API

MC300 API Manual

2016-10-10

CATALOGUE

1. Linux STANDARD C Function 4
2. System Error Code Defination 5
3. AT Commad API 7
   - **hfat_get_words** 7
   - **hfat_send_cmd** 7
4. DEBUG API 9
   - **HF_Debug** 9
   - **hfdbg_get_level** 9
   - **hfdbg_set_level** 9
5. GPIO Control API 11
   - **hfgpio_configure_fpin** 11
   - **hfgpio_fconfigure_get** 12
   - **hfgpio_fpin_add_feature** 12
   - **hfgpio_fpin_clear_feature** 13
   - **hfgpio_fpin_is_high** 13
   - **hfgpio_fset_out_high** 14
   - **hfgpio_fset_out_low** 14
6. WIFI API 15
   - **hfsmtlk_start** 15
   - **hfsmtlk_stop** 15
   - **hfwifi_scan** 15
7. UART API 17
   - **hfuart_close** 17
   - **hfuart_open** 17
   - **hfuart_send** 17
   - **hfuart_recv** 18
8. Timer API 19
   - **hftimer_start** 19
   - **hftimer_create** 19
   - **hftimer_change_period** 20
   - **hftimer_delete** 20
   - **hftimer_get_counter** 20
   - **hftimer_get_timer_id** 21
   - **hftimer_stop** 21
9. Multitask API 23
   - **PROCESS** 23
10. Network API 23
   - **hfnet_start_uart** 23
   - **hfnet_start_socketa** 24
   - **hfnet_start_socketb** 24
   - **hfnet_tcp_listen** 25
   - **hfnet_tcp_unlisten** 25
   - **hfnet_tcp_close** 26

- **hfnet_tcp_connect** 26
- **hfnet_tcp_disconnect**....27
- **hfnet_tcp_send**....27
- **hfnet_udp_create**....28
- **hfnet_udp_close**....28
- **hfnet_udp_sendto**....29

11. System Function 30
- **hfmem_free** 30
- **hfmem_malloc**....30
- **hfmem_realloc**....30
- **hfsys_get_reset_reason** 31
- **hfsys_get_run_mode**....32
- **hfsys_get_time** 32
- **hfsys_nvm_read** 32
- **hfsys_nvm_write** 33
- **hfsys_register_system_event** 33
- **hfsys_reload** 34
- **hfsys_reset**....34
- **hfsys_softreset** 35
- **hfsys_switch_run_mode** 35

12. User Flash API....37
- **hfuflash_erase_page** 37
- **hfuflash_read**....37
- **hfuflash_write** 37

13. User File API....39
- **hffile_userbin_read** 39
- **hffile_userbin_size** 39
- **hffile_userbin_write**....39
- **hffile_userbin_zero** 40

14. Auto-updgrade API....41
- **hfupdate_complete**....41
- **hfupdate_start**....41
- **hfupdate_write_file** 42

Appendix a：hardware timer 43

Appendix b：GPIO Interrupt....45

1. LINUX STANDARD C FUNCTION

HSF MC300 support standard c, such as memory management, string, time, stdio and so on. Refer to standard C for detail.

Note:

It is not allowed to call the standard libc memory management API in HSF MC300 system. Use the following to replace that hfmem_malloc, hfmem_free, hfmem_realloc. This API document is applicable for HF-MC300 SOC series modules(HF-LPB120, HF-LPT120, HF-LPT120G，HF-LPT220, HF-LPB125) and HF-SIP120 chip.

