High-Flying Software Framework (HSF) API Reference Manual V1.1x

version 1.1x
2013/12

Update records:

Update time	editor	update	version
2013.8.26	Jim	First edition	V1.0
2013.9.13	Jim	Add time API and serial transceiver API	V1.03
2013.10.16	Jim	1， Add user file operate interface 2， Add hfsys_get_time API function 3， Modify some description 4， Update hfgpio_configure_fpin_interrupt	V1.13
2013.10.28	Jim	1,add hfnet_httpd_set_get_nvram_callback websever function 2,add WPS, enter WPS through AT+WPS	V1.15
2013.11.19	Jim	1,support SmartLink function 2，add hfuflash_xx interface； 3 , add 2MB flash	V1.16
2013.11.24	Jim	1,add nvm interface 2，add pwm interface 3,add watchdog thread interface 4,add hftimer_change_period interface 5,add hfsys_get_reset_reason function	V1.17
2013.12.03	Jim	1， Timer support hardware timer 2， Fix the bug that UDP cannot receive broadcast packet	V1.17
2013.12.05	Jim	1,add timer counter interface	V1.17

Content

1. Framework Definition

1.1 System Error Code Definition

API function return value(except for special instruction) defines “HF_SUCCESS” or “>0” as success, “<0” as failure. Error code is 4Bytes signed integer, return value is the minus of error code; 24-31 bit is module index;8-23 is reserved; 0-7 is specific error code.

```
#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 Function Description

2.1 Libc Function

HSF is compatible with standard C library, such as memory management, character string, time, standard input and output. Please refer to standard C library for more function description.

2.2 System Function

2.2.1 hfsys_switch_run_mode

Switch system run mode.

int hfsys_switch_run_mode(int mode);

Parameter:

mode: switching run mode, currently system supported run modes as follow:

```
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: Transparent transmit mode
HFSYS_STATE_RUN_CMD: Command mode
HFSYS_STATE_RUN_GPIO: GPIO mode

Return value:

HF_SUCCESS: success; otherwise failure, please check HSF error code.

Requests:

The header file: hfsys.h
The library: libKernel.a
HSF version **Requests:** V1.0 above
Hardware: LPBXX

2.2.2 hfsys_get_run_mode

Acquire current system run mode

int hfsys_get_run_mode()；

Parameter:

none

Return value:

Return to current run mode, run mode can be below value:

```
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
};
```

Remark:

Example:

Requests:

The header file:hfsys.h
The library: libKernel.a
HSFversion Requests :V1.0 above
Hardware: LPBXX

2.2.3 hfsys_malloc

Dynamic allocate memory

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

Parameter:

size: the allocated memory size

Return value:

If NULL, indicates system has no free memory; if success, then return to memory address

Remark:

It is a thread safe function, use this function to develop multi-thread application, do not use malloc from Libc, which is not thread safe.

Example:

Requests:

The header file: hfsys.h
The library: libKernel.a

HSFversion Requests :V1.0 above
Hardware: LPBXX

2.2.4 hfsys_free

Free the allocated memory from hfsys_malloc
void HSF_API hfmem_free(void *pv)；

Parameter:
pv: point to vacancy about to free memory.

Return value:
none

Remark:
It is a thread safe function, use this function to develop multi-thread application, do not use malloc from Libc, which is not thread safe.

Example:
none

Requests:
The header file: hfsys.h
The library: libKernel.a
HSFversion Requests :V1.0 above
Hardware: LPBXX

2.2.5 hfsys_reset

Reset system, IO level is not maintained
void HSF_API hfsys_reset(void);

Parameter:
none

Return value:
none

Remark:
none

Example:
none

Requests:
The header file: hfsys.h
The library: libKernel.a

HSF version Requests :V1.0 above
Hardware: LPBXX

2.2.6 hfsys_softreset

Soft reset system, IO level maintain

void HSF_API hfsys_softreset(void);

Parameter:

none

Return value:

none

Remark:

none

Example:

none

Requests:

The header file: hfsys.h
The library: libKernel.a
HSF version Requests :V1.17above
Hardware: LPBXX

2.2.7 hfsys_reload

System restored to factory setting

void HSF_API hfsys_reload()；

Parameter:

none

Return value:

none

Remark:

after calling this function, recommend call hfsys_reset to reset system

Example:

none

Requests:

The header file: hfsys.h
The library: libKernel.a

HSF version Requests :V1.0 above
Hardware: LPBXX

2.2.8 hfsys_get_time

Acquire the spend time from start to now（MSEL）

uint32_t HSF_API hfsys_get_time (void);

Parameter:

none

Return value:

The MSEL time spent from system back to run to now

Remark:

none

Example:

none

Requests:

The header file: hfsys.h
The library: libKernel.a
HSF version Requests :V1.0 above
Hardware: LPBXX

2.2.9 hfsys_nvm_read

Read data from NVM.

