile_userbin_zero (void);

参数：

无

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

要求：

所在头文件：hffile.h
所在库：libKernel.a
HSF 版本要求：V1.13 以上
硬件：LPBXX

2.11用户 Flash 操作 API

2.11.1 hfuflash_erase_page

擦写用户 flash 的页

int HSF_API hfuflash_erase_page(uint32_t addr, int pages);

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 128KB 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 中的 uflash.

要求：

所在头文件：hfflash.h
所在库：libKernel.a
HSF 版本要求：V1.16a 以上
硬件：LPBXX

2.11.1 hfuflash_write

从用户文件中读数据

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

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 中的 uflash.

要求：

所在头文件：hffile.h
所在库：libKernel.a
HSF 版本要求：V1.16a 以上
硬件：LPBXX

2.11.2 hfuflash_read

从用户文件中读数据
int HSF_API hfuflash_read(uint32_t addr, char *data, int len);

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：

小于零失败，否则返回实际从 flash 读到的 Bytes 数;

备注：

无

例子：

参考 example 中的 uflash.

要求：

所在头文件：hffile.h
所在库：libKernel.a
HSF 版本要求：V1.16a 以上
硬件：LPBXX

2.12 WIFI API

WIFI 大部分接口可以用 hfat_send_cmd 来实现，具体请参考用户 AT 命令手册。

2.12.1 hfwifi_scan

扫描附件的存在的 AP。

int HSF_API hfwifi_scan(hfwifi_scan_callback_t p_callback);

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
    char    mac[6];//AP的mac地址
}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：

扫描成功返回扫描到 AP 的个数,小于 0 失败，函数返回说明一次扫描结束。

备注：

无

例子：

参考 example 中的 wifi

要求：

所在头文件：hfwifi.h
所在库：libKernel.a
HSF 版本要求：V1.17 以上
硬件：LPBXX

2.12.2 hfsmtlk_start

开始 smartlink.

int HSF_API hfsmtlk_start(void);

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：

参考 example 中的 wifi

要求：

所在头文件：hfsmtlk.h
所在库：libKernel.a
HSF 版本要求：V1.17 以上
硬件：LPBXX

2.12.3 hfsmtlk_stop

停止 smartlink.

int HSF_API hfsmtlk_stop(void);

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：

参考 example 中的 wifi

要求：

所在头文件：hfsmtlk.h
所在库：libKernel.a
HSF 版本要求：V1.17 以上
硬件：LPBXX

2.12.4 hfsmtlk_register

注册 smartlink 抓包的回调函数.

int HSF_API hfsmtlk_register(hfsmtlk_main_callback_t main_callback, hfsmtlk_recv_callback_t recv_callback);

参数：

hfsmtlk_main_callback_t：用户smartlink的主回调函数，用于处理扫描到的AP列表（PWIFI_SCAN_RESULT_ITEM），用户可根据该结果确定扫描信道，加密算法等等

typedef int (*hfsmtlk_main_callback_t)(PWIFI_SCAN_RESULT_ITEM /*ap_list*/,uint32_t /*ap_cnt*/);

hfsmtlk_recv_callback_t：用户处理在扫描过程中收到的数据包，包括原始数据，数据长度，及数据来自的信道号

typedef int (*hfsmtlk_recv_callback_t)(void * /*pkt_data*/,uint32_t /*pkt_length*/,uint8_t /*channel*/);

返回值：

成功返回 HF_SUCCESS,否则失败

备注：

无。

例子：

无。

要求：

所在头文件：hfsmtlk.h
所在库：libKernel.a
HSF 版本要求：V1.40 以上
硬件：LPBXX

2.12.5 hfsmtlk_set_filter

设置 smartlink 抓包时的过滤参数。

int HSF_API hfsmtlk_set_filter(uint8_t channel,uint8_t rssi,uint32_t max_pkt_len,uint32_t min_pkt_len,uint32_t flags);

参数：

channel：信道号

rssi：最低信号强度（0～100，0为默认值，即抓所有的包）

max_pkt_len：最大数据包长度，大于该长度的包将被过滤掉

min_pkt_len：最小数据包长度，小于该长度的包将被过滤掉

flags：以下值的组合，也可以都不选(0)，SCA和SCB选一，如都不选，表示只抓20M（主信道），用户smartlink的主回调函数（见2.12.4）会返回AP支持的参数

#define HFSMTLK_FLAG_RECV_MGT_PKT 0x00000001 //收管理包

#define HFSMTLK_FLAG_CHNL_EXT_SCA 0x00010000 //40M上偏辅助信道，用户smartlink的主回调函数（见2.12.4）会返回AP支持的参数

#define HFSMTLK_FLAG_CHNL_EXT_SCB 0x00020000 //40M下偏辅助信道，用户smartlink的主回调函数（见2.12.4）会返回AP支持的参数

返回值：

成功返回 HF_SUCCESS,否则失败

备注：

无。

例子：

无。

要求：

所在头文件：hfsmtlk.h

所在库：libKernel.a

HSF 版本要求：V1.40 以上

硬件：LPBXX

2.12.6 hfsmtlk_finished_ok

用户的 Smartlink（非汉枫标准 Smartlink）成功时，用于保存 SSID，密码等信息。

void HSF_API hfsmtlk_finished_ok(char *key, int type, uint8_t *apmac,char *ssid);

参数：

key：密码
type：密码类型，定义在Hfwifi.h中

```
#define ENC_TYPE_NONE        0
#define ENC_TYPE_WEP         1
#define ENC_TYPE_TKIP        2
#define ENC_TYPE_AES         3
#define ENC_TYPE_TKIPAES     4
```

apmac：AP的MAC地址
ssid：AP的SSID

返回值：

无。

备注：

无。

例子：

无。

要求：

所在头文件：hfsmtlk.h
所在库：libKernel.a
HSF 版本要求：V1.40 以上
硬件：LPBXX

2.13 自动升级 API

2.13.1 hfupdate_start

开始升级.

```
int hfupdate_start(HFUPDATE_TYPE_E type);
```

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
```

```
    HFUPDATE_WIFIFW,//升级WIFI固件
    HFUPDATE_WEB,//升级web
}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/at

要求：

所在头文件：hfupdate.h
所在库：libKernel.a
HSF 版本要求：V1.17 以上
硬件：LPBXX

2.13.2 hfupdate_write_file

把升级文件数据写到升级区.

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

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/at

要求：

所在头文件：hfupdate.h
所在库：libKernel.a
HSF 版本要求：V1.17 以上
硬件：LPBXX

2.13.3 hfupdate_complete

升级完成

```
Int  hfupdate_complete(
                HFUPDATE_TYPE_E type,
                uint32_t file_total_len
                );
```

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/at

要求：

所在头文件：hfupdate.h
所在库：libKernel.a
HSF 版本要求：V1.17 以上
硬件：LPBXX

<结束>