2. SYSTEM ERROR CODE DEFINATION

The return value of API function is as following, “HF_SUCCESS” or “>0” for success, “<0” for fail. The error code is 4Bytes signed integer value. The return value is the negative of error code. “31-24” bits for index, “23-8” bits is reserved and “7-0” bits is the 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*/
};
```

Header file:

hferrno.h

3. AT COMMAD API

- **hfat_get_words**

Function prototype:

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

Description:

Get all the response parameters of AT command

Parameters:

str: Pointer to the AT command response string(Ex. "+ok=WPA2PSK,AES,12345678"), the str pointed address should be in RAM area.

words: Pointer to the string value of each AT command response parameters

size: number of words.

Return Value:

<=0: The string of str pointed is not a valid AT command response.

>0: The number of words parsed.

Notes:

AT command use the folloing character separator ',', '=', ' ', "\r\n"

Examples:

Example/attest.c

Header file:

hfat.h

- **hfat_send_cmd**

Function prototype:

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

Description:

Send AT command. Response is saved in buffer.

Parameters:

cmd_line: AT command string,

Format is AT+CMD_NAME[=][arg,]…[argn], E.g "AT+WMODE\r\n"

cmd_len: Length of cmd_line including end character

rsp: The AT command response buffer

len: The response length

Return Value:

HF_SUCCESS:Set success,HF_FAIL:Set fail

Notes:

The execution of this API is the same as UART AT command, it does not support "AT+H" and "AT+WSCAN" now, Refer to hfwifi_scan for Wi-Fi scan application. The response of AT command is saved in rsp. See module user manual for detailed AT command. Use this API to get and set module parameters.

This API does not support AT command defined extended by user_define_at_cmds_table due to it can be direct called. If user extends the current AT command E.g "AT+VER", if use hfat_send_cmd("AT+VER\r\n", sizeof("AT+VER\r\n"),rsp,64), it will response with the system defined AT command, not the extended user defined AT command.

Examples:

example/attest.c

Header file:

hfat.h

4. DEBUG API

- **HF_Debug**

Function prototype:

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

Description:

Output debug information to debug UART

Parameters:

debug_level: Debug level, it can be as following or more.
#define DEBUG_LEVEL_LOW 1
#define DEBUG_LEVEL_MID 2
#define DEBUG_LEVEL_HI 3
It can also be set via hfdbg_set_level API
format: formated output, the same as printf, maximum 250 bytes length.

Return Value:

None

Notes:

Use AT+NDBGL command to enable or disable UART debug information output, see module user manual for detailed command description. The released software should close the debug information output..

Examples:

None

Header file:

hf_debug.h

- **hfdbg_get_level**

Function prototype:

int hfdbg_get_level ();

Description:

Get the current debug level;

Parameters:

None

Return Value:

Return the current debug level.

Notes:

None

Examples:

None

Header file:

hf_debug.h

- **hfdbg_set_level**

Function prototype:

void hfdbg_set_level (int debug_level);

Description:

Set or close debug level

Parameters:

debug_level: debug level, it can be 0(close), the following or more bigger like 10.
#define DEBUG_LEVEL_LOW 1
#define DEBUG_LEVEL_MID 2
#define DEBUG_LEVEL_HI 3

Return Value:

None

Notes:

None

Examples:

None

Header file:

hf_debug.h

5. GPIO CONTROL API

- **hfgpio_configure_fpin**

Function prototype:

int hfgpio_configure_fpin(int fid,int flags);

Description:

Configure the PIN according to fid(function id);

Parameters:

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_SLEEP_ON,
  HFGPIO_F_SLEEP_WPS,
  HFGPIO_F_SLEEP_IR,
  HFGPIO_F_RESERVE2,
  HFGPIO_F_RESERVE3,
  HFGPIO_F_RESERVE4,
  HFGPIO_F_RESERVE5,
  HFGPIO_F_USER_DEFINE
};
```

Fid can also be user defined. It should start from HFGPIO_F_USER_DEFINE.

flags: PIN attribute, it can be one or multiple of the following value(use '|' character).

HFPIO_DEFAULT	Default
HFM_IO_TYPE_INPUT	Output low
HFM_IO_OUTPUT_0	Output low
HFM_IO_OUTPUT_1	Output High

Return Value:

HF_SUCCESS:Set success,HF_E_INVAL: fid is invalid or PIN is invalid.
HF_E_ACCES: The corresponding PIN does not support the setting attribute(flags), E.g HFGPIO_F_JTAG_TCK is a peripheral PIN, not a GPIO, it should not be configred as HFM_IO_XXX except HFPIO_DEFAULT. .