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

Parameter:

nvm_addr:NVM address, can be (0-99);
buf: buffer area for reserving the read data from NVM
length: the sum of length and nvm_addr less than 100;

Return value:

If success feedback HF_SUCCESS, otherwise feedback ＜0

Remark:

when module reset, soft reset, NVM data will not be cleared ,LPB provide 100 bytes NVM, if module power off, the data on NVM will be cleared

Example:

none

Requests:

The header file: hfsys.h
The library: libKernel.a
HSF version Requests :V1.17 above
Hardware: LPBXX

2.2.10 hfsys_nvm_write

Write data in NVM

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

Parameter:

nvm_addr: NVM address， can be(0-99);
buf: buffer area for reserving the data written into NVM
length: the sum of length and nvm_addr less than 100

Return value:

If success feedback HF_SUCCESS, otherwise feedback < 0

Remark:

when module reset, soft reset, NVM data will not be cleared ,LPB provide 100 bytes NVM, if module power off, the data on NVM will be cleared

Example:

none

Requests:

The header file: hfsys.h
The library: libKernel.a
HSF version Requests :V1.17 above
Hardware: LPBXX

2.2.11 hfsys_get_reset_reason

Acquire the reason why module reset

uint32_t HSF_API hfsys_get_reset_reason (void);

Parameter:

none

Return value:

The reasons return to reset can be the one or ones in below list.

HFSYS_RESET_REASON_NORMAL	Module power off
HFSYS_RESET_REASON_ERESET	reset of hardware watch dog or

	press reset key from exterior
HFSYS_RESET_REASON_IRESET0	Software call hfsys_softreset reset (software watchdog reset, or program error or memory access error)
HFSYS_RESET_REASON_IRESET1	Software call hfsys_reset to reset
HFSYS_RESET_REASON_WPS	Module reset for WPS
HFSYS_RESET_REASON_SMARTLINK_START	Launch of Smart Link
HFSYS_RESET_REASON_SMARTLINK_OK	SmartLink config successfully

Remark:

usually by entrance function call, user can judge if this start is reset or blackout start, and the reason why reset. Recovery operation should be done based on different reset reasons..

Example:

Refer to example callbacktest

Requests:

The header file: hfsys.h
The library: libKernel.a
HSF version Requests :V1.17 above
Hardware: LPBXX

2.3 Timer API

The accuracy of LPB timer is 1 ms, the accuracy of LPB100 IS 10ms. If required strict timer, please do not use below timer API function

2.3.1 hftimer_create

Create a timer

```
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 );
```

Parameter:

name: name of timer
period: the trigger period of Timer ,the unit in MS

if flags set as HFTIMER_FLAG_HARDWARE_TIMER, the unit is in μ s

auto_reload: appoint auto or manual. If true, only requires call hftimer_start one time, once Timer was triggered, no need to re-call hftimer_start; if flase, after trigger, requires re-call hftimer_start to re-trigger

timer_id: appoint a only ID, represent timer. When multi timer call one callback function, can use the ID to distinguish timer.

flags: currently can be 0 or HFTIMER_FLAG_HARDWARE_TIMER, if the created timer is hardware timer, please set flags as HFTIMER_FLAG_HARDWARE_TIMER

Return value:

If function succeed, feedback a pointer pointed to timer, otherwise feedback Null

Remark:

After creation of timer, it will not start right away, timer will start after call hftimer_start. If timer was set manually, after timer triggered, requires to re-call hftimer_start to re-trigger; if set as auto, timer will automatically trigger in next period.

If create a hardware timer, flags formulate HFTIMER_FLAG_HARDWARE_TIMER, can only create one hardware timer. The hardware period unit is μ s. Timer may not be accuracy because of hardware, and requires fine tuning. Now around 1374 /1ms. Hardware timer only supported by V1.17 or above

Example:

Create a auto timer with interval as 1s, control the twinkling of nReady light.

```
#include <hsf.h>

#define TEST_TIMER_ID        (1)

hftimer_handle_t test_timer=NULL;

void test_timer_callback( hftimer_handle_t htimer )
{
        if(hftimer_get_timer_id(htimer)==TEST_TIMER_ID)
        {
                //u_printf("TEST_TIMER_ID active\n");
                if(hfgpio_fpin_is_high(HFGPIO_F_NREADY))
                        hfgpio_fset_out_low(HFGPIO_F_NREADY);
                else
                        hfgpio_fset_out_high(HFGPIO_F_NREADY);
                //hftimer_start(htimer);//  if create a manually ,open the comment
        }
}

void  test_timer()
{
```

```
        if((test_timer = hftimer_create("TEST-TIMER",
                        1000,true,1,test_timer_callback,0))==NULL)
        {
                u_printf("create timer fail\n");
        }
}
```

Requests:

The header file :hftimer.h
The library: libKernel.a
HSF version Requests :V1.03 above
Hardware: LPBXX

2.3.2 hftimer_delete

Delete a timer

void HSF_API hftimer_delete(hftimer_handle_t htimer);

Parameter:

htimer: deleted timer created by hftimer_create；

Return value:

none

Example:

Requests:

The header file :hftimer.h
The library: libKernel.a
HSF version: HSF V1.03 above

2.3.3 hftimer_start

Start timer

int HSF_API hftimer_start(hftimer_handle_t htimer);

Parameter:

htimer: created by hftimer_create

Return value:

If success feedback HF_SUCCESS, otherwise feedback HF_FAIL;

Remark:

Example:

Refer to hftimer_create

Requests:

The header file :hftimer.h
The library: libKernel.a
HSF version:HSF V1.03 above

2.3.4 hftimer_stop

Stop timer

void HSF_API hftimer_stop(hftimer_handle_t htimer);

Parameter:

htimer: created by hftimer_create

Return value:

none;

Remark:

after call the function, timer will no longer triggered, until re-call hftimer_start;

Example:

Requests:

The header file :hftimer.h
The library: libKernel.a
HSF version:HSF V1.03 above

2.3.5 hftimer_get_timer_id

Acquire tiimer ID

uint32_t HSF_API hftimer_get_timer_id(hftimer_handle_t htimer);

Parameter:

htimer: created by hftimer_create

Return value:

If success, feedback timer's ID, appointed by hftimer_create; if failure, feedback HF_FAIL;

Remark:

this function usually be called when timer callback or to distinguish the circumstances when multi timer use the same callback function

Example:

Refer to hftimer_create

Requests:

The header file :hftimer.h
The library: libKernel.a
HSF version:HSF V1.03 above

2.3.6 hftimer_change_period

Change the period of timer

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

Parameter:

htimer: created by hftimer_create
new_period: new period, unit is ms;

Return value:

none;

Remark:

change period of timer, after the function call, timer will run with new period

Example:

Refer to example timer

Requests:

The header file :hftimer.h
The library: libKernel.a
HSF version:HSF V1.17 above

2.3.7 hftimer_get_counter

Acquire the CLK counter from hardware timer from start to now

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

Parameter:

htimer: appoint to hardware timer created by hftimer_create

Return value:

Feedback the CLK counter from the timer start to now. The current frequency of LPB100 IS 48mhz, a CLK is 1/48 us. From the start of timer to now, the time is counter/48 us. If return value is 0, it indicates the time of timer is over.

Remark:

of software requires more accurate time, it can be realized by this function

plus hardware timer.

Example:

Refer to example timer

Requests:

The header file :hftimer.h
The library: libKernel.a
HSF version:HSF V1.17 above

2.4 Multi Thread 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);
```

description:create a thread

Parameter:

routine:input parameter: entrance function of thread

```
typedef void (*PHFTHREAD_START_ROUTINE)( void * );
```

stack_depth: input parameter thread stack depth， depth is 4Bytes/unit，stack_size = stack_depth*4;

parameters: input parameter, thread entrance function parameter;

uxpriority: input parameter，thread priority level，HSF priority level has:

HFTHREAD_PRIORITIES_LOW: priority level low
HFTHREAD_PRIORITIES_MID:priority level middle
HFTHREAD_PRIORITIES_NORMAL: priority level normal
HFTHREAD_PRIORITIES_HIGH:priority level high
User thread usually apply HFTHREAD_PRIORITIES_MID, HFTHREAD_PRIORITIES_LOW;

created_thread: optional，if function succeed, returns a pointer to the thread creation ; if none, no returns

stack_buffer: reserve for further use

Return value:

HF_SUCCESS: succeed, otherwise failure, please check HSF error code

Example:

```
#include <hsf.h>

//thread entrance fucntion
void test_thread_func(void *arg)
{
        while(1)
        {
                msleep(1000);// thread sleep 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;
}
```

Requests:

The header file :hfthread.h
The library: libKernel.a
HSF version Requests :V1.0 above
Hardware: LPBXX

2.4.2 hfthread_delay

delay current thread ms
void hf_thread_delay(uint32_t ms);

Parameter:

ms , appoint delayed time(unit is ms)

Return value:

This function no Return value

Requests:

The header file :hfthread.h
The library: libKernel.a
HSF version:HSF V1.0 above

2.4.3 **hfthread_destroy**

void hfthread_destroy(hfthread_hande_t thread);

Description:

delete the thread created by hfthread_create

Parameter:

thread: point to deleted thread, if null, delete current thread

Return value:

This function has no Return value

Requests:

The header file :hfthread.h
The library: libKernel.a
HSF version:HSF V1.0 above

2.4.4 **hfthread_enable_softwatchdog**

enable the software watchdog of the thread

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

Parameter:

thread: a pointer to the thread，feedback hfthread_create，this parameter can be NULL, if NULL, enable the software watch dog of the thread

time: software watchdog overtime, unit is S

Return value:

HF_SUCCESS: succeed , otherwise failure, please check HSF error code，

Remark:

thread watchdog can check thread locking, if watchdog enables, thread does not call hfthread_reset_softwatchdog in set time, LPB module will soft-reset. This function can be re-called many time, can change overtime dynamically. When calling, system will reset thread software watchdog

thread watchdog is disabled default. It only works when calling this thread function.

Example:

Refer to example thread.

Requests:

The header file :hfthread.h
The library: libKernel.a
HSF version:HSF V1.7 above

2.4.5 hfthread_disable_softwatchdog

software watchdog used for disable thread

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

Parameter:

thread: a pointer to the thread，feedback hfthread_create，this parameter can be NULL, if NULL, disable the software watchdog of current thread

Return value:

HF_SUCCESS: succeed , otherwise failure, please check HSF error code

Remark:

during the thread running process, if one operation takes too long time (or wait for a signal for too long time), and bigger than overtime, user can disable watchdog in order to prevent watchdog effect and restart module because of too long operation time. After operation finished, enables watchdog.

Example:

Refer to example thread.

Requests:

The header file :hfthread.h
The library: libKernel.a
HSF version:HSF V1.7 above

2.4.6 hfthread_reset_softwatchdog

software watchdog for thread reset (feed dog)。