Notes:

The PIN should support the attribute to be set. Otherwise it will return HF_E_ACCES.

Examples:

None

Header file:

hfgpio.h

- **hfgpio_fconfigure_get**

Function prototype:

int hfgpio_fset_out_high(int fid);

Description:

Set the fid mapping PIN output high.

Parameters:

fid: Function id, refer to HF_GPIO_FUNC_E, it can also be user defined fid。

Return Value:

HF_SUCCESS:Set success,HF_E_INVAL: fid or PIN is invalid,
HF_FAIL:Set fail；HF_E_ACCES:The pin can not be set as input

Notes:

This API is the same as hfgpio_configure_fpin(fid, HFM_IO_OUTPUT_1| HFPIO_DEFAULT);

Examples:

example/gpiotest.c

Header file:

hfgpio.h

- **hfgpio_fpin_add_feature**

Function prototype:

int HSF_API hfgpio_fpin_add_feature(int fid,int flags);

Description:

Add attribute to fid defined PIN.

Parameters:

fid: Function id, refer to HF_GPIO_FUNC_E, it can also be user defined fid.
flags: Refer to hfgpio_configure_fpin flags;

Return Value:

HF_SUCCESS:Set success, HF_E_INVAL: fid or PIN is invalid

Notes:

None

Examples:

None

Header file:

hfgpio.h

- **hfgpio_fpin_clear_feature**

Function prototype:

int HSF_API hfgpio_fpin_clear_feature (int fid,int flags);

Description:

Clear attribute of fid PIN

Parameters:

fid: Function id, refer to HF_GPIO_FUNC_E, it can also be user defined fid.

flags: Refer hfgpio_configure_fpin flags;

Return Value:

HF_SUCCESS:Set success, HF_E_INVAL: fid or PIN is invalid

Notes:

None

Examples:

None

Header file:

hfgpio.h

- **hfgpio_fpin_is_high**

Function prototype:

int hfgpio_fpin_is_high(int fid);

Description:

Judge the fid PIN voltage.

Parameters:

fid: Function id, refer to HF_GPIO_FUNC_E, it can also be user defined fid.
,the corresponding PIN must have F_GPO or F_GPI attribute

Return Value:

Return 0 if PIN is low, return 1 if PIN is high, reutrn <0 if PIN is illegal.

Notes:

None

Examples:

example/gpiotest.c

Header file:

hfgpio.h

- **hfgpio_fset_out_high**

Function prototype:

int hfgpio_fset_out_high(int fid);

Description:

Set the fid mapping PIN output high.

Parameters:

fid: Function id, refer to HF_GPIO_FUNC_E, it can also be user defined fid。

Return Value:

HF_SUCCESS:Set success,HF_E_INVAL: fid or PIN is invalid,

HF_FAIL:Set fail ； HF_E_ACCES:The pin can not be set as input

Notes:

This API is the same as hfgpio_configure_fpin(fid, HFM_IO_OUTPUT_1| HFPIO_DEFAULT);

Examples:

example/gpiotest.c

Header file:

hfgpio.h

- **hfgpio_fset_out_low**

Function prototype:

int hfgpio_fset_out_low(int fid);

Description:

Set fid mapping pin output low

Parameters:

fid: Function id, refer to HF_GPIO_FUNC_E, it can also be user defined fid.

Return Value:

HF_SUCCESS:Set success,HF_E_INVAL: fid or PIN is invalid.

Notes:

This API is the same as hfgpio_configure_fpin(fid, HFM_IO_OUTPUT_0| HFPIO_DEFAULT);

Examples:

example/gpiotest.c

Header file:

hfgpio.h

6. WIFI API

- **hfsmtlk_start**

Function prototype:

int HSF_API hfsmtlk_start(void);

Description:

Start smartlink.