```
int HSF_API hfthread_disable_softwatchdog(

                                    );
```

Parameter:

thread: pointer to the thread, feedback hfthread_create，the parameter can be NULL, if NULL, reset software watchdog of current thread

Return value:

HF_SUCCESS: success , otherwise failure, please check HSF error code

Remark:
after enables watchdog, thread must call this function in set time to feed dog; when overtime, module will soft reset

Example:
Refer to example thread.
Requests:
The header file :hfthread.h
The library: libKernel.a
HSF version:HSF V1.7 above

2.4.7 hfthread_mutext_new

int HSF_API hfthread_mutext_new(hfthread_mutex_t *mutex)

Description:
create a thread mutex

Parameter:
mutex: function succeeds, return and point to the created nutex
Return value
HF_SUCCESS: success , otherwise failure, please check HSF error code
Remark:
when do not use mutex, please call hfthread_mutext_free to release resource

Example:
Requests:
The header file :hfthread.h
The library: libKernel.a
HSF version:HSF V1.0 above

2.4.8 hfthread_mutext_free

void hfthread_mutext_free(hfthread_mutex_t mutex);

Description:
delete thread created by hfthread_mutext_new

Parameter:
mutex: point to deleting mutex

Return value:
The function has no Return value ;

Example:

Refer to hfthread_create

Requests:

The header file :hfthread.h
The library: libKernel.a
HSF version:HSF V1.0 above

2.4.9 hfthread_mutext_unlock

void hfthread_mutext_unlock(hfthread_mutex_t mutex);

Description:

free mutes

Parameter:

mutex: point to a mutex, created by hfthread_mutext_new

Return value:

The function has no Return value ;

Example:

Refer to hfthread_create

Requests:

The header file :hfthread.h
The library: libKernel.a
HSF version:HSF V1.0 above

2.4.10 hfthread_mutext_lock

int hfthread_mutext_lock (hfthread_mutex_t mutex);

Description:

Parameter:

mutex: point to a mutex,created by hfthread_mutext_new

Return value:

HF_SUCCESS indicates success; HF_FAIL may occur deadlock, others please refer HSF error code

Remark:

hfthread_mutext_lock and hfthread_mutex_unlock occurs in pair. If call hfthread_mutex_lock, and do not call hfthread_mutex_unlock at the same time, recall hfthread_mutex_lock may occurs deadlock

Example:

Requests:

The header file :hfthread.h
The library: libKernel.a
HSF version:HSF V1.0 above

2.5 Network API

2.5.1 hfnet_ping

send ping package to target address, check if the address is reachable
int hfnet_ping(const char* ip_address);

Parameter:

ip_address: check the character string of target IP address，address form is xxx.xxx.xxx.xxx, if ping a host name, please call hfnet_gethostbyname to get IP address;

Return value:

Succeed feedback HF_SUCCESS, otherwise failure, please refer to HSF error code for specific reason

Remark:

if network disconnect, DNS server wrong will lead failure

Example:

Requests:

The header file :hfnet.h
The library: libKernel.a
HSF version:HSF V1.0 above

2.5.1 hfnet_gethostbyname

acquire IP address of host name。

Parameter:

Return value:

Succeed feedback HF_SUCCESS，HF_FAIL indicated failure

Remark:

Example:

Requests:

The header file :hfnet.h
The library: libKernel.a
HSF version:HSF V1.0 above

2.5.1 hfnet_start_httpd

start httpd, s small sized web server。

int hfnet_start_httpd(uint32_t uxpriority)；

Parameter:

uxpriority:httpd service priority level ,please refer to hfthread_create parameteruxpriority;

Return value:

Succeed feedback HF_SUCCESS，HF_FAIL indicates failure

Remark:

If application requires to support web interface, please call this function when start

Example:

Requests:

The header file :hfnet.h
The library: libKernel.a
HSF version:HSF V1.0 above

2.5.2 hfnet_httpd_set_get_nvram_callback

setup webserver, get module parameter callback setup

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

Parameter:

p_set:optional parameter，if no need to extend WEB parameter interface, please set as NULL, otherwise point to entrance function

the type of callback function setting as below:

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

cfg_name is the name of correspondent configuration,，name_len is the length of cfg_name，value is the configuration value，val_len is the length of value

p_get: optional parameter，if do not need to extend WEB to get parameter 如果 interface, please set as NULL, otherwise point to the entrance function to get parameter

read callback function type of **Parameter:**

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

cfg_name: read the name of parameter. Attention: cfg_name did not necessary include the end of string; name_len: the length of cfg_name; value: reserve the value of cfg_name; val_len: the length of value

Return value:

null

Remark:

Example:

Refer to SDK example file .

Requests:

The header file :hfnet.h

The library: libKernel.a

HSF version:HSF V1.15 above

2.5.3 hfnet_start_socketb

Start the own socket service of start HSF

int hfnet_start_socketb(uint32_t uxpriority,hfnet_callback_t p_callback);

Parameter:

uxpriority: socket service priority level, please refer to hfthread_create parameteruxpriority;

p_callback: callback function, optional; if do not need, set the value of callback as NULL, it will be triggered when socketb receive data package

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

event: time ID ,reserved for further application; right now only support the data already received by socketb

data: point to buffer for receiving data, user can change the value of buffer in callback function；

len: the length of received data

buf_len: the actual length of data pointed to buffer the value is no less than len

callback function Return value ， it is the length of processed value for user, if user do not change the data but read, the value is equal to len

Return value:

Succeed feedback HF_SUCCESS，HF_FAIL indicates failure

Remark:

when socketb service receive data from network, call p_callback, and send the value processed by p_callback to serial, user can use p_callback to resolve or secondary treat the data, such as encryption or decryption, send the processed data back to socketb service.

Example:

Below example realized when socket receiving data from network, add the length of received data to buffer's last two bytes.