Parameters:

None

Return Value:

Return HF_SUCCESS if success, return others if fail

Notes:

The system will software reset once call this API.

Examples:

None

Header file:

hfsmtlk.h

- **hfsmtlk_stop**

Function prototype:

int HSF_API hfsmtlk_stop(void);

Description:

Stop smartlink.

Parameters:

None

Return Value:

Return HF_SUCCESS if success, return others if fail

Notes:

None

Examples:

None

Header file:

hfsmtlk.h

- **hfwifi_scan**

Function prototype:

int HSF_API hfwifi_scan(hfwifi_scan_callback_t p_callback);

Description:

Scan the around AP

Parameters:

hfwifi_scan_callback_t: The callback function if the module scan out the AP information.

typedef int (*hfwifi_scan_callback_t)(PWIFI_SCAN_RESULT_ITEM);

typedef struct _WIFI_SCAN_RESULT_ITEM

```
{
   uint8_t auth; //Authentication
   uint8_t encry;//Encryption
   uint8_t channel;//AP Channel
   uint8_t rssi;//RSSI in percentage
   char ssid[32+1];//AP SSID
   uint8_t mac[6];//AP MAC
   int rssi_dbm;//The RSSI in dBm vale
   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
```

Return Value:

Return HF_SUCCESS if success, return others if fail

Notes:

Scan is finished if receive the NULL pointer callback.

Examples:

example/wifitest.c

Header file:

hfwifi.h

7. UART API

- **hfuart_close**

Function prototype:

hfuart_handle_t HSF_API hfuart_close(int uart_no);

Description:

close uart;

Parameters:

uart_no: UART channel number, 0 or 1.

Return Value:

Return HF_SUCCESS if success, return HF_FAIL if fail;

Notes:

Call hfuart_close to release resources if UART is no longer used.

Examples:

example/uarttest.c

Header file:

hfuart.h

- **hfuart_open**

Function prototype:

hfuart_handle_t HSF_API hfuart_open(int uart_no);

Description:

Opern UART pin

Parameters:

uart_no: UART channel number, 0 or 1;

Return Value:

Return hfuart_handle_t pointer if success, otherwise return NULL.

Notes:

Call hfuart_open before hfuart_recv.

Examples:

example/uarttest.c

Header file:

hfuart.h

- **hfuart_send**

Function prototype:

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

Description:

Send data to UART

Parameters:

huart: UART object, it should be created by hfuart_open, The hfnet_start_uart thread will automatically crate HFUART0 object for use.

data: Data buffer for send.

bytes: Data buffer length.

timeouts: Timeout, not used yet, reserved to 0.

Return Value:

Return the length of sent data, return error code if fail.

Notes:

None

Examples:

example/uarttest.c

Header file:

hfuart.h

- **hfuart_recv**

Function prototype:

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

Description:

Receive data from UART

Parameters:

huart: UART object.The object is created by hfuart_open.

recv: Received data buffer.

bytes: Received data buffer length.

timeouts: Receive data time out. It should be set as 0 when use select operation

Return Value:

Return the data length received.

Notes:

If the system comes with serial transparent transmission and command mode, please do not call this function.It may lead to the serial port transparent transmission and command mode exception.You can get serial callbacks by using hfnet_start_uart.

Examples:

example/uarttest.c

Header file:

hfuart.h

8. TIMER API

- **hftimer_start**

Function prototype:

int HSF_API hftimer_start(hftimer_handle_t htimer);

Description:

Start a timer

Parameters:

htimer: Timer object

Return Value:

Return HF_SUCCESS if success, return HF_FAIL if fail;

Notes:

None

Examples:

example/timertest.c

Header file:

hftimer.h

- **hftimer_create**

Function prototype:

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);

Description:

Create a timer.