```
int socketb_recv_callback( uint32_t event,void *data,uint32_t len,uint32_t buf_len)
{
        If(buf_len>len+2)
                return len;
        data[len]=len&0xFF;
        data[len+1]=(len&0x00FF)>>8;
        return len+2;
}
int USER_FUNC app_main (void)
{

        if(hfnet_start_socketb(HFTHREAD_PRIORITIES_LOW,(hfnet_callback_t)socketb_re
cv_callback)!=HF_SUCCESS)
        {
                HF_Debug(DEBUG_WARN,"start socketb fail\n");
        }
        ……
}
```

Requests:

The header file :hfnet.h
The library: libKernel.a
HSF version:HSF V1.0 above

2.5.4 hfnet_start_socketa

start the own socket service of HSF

int hfnet_start_socketa(uint32_t uxpriority,hfnet_callback_t p_callback);

Parameter:

uxpriority:socket service priority level please refer to hfthread_create parameteruxpriority

p_callback: optional，if not use callback, set as NULL,please refer to

hfnet_start_socketb

Return value:

If succeed feedback HF_SUCCESS，HF_FAIL indicates failure

Remark:

Example:

Please refer to hfnet_start_socketb；

Requests:

The header file :hfnet.h
The library: libKernel.a
HSF version:HSF V1.0 above

2.5.5 hfnet_start_uart

start own UART serial transceiver service of HSF

int hfnet_start_uart(uint32_t uxpriority,hfnet_callback_t p_uart_callback);

Parameter:

uxpriority:uart service priority level please refer to hfthread_create parameteruxpriority

p_uart_callback: serial callback function, optional; if not use, please set as NULL, user can call when serial receive data, the description of callback function and parameter please refer to hfnet_start_socketb;

Return value:

If succeed feedback HF_SUCCESS，HF_FAIL indicates failure.

Remark:

when serial receiving data, if p_uart_callback is not NULL, call p_uart_callback first; if work in transparent transmit mode, send the receiving data to socketa, sockatb servie (if these two services existed), if work in command mode, pass the receiving command to command resolve program

under transparent transmit mode, user can realize the encryption, decryption and secondary treatment of data by the callback function and socketa, socketb service; under command mode, user can define AT command name and form by callback.

Example:

Requests:

The header file :hfnet.h
The library: libKernel.a
HSF version:HSF V1.0 above

2.5.6 hfnet_socketa_send

send data to SOCKETA

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

Parameter:

data: buffer area for reserve sending data
len: the length of sending buffer
timeouts: send timeout, not available currently

Return value:

If succeed, feedback the length of actual sending data, otherwise feedback error code

Remark:

Example:

null

Requests:

The header file :hfnet.h
The library: libKernel.a
HSF version:HSF V1.03 above

2.5.7 hfnet_socketb_send

send data to SOCKETB

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

Parameter:

data: buffer area for reserve sending data
len: the length of sending buffer;
timeouts: send timeout, not available currently

Return value:

If succeed, feedback the length of actual sending data, otherwise feedback error code

Remark:

Example:

null

Requests:

The header file :hfnet.h
The library: libKernel.a
HSF version:HSF V1.03 above

2.5.8 hfnet_set_udp_broadcast_port_valid

set the valid area of broadcast port of UDP

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

Parameter:

start_port: start port number;
end_port: end port number;

Return value:

If succeed ,feedback HF_SUCCESS, otherwise feedback -HF_E_INVAL;

Remark:

LPB100 will default to filter broadcast in network to unburden the system. So if the created socket need to receive broadcast, user need to set listening port through the function

Example:

Refer to example of threadtest

Requests:

The header file :hfnet.h
The library: libKernel.a
HSF version:HSF V1.17 above

2.5.9 standard socket API

HSF apply lwip protocol stack ,compatible with standard socket interface, such as socket, recv, select,sendto,ioct. If source code apply standard socket function, user just need to import head file hsf.hand hfnet.h. the use of standard socket please refer to relevant Manuel.

Remark:since the limitation of system, when set up socket by lwip, please stay the same thread when set up socket and receive, but not

in the same thread when sending data, or data will not be received

2.6 GPIO Control API

2.6.1 hfgpio_configure_fpin

int hfgpio_configure_fpin(int fid,int flags);
description:configure matched pin according to fid
Parameter:fid （function ID）