Parameters:

name: Timer name

period: Timer period in ms

auto_reload: auto reload is enabled or disabled. If set it to true, only need to call hftimer_start once for start. If set it to false, once the time is up, need to call hftimer_start for restart the timer.

timer_id: the timer id used in the callback function when multiple timer uses the same callback function.

flags:0 for softwaretimer, 1 for hardware timer(HFTIMER_FLAG_HARDWARE_TIMER, hardware timer is not supported yet).

Return Value:

Return timer object if success, otherwise return NULL

Notes:

The timer won't start until call hftimer_start when create a timer object.

Examples:

example/timertest.c

Header file:

hftimer.h

- **hftimer_change_period**

Function prototype:

void HSF_API hftimer_change_period(hftimer_handle_t timer,int32_t new_period);

Description:

Change timer period.

Parameters:

htimer: Object created by hftimer_create

new_period: new period in ms. If create hardware timer, the unit is in us.

Return Value:

None;

Notes:

None

Examples:

example/timertest.c

Header file:

hftimer.h

- **hftimer_delete**

Function prototype:

void HSF_API hftimer_delete(hftimer_handle_t htimer);

Description:

Delete a timer object

Parameters:

htimer: The deleted timer object created by hftimer_create

Return Value:

None

Examples:

example/timertest.c

Header file:

hftimer.h

- **hftimer_get_counter**

Function prototype:

void HSF_API hftimer_get_counter (hftimer_handle_t htimer);

Description:

Get hadware timer counter value.

Parameters:

htimer: The timer object

Return Value:

Return the CLK counter time, the module current frequency is 48MHz, one clock time is 1/48 us.If time is up, return 0

Notes:

Use hardware timer for accurate timer..

Examples:

example/timertest.c

Header file:

hftimer.h

- **hftimer_get_timer_id**

Function prototype:

uint32_t HSF_API hftimer_get_timer_id(hftimer_handle_t htimer);

Description:

Get timer ID.

Parameters:

htimer: Timer object

Return Value:

Return timer ID, return HF_FAIL if failure.;

Notes:

This API is used in timer callback, to distinguish multiple timer uses the same timer callback function.

Examples:

example/timertest.c

Header file:

hftimer.h

- **hftimer_stop**

Function prototype:

void HSF_API hftimer_stop(hftimer_handle_t htimer);

Description:

Stop timer

Parameters:

htimer: Timer object.

Return Value:

None;

Notes:

The timer stop counting unless hftimer_start is recalled.

Examples:

example/timertest.c

Header file:
hftimer.h

9. MULTITASK API

- **PROCESS**

MC300 HSF uses Contiki OS, there is no thread concept. The task is switched by switch case sentence.The following rules should be followed.

1.switch/case is not allowed

2.Be careful to use local variable in process. The main task will return when execution. See contiki OS for detail.

PROCESS(name, strname)

Declare the main process function and named as name.

AUTOSTART_PROCESSES(...)

Define a process pointer array autostart_processe;

PROCESS_THREAD(name, ev, data)

Define or declare process name based on the marco ";" or "{}";

PROCESS_BEGIN()

Process start function;

PROCESS_EXIT()

Process end function

- **PROCESS_WAIT_EVENT_UNTIL(c)**

Wait for event

int process_post(struct process *p, process_event_t ev, void* data);

Send event to process;

void process_post_synch(struct process *p,process_event_t ev, void* data);

Send event to process and switch to that task immediately;

Examples:

example/processtest.c. Refer to contiki for more details.

Header file:

hsf.h

10. NETWORK API

- **hfnet_start_uart**

Function prototype:

int hfnet_start_uart(uint32_t uxpriority,hfnet_callback_t p_uart_callback);

Description:

Start the HSF default UART task.

Parameters:

uxpriority:uart service priority. Refer to hfthread_create Parametersuxpriority

p_uart_callback: UART callback, set to NULL if no need to use callback. The UART received data can be changed in the callback function.

Return Value:

Return HF_SUCCESS if success, otherwise return HF_FAIL.

Notes:

When UART receive data, if the callback is not NULL, the received UART data can be modified. When return length if module is in throughput mode, the data will be sent to socketa and socket b, if module is in command mode, it will be sent to command analysis program. The data can be also encrypted in the callback for some special application;

Examples:

example/callbacktest.c

Header file:

hfnet.h

- **hfnet_start_socketa**

Function prototype:

int hfnet_start_socketa(uint32_t uxpriority,hfnet_callback_t p_callback);

Description:

Launch build-in socketa service in HSF

Parameters:

uxpriority: Socketa service priority,please refer to hfthread_create

Parametersuxpriority;

p_callback:Callback function is alternative, if no needs,set the value NULL.It triggers when socketa service receives packes or state changes.

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

event: Event ID ;

data: Point to the data storing buffer, user can modify the value of buffer in callback function.If working in UDP mode,data+len after 6 bytes store 4 Bytes IP address and 2 Bytes port number on the sending side.If socketa working under TCP-server mod, data+len after 4 Bytes is cid of server.You can use hfnet_socketa_get_client to get detailed introduction.

len: The length of received data;

buf_len: Data points to actual length of buffer,the value is greater than or

equal to len;Callback function Return Value is the length of processed data.If user only read the data, not modify, the return value should be len;

Return Value:

Success returns HF_SUCCESS, HF_FAIL indicates failure

Notes:

When socketa service receive the data from the internet,call for p_callback and then send the processed to serial port.User can analyze the received data by p_callback,or double process,such as encryption and decryption.It return the data back to socketa service.

Examples:

example/callbacktest.c

Header file:

Hfnet.h

- **hfnet_start_socketb**

Function prototype:

int hfnet_start_socketb(uint32_t uxpriority,hfnet_callback_t p_callback);

Description:

Launch build-in socketb service in HSF

Parameters:

uxpriority:Socketb service priority,please refer to hfthread_create ***Parametersuxpriority;***
p_callback:Alternatively, do not use callbacks pass NULL, please refer to hfnet_start_socketa

Return Value:

Success returns HF_SUCCESS, HF_FAIL indicates failure

Notes:

None

Examples:

example/callbacktest.c

Header file:

hfnet.h

- **hfnet_tcp_listen**

Function prototype:

int HSF_API hfnet_tcp_listen(struct tcp_socket *socket);

Description:

Create TCP Serer, allow for TCP client to connect

Parameters:

socket:TCP Socket:
listen_port:listen port;
recv_callback:receive data callback.;
accept_callback:accept callback
send_callback:send data callback
close_callback:close connction callback
recv_data_maxlen:Received data buffer length.(default 2028 if no setting);

Return Value:

HF_SUCCESS if success, HF_FAIL if fail

Notes:

None。

Examples:

example/tcpservertest.c

Header file:

hfnet.h

- **hfnet_tcp_unlisten**

Function prototype:

int HSF_API hfnet_tcp_unlisten(struct tcp_socket *socket);

Description:

Close TCP Serer。

Parameters:

socket:TCP Socket struct,the same as created TCP Server socket

Return Value:

HF_SUCCESS if success,HF_FAIL if fail

Notes:

The resource will be released after close, if need to use again, call hfnet_tcp_listen to create a new TCP Server

Examples:

example/tcpservertest.c

Header file:

hfnet.h

- **hfnet_tcp_close**

Function prototype:

int HSF_API hfnet_tcp_close(NETSOCKET socket_id);

Description:

Close the TCP Client connected to TCP server

Parameters:

socket_id:TCP Client socket index.

Return Value:

HF_SUCCESS if success, HF_FAIL if fail

Notes:

After close, the TCP Server still may access new TCP client.