```
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
};
```

User can define own FID, start from HFGPIO_F_USER_DEFINE

flags:

PIN parameter，can be one or more values as below to run ”|” calculate

HFPIO_DEFAULT
HFPIO_PULLUP:pull up inside

HFPIO_PULLDOWN:pull down inside
HFPIO_IT_LOW_LEVEL: low level trigger interrupt
HFPIO_IT_HIGH_LEVEL: high level trigger interrupt
HFPIO_IT_FALL_EDGE :fall edge trigger interrupt
HFPIO_IT_RISE_EDGE: rise edge trigger interrupt
HFPIO_IT_EDGE :edge trigger interrupt

HFM_IO_TYPE_INPUT:input type
HFM_IO_OUTPUT_0 :low level output
HFM_IO_OUTPUT_1 :high level output

Return value:

HF_SUCCESS: succeed setting, HF_E_INVAL: fid invalid ,or its correspondent PIN invalid, HF_E_ACCES: correspondent PIN do not have the attributes (flags), for example, the correspondent PIN of HFGPIO_F_JTAG_TCK is a peripheral PIN, not GPIO, cannot configure any other attribute except HFPIO_DEFAULT.

Remark:

before setting, user need to figure out the attribute of each PIN correspondent to the FID. Please refer to data manual for the attribute of each PIN. If configure the PIN an attribute it doesn't have, it will feedback to HF_E_ACCES error.

Example:

Requests:

The header file :declare in hfgpio.h
The library: libKernel.a
HSF version:HSF V1.0 above
Hardware: LPBXX

2.6.1 hfgpio_fconfigure_get

acquire the attribute value of PIN correspondent to FID

int HSF_API hfgpio_fconfigure_get(int fid);

Parameter:

fid: function ID, refer to HF_GPIO_FUNC_E, or user can define their own function ID

Return value:

If succeed, feedback the attribute value of PIN, attribute value please refer

to hfgpio_configure_fpin; HF_E_INVAL: FID is invalid or its correspondent PIN is invalid

Remark:

none

Example:

none

Requests:

The header file :declare in hfgpio.h
The library: libKernel.a
HSF version:HSF V1.16 above
Hardware: LPBXX

2.6.2 hfgpio_fconfigure_ add_feature

add attribute value to PIN correspondent to FID

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

Parameter:

fid: function ID, please refer HF_GPIO_FUNC_E, or can be self-defined function ID
flags: refer to hfgpio_configure_fpin flags;

Return value:

HF_SUCCESS: succeed; HF_E_INVAL: fid is invalid, or its correspondent PIN is invalid

Remark:

none

Example:

none

Requests:

The header file :declare in hfgpio.h
The library: libKernel.a
HSF version:HSF V1.16 above
Hardware: LPBXX

2.6.3 hfgpio_fconfigure_ clear_feature

clear one or more attribute value of PIN correspondent to FID

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

Parameter:

fid: function ID, please refer HF_GPIO_FUNC_E, or can be self-defined function ID

flags: refer to hfgpio_configure_fpin flags;

Return value:

HF_SUCCESS: succeed ; HF_E_INVAL: fid is invalid, or its correspondent PIN is valid

Remark:

none

Example:

none

Requests:

The header file :declare in hfgpio.h
The library: libKernel.a
HSF version:HSF V1.16 above
Hardware: LPBXX

2.6.4 hfgpio_fset_out_high

set the FID correspondent PIN as output high level

int hfgpio_fset_out_high(int fid);

Parameter:

fid: please refer HF_GPIO_FUNC_E, or can be self-defined function ID

Return value:

HF_SUCCESS: succeed ; HF_E_INVAL: fid is invalid, or its correspondent PIN is invalid. HF_FAIL: failure ; HF_E_ACCES: correspondent PIN can no t be input PIN

Remark:

this function equal to hfgpio_configure_fpin(fid, HFM_IO_OUTPUT_1| HFPIO_DEFAULT);

Example:

Below code control nLink light:

```
#include <hsf.h>

while(1)
{
        hfgpio_fset_out_high(HFGPIO_F_NLINK);
        msleep(1000);
        hfgpio_fset_out_low(HFGPIO_F_NLINK);
        msleep(1000);
}
```

Requests:

The header file :declare in hfgpio.
The library: libKernel.a
HSF version:HSF V1.0 above
Hardware: LPBXX

2.6.5 hfgpio_fset_out_low

set the FID correspondent PIN as output low level

int hfgpio_fset_out_low(int fid);

Parameter:

fid: function ID, please refer HF_GPIO_FUNC_E, or user can define their own FID

Return value:

HF_SUCCESS: succeed ， HF_E_INVAL: fid is invalid or its correspondent PIN is invalid

Remark:

this function equals to hfgpio_configure_fpin(fid, HFM_IO_OUTPUT_0| HFPIO_DEFAULT);

Example:

Refer to hfgpio_fset_out_high;

Requests:

The header file :declare in hfgpio.h
The library: libKernel.a
HSF version:HSF V1.0 above
Hardware: LPBXX

2.6.6 hfgpio_fpin_is_high

judge if the FID correspondent PIN is high level

int hfgpio_fpin_is_high(int fid);

Parameter:

fid: function ID, refer to HF_GPIO_FUNC_E, user can define their own FID

Return value:

If the PIN is low level ,feedback 0，otherwise it is high level

Remark:

Example:

Refer to example gpio;

Requests:

The header file :declare in hfgpio.h
The library: libKernel.a
HSF version:HSF V1.0 above
Hardware: LPBXX

2.6.1 hfgpio_configure_fpin_interrupt

configure FID correspondent PIN as interrupt input PIN, appoint interrupt entrance function and interrupt trigger mode

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

Parameter:

fid: configure function ID, system fixed function ID can refer to HF_GPIO_FUNC_E, or user can define their own FID

flags:

configure interrupt trigger mode, interrupt mode can be:
HFPIO_IT_LOW_LEVEL: low level trigger
HFPIO_IT_HIGH_LEVEL: high level trigger
HFPIO_IT_FALL_EDGE: fall edge trigger
HFPIO_IT_RISE_EDGE: rise edge trigger
HFPIO_IT_EDGE: edge trigger

Except configure interrupt mode, flags can be other value, please

refer to hfgpio_configure_fpin for details

handle: interrupt entrance function, function type
void interrupt_hande(uint32_t,uint32_t);

enable: enables interrupt; 1 after configuration, enables interruption; 0 after configuration, disable interruption, the interruption take effect until call hfgpio_fenable_interrupt(fid) .

Return value:

HF_SUCCESS: succeed ; HF_E_INVAL: fid is invalid, or its correspondent PIN is invalid HF_FAIL: failure ; HF_E_ACCES: its correspondent PIN cannot be interrupt PIN

Remark:

Example:

Requests:

The header file : declare in hfgpio.h
The library: libKernel.a
HSF version:HSF V1.0 above
Hardware: LPBXX

2.6.2 hfgpio_fenable_interrupt

enables interruption of fid correspondent PIN

int hfgpio_fenable_interrupt(int fid);

Parameter:

fid: configure function ID, system fixed please refer to HF_GPIO_FUNC_E, or can be self-defined function ID

Return value:

HF_SUCCESS: succeed configuration ;HF_E_INVAL: fid is invalid, or its correspondent PIN is invalid ;HF_FAIL: failure ; HF_E_ACCES: the correspondent PIN cannot be interrupt PIN

Remark:

before call this function, must call hfgpio_configure_fpin_interrupt first to configure interrupt.

Example:

Requests:

The header file :declare in hfgpio.h
The library: libKernel.a
HSF version:HSF V1.0 above
Hardware: LPBXX

2.6.3 hfgpio_fdisable_interrupt

disable the interruption of FID correspondent PIN

int hfgpio_fdisable_interrupt(int fid);

Parameter:

fid: configured function ID, system fixed FID ,please refer to HF_GPIO_FUNC_E, or can be self-defined function ID

Return value:

HF_SUCCESS: succeed configuration ， HF_E_INVAL: fid is invalid, or its correspondent PIN is invalid ;HF_FAIL: failure ;HF_E_ACCES: the correspondent PIN cannot be interrupt PIN

Remark:

before call this function, must call hfgpio_configure_fpin_interrupt first to configure interrupt.

Example:

Requests:

The header file :declare in hfgpio.h
The library: libKernel.a
HSF version:HSF V1.0 above
Hardware: LPBXX

2.6.4 hfgpio_pwm_enable

enables the PWM function of FID correspondent PIN

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

Parameter:

fid: configured function ID, system fixed FID ,please refer to HF_GPIO_FUNC_E, or can be self-defined function ID
freq: frequency of PWM, the frequency of pwm in LPB is divided from

12MHZ.
hrate: the rate of high level in PWM, can be (1-99);

Return value:

HF_SUCCESS: configure succeed ， HF_E_INVAL: fid is invalid or its correspondent PIN is invalid. HF_FAIL: failure ;HF_E_ACCES: the correspondent PIN do not have F_PWM attribute, cannot configure as PWM mode

Remark:

the PWM frequency of LPBXXX module is divided from 12 mhz

Example:

none

Requests:

The header file :declare in hfgpio.h
The library: libKernel.a
HSF version:HSF V1.17 above
Hardware: LPBXX

2.6.5 hfgpio_pwm_disable

disable PWM function of FID correspondent PIN
int HSF_API hfgpio_pwm_disable(int fid);

Parameter:

fid: configured function ID, system fixed FID ,please refer to HF_GPIO_FUNC_E, or can be self-defined function ID

Return value:

HF_SUCCESS: configure succeed ， HF_E_INVAL: fid is invalid or its correspondent PIN is invalid. HF_FAIL: failure ;HF_E_ACCES: the correspondent PIN do not have F_PWM attribute, cannot configure as PWM mode

Remark:

the PWM frequency of LPBXXX module is divided from 12 Mhz

Example:

none

Requests:

The header file :declare in hfgpio.h
The library: libKernel.a
HSF version:HSF V1.17 above

Hardware: LPBXX

2.7 Serial API

2.7.1 hfuart_send

send data to uart serial

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

Parameter:

huart: serial device object, can be HFUART0，HFUART1, equals to a module with only one serial , device object is HFUART0;

data: buffer area of sending area
bytes: the length of sent data
timeouts: overtime

Return value:

If succeed, feedback the actual sending data,; if failure, feedback error code

Remark:

Example:

none

2.7.2 hfuart_recv

acquire current debug level

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

Parameter:

huart: serial device object, can be HFUART0，HFUART1, equals to a module with only one serial , device object is HFUART0;
recv: buffer area for reserve receiving data
bytes: length of receiving buffer

timeouts: receive time out

Return value:

If succeed, feedback the actual sending data,; otherwise feedback error code

Remark:

if apply system own transparent mode and command mode, please do not call this function, it may cause abnormal ; can use hfnet_start_uart to callback and acquire serial data

Example:

none

2.8 AT command API

2.8.1 hfat_send_cmd

send AT command, the outcome back to appointed buffer.
int hfat_send_cmd(char *cmd_line,int cmd_len,char *rsp,int len);

Parameter:

cmd_line: includes AT command character string
form is AT+CMD_NAME[=][arg,]…[argn]<CR><CL>
cmd_len: the length of cmd_line, includes ending mark
rsp: buffer for reserve AT command execution result
len: the length of rsp

Return value:

HF_SUCCESS: succeed， HF_FAIL: failure

Remark:

this function is the same with send AT command through UART. This function do not support “AT+H” and “AT+WSCAN”; AT command execution result reserved in RSP, RSP is a character string, the form please refer to AT command help menu; through this function, user can get system configuration.

Example:

Module IP address in network: 10.10.100.254

```
#include <hsf.h>

char rsp[64]={0};
char *words[6];

if(hfat_send_cmd(“AT+LANN\r\n”,sizeof(“AT+LANN\r\n”,rsp,64)==HF_SUCCESS)
```

```
{
        HF_Debug(DEBUG_LEVEL,"%s\n",rsp);
        if(hfat_get_words(rsp,words,6)>0)
        {
        HF_Debug(DEBUG_LEVEL,"LAN IP=%s MASK=%s\n",words[1],words[2]);
        }