Examples:

example/tcpservertest.c

Header file:

hfnet.h

- **hfnet_tcp_connect**

Function prototype:

NETSOCKET HSF_API hfnet_tcp_connect(struct tcp_socket *socket);

Description:

Create a TCP Client

Parameters:

socket:TCP Socket
l_port:local port;
r_ip:remote IP;
r_port:remote port
recv_callback:receive data callback;

connect_callback:connection callback;
send_callback:data sent callback
close_callback:connection close callback
recv_data_maxlen:Received data buffer length.(default 2028 if no setting）;

Return Value:
Socket index.

Notes:
None

Examples:
example/tcpclienttest.c

Header file:
hfnet.h

- **hfnet_tcp_disconnect**

Function prototype:
int HSF_API hfnet_tcp_disconnect(NETSOCKET socket_id);

Description:
Close TCP Connection.

Parameters:
socket_id:Socket index;

Return Value:
HF_SUCCESS if success, HF_FAIL if fail

Notes:
None
Examples:
example/tcpclienttest.c

Header file:
hfnet.h

- **hfnet_tcp_send**

Function prototype:
int HSF_API hfnet_tcp_send(NETSOCKET socket_id, char *data, unsigned short datalen);
Description:
Send TCP data。

Parameters:
socket_id:Socket index
data:data sent
datalen: data sentlength;

Return Value:
HF_SUCCESS if success, HF_FAIL if fail

Notes:

Return success only means the data is sent to sent queue, the send_callback function will be called if the data is sent successfully.

Examples:

example/tcpclienttest.c

Header file:

hfnet.h

- **hfnet_udp_create**

Function prototype:

NETSOCKET HSF_API hfnet_udp_create(struct udp_socket *socket);

Description:

Create a UDP

Parameters:

UDP Socket Struct:
l_port:local port
recv_callback:receive data callback
connect_callback:connect callback
recv_data_maxlen:receive data maximum length（default 2048）;

Return Value:

Socket Index

Notes:

None

Examples:

example/udptest.c

Header file:

hfnet.h

- **hfnet_udp_close**

Function prototype:

int HSF_API hfnet_udp_close(NETSOCKET socket_id);

Description:

Close a UDP

Parameters:

socket_id: socket index

Return Value:

HF_SUCCESS if success,HF_FAIL if fail

Notes:

None

Examples:

example/udptest.c

Header file:

hfnet.h

- **hfnet_udp_sendto**

Function prototype:

int HSF_API hfnet_udp_sendto(NETSOCKET socket_id, char *data, unsigned short datalen,uip_ipaddr_t *peeraddr, unsigned short peerport);

Description:

Send UDP Data

Parameters:

socket_id:Socket index
data:Sent data
datalen:Sent data length
peeraddr:Remote IP
peerport:Remote port

Return Value:

HF_SUCCESS if success,HF_FAIL if fail.

Notes:

The data is sent out if success, it won't call send callback. The data buffer can be released if send OK.

Examples:

example/udptest.c

Header file:

hfnet.h

11. SYSTEM FUNCTION

- **hfmem_free**

Function prototype:

void HSF_API hfmem_free(void *pv);

Description:

Free the memory allocated by hfsys_malloc

Parameters:

pv: Pointer to the memory varible need to be free.

Return Value:

None

Notes:

Do not use libc free function.

Examples:

None

Header file:

hfsys.h

- **hfmem_malloc**

Function prototype:

void *hfmem_malloc(size_t size)

Parameters:

Allocate memory

Parameters:

size: memory size

Return Value:

Return RAM address if success, otherwise return NULL;

Notes:

Do not call libc malloc.

Header file:

hfsys.h

- **hfmem_realloc**

Function prototype:

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

Description:

Reallocate RAM resource

Parameters:

pv: RAM pointer allocated by hfmem_malloc before

size: The new RAM size

Return Value:

None

Notes:

Do not call libc realloc.

Examples:

None

Header file:

hfsys.h

- **hfsys_get_reset_reason**

Function prototype:

uint32_t HSF_API hfsys_get_reset_reason (void);

Description:

Get module reboot reason.

Parameters:

None

Return Value:

REturn reboot reason. It can be the following one or more..