}
```

Execute result:
LAN IP=10.10.100.254 MASK=255.255.255.0

Requests:

The header file :declare in hfgpio.h
The library: libKernel.a
HSF version:HSF V1.0 above
Hardware: LPBXX

2.8.2 hfat_get_words

get AT commmand or each responsive parameter value
int hfat_get_words((char *str,char *words[],int size);

Parameter:

str: point to AT command or response; correspondent RAM address can be write and read
words: reserve each parameter value
size: number of word

Return value:

<=0 str correspondent character string is incorrect AT command or illegal response ;>0 the number of word in correspondent character string

Remark:

AT command divided by ",","=" "\r\n"

Example:

Requests:

The header file :declare in hfgpio.h
The library: libKernel.a
HSF version:HSF V1.0 above
Hardware: LPBXX

2.9 Debug API

2.9.1 HF_Debug

Output debug message to serial

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

Parameter:

debug_level: debug level can be:

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

Set debug level by hfdbg_set_level ;

format: formatted output , the same as eprintf

Return value :

none

Remark:

for device without debug serial, debug message will be output to AT command correspondent serial, so must close debug after debug completion; after program released, debug should be open, can be opened by AT+NDBGL=level; closed by AT+NDBGL=0

Example:

none

Requests:

The header file :hf_debug.h

The library: libKernel.a

HSF version Requests :V1.0 above

Hardware: LPBXX

2.9.2 hfdbg_get_level

acquire current debug level

int hfdbg_get_level ();

Parameter:

none

Return value:

Feedback current debug level

Remark:

none

Example:
none

Requests:
The header file :hf_debug.h
The library: libKernel.a
HSF version Requests :V1.0 above
Hardware: LPBXX

2.9.3 hfdbg_set_level

set debug level or close debug
void hfdbg_set_level (int debug_level);

Parameter:
debug_level: debug level can be

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

Return value:
none

Remark:
none

Example:
none

Requests:
The header file :hf_debug.h
The library: libKernel.a
HSF version Requests :V1.0 above
Hardware: LPBXX

2.10 User File Operate API

2.10.1 hffile_userbin_write

write data into user file
int HSF_API hffile_userbin_write(uint32_t offset,char *data,int len);

Parameter:
offset: file offset
data: buffer area for reserve the written file data
len : the size of buffer area

Return value:

If < 0 means failure, otherwise feedback the actual bytes written into file

Remark:

user configuration file is a fixed size file, file reserved in flash. It can reserve user data. User configuration file has backup function. When power off when written, it will auto recover to former content.

Example:

none

Requests:

The header file :hffile.h
The library: libKernel.a
HSF version Requests :V1.13 above
Hardware: LPBXX

2.10.2 hffile_userbin_read

read data from user file
int HSF_API hffile_userbin_read(uint32_t offset,char *data,int len);

Parameter:

offset: file offset
data: buffer area for reserve the written file data
len : the size of buffer area;

Return value:

If < 0 means failure, otherwise feedback the actual bytes read from file

Remark:

none

Example:

none

Requests:

The header file :hffile.h
The library: libKernel.a
HSF version Requests :V1.13 above
Hardware: LPBXX

2.10.3 hffile_userbin_size

acquire the size of user file
int HSF_API hffile_userbin_size(void);

Parameter:
none

Return value:
If < 0 means failure, otherwise feedback the size of file

Remark:
none

Example:
none

Requests:
The header file :hffile.h
The library: libKernel.a
HSF version Requests :V1.13 above
Hardware: LPBXX

2.10.4 hffile_userbin_zero

clear the whole file
int HSF_API hffile_userbin_zero (void);

Parameter:
none

Return value:
If < 0 means failure, otherwise feedback the size of file

Remark:
call this function can quickly clear the whole file, much quicker than hffile_userbin_write

Example:
none

Requests:
The header file :hffile.h
The library: libKernel.a
HSF version Requests :V1.13 above

Hardware: LPBXX

2.11 User Flash Operate API

2.11.1 hfuflash_erase_page

erase user's flash page

int HSF_API hfuflash_erase_page(uint32_t addr, int pages);

Parameter:

addr: the logic address of user flash, not physical address

pages: the flash page about to erase

Return value:

If succeed feedback HF_SUCCESS; if failure feedback HF_FAIL;

Remark:

user flash is a 128 kb area in physical flash; user can only operate this area through API; API operation address is logic address of user flash, we don't need to concern its actual address

Example:

Refer to example uflash.

Requests:

The header file :hfflash.h

The library: libKernel.a

HSF version Requests :V1.16a above

Hardware: LPBXX

2.11.1 hfuflash_write

write data from user file

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

Parameter:

addr: logic address of user flash (0- HFUFLASH_SIZE-2);

data: buffer area for reserve data written into flash

len : size of buffer area

Return value:

If < 0 means failure, otherwise feedback the actual bytes write into file

Remark:

none

Example:

Refer to example uflash.

Requests:

The header file :hffile.h
The library: libKernel.a
HSF version Requests :V1.16a above
Hardware: LPBXX

2.11.2 hfuflash_read

read data from user file
int HSF_API hfuflash_read(uint32_t addr, char *data, int len);

Parameter:

addr: logic address of user flash (0- HFUFLASH_SIZE-2);
data: buffer area for reserve data written into flash
len : size of buff area ;

Return value:

If < 0 means failure, otherwise feedback the actual bytes read from file

Remark:

none

Example:

Refer to example uflash.

Requests:

The header file :hffile.h
The library: libKernel.a
HSF version Requests :V1.16a above
Hardware: LPBXX

<end>