HFSYS_RESET_REASON_NORMAL	Caused by power on/off
HFSYS_RESET_REASON_ERESET	Caused by hardware watchdog or external reset PIN
HFSYS_RESET_REASON_IRESET0	Caused by hfsys_softreset API (Software watchdog reset, RAM accress error will all call this API)
HFSYS_RESET_REASON_IRESET1	Caused by hfsys_reset API
HFSYS_RESET_REASON_WPS	Caused by WPS start(Reserved)
HFSYS_RESET_REASON_SMARTLINK_START	Caused by Smartlink start
HFSYS_RESET_REASON_SMARTLINK_OK	Caused by Smartlink finished
HFSYS_RESET_REASON_WPS_OK	Caused by WPS finished.(Reserved)

Notes:

Usually call this to do special operation due to different reboot reason.。

Examples:

None

Header file:

hfsys.h

- **hfsys_get_run_mode**

Function prototype:

int hfsys_get_run_mode()

Description:

Get system run mode(AT+TMODE)

Parameters:

None

Return Value:

It can be the following mode:

enum HFSYS_RUN_MODE_E

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

Header file:

hfsys.h

- **hfsys_get_time**

Function prototype:

uint32_t HSF_API hfsys_get_time (void);

Description:

Get system running time in ms

Parameters:

None

Return Value:

Return the OS running time in ms

Notes:

None

Examples:

None

Header file:

hfsys.h

- **hfsys_nvm_read**

Function prototype:

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

Description:

Read data from NVM

Parameters:

nvm_addr:NVM address, which can be (0-99);

buf:Save the read data from NVM into buffer;

length:Sum of length and nvm_addr is less than 100;

Return Value:

Success returns HF SUCCESS, otherwise the return value is less than zero

Notes:

When the module restart or soft reset, NVM data will not be cleared.It provides 100 bytes of NVM.If modele powers off,the data of NVM will not be cleared.

Examples:

None

Header file:

hfsys.h

- **hfsys_nvm_write**

Function prototype:

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

Description:

Write data into NVM

Parameters:

nvm_addr:NVM address, which can be (0-99);
buf: Save the read data from NVM into buffer;
length: Sum of length and nvm_addr is less than 100;

Return Value:

Success returns HF SUCCESS, otherwise the return value is less than zero.

Notes:

When the module restart or soft reset, NVM data will not be cleared.It provides 100 bytes of NVM.If modele powers off,the data of NVM will not be cleared.

Examples:

None

Header file:

hfsys.h

- **hfsys_register_system_event**

Function prototype:

int HSF_API hfsys_register_system_event(hfsys_event_callback_t p_callback);

Description:

Register system event callback

Parameters:

p_callback: Point to the callback function when event occures.;

Return Value:

Return HF_SUCCESS if success, otherwise return HF_FAIL.

Notes:

The time consuming operation is not allowed in the callback function, the callback function should immediate return after process.The support event is as following

HFE_WIFI_STA_CONNECTED	When STA connect to AP
HFE_WIFI_STA_DISCONNECTED	When STA disconnect to AP
HFE_CONFIG_RELOAD	When reload is execute.(nReload Pin or AT+RELD)
HFE_DHCP_OK	When STA connect to AP and get DHCP IP address from AP 当 STA
HFE_SMTLK_OK	When Smartlink get AP password, the default operation is reboot, if the callback return value is not HF_SUCCESS, the module won't do reboot operation, user need to reboot manually.

Examples:

example/tcpclienttest.c

Header file:

hfsys.h

- **hfsys_reload**

Function prototype:

void HSF_API hfsys_reload()；

Description:

Restore the parameter to factory setting

Parameters:

None

Return Value:

None

Notes:

None

Examples:

None

Header file:

hfsys.h

- **hfsys_reset**

Function prototype:

void HSF_API hfsys_reset(void);

Description:

Hardware reset, the IO status is lost.

Parameters:

None

Return Value:

None

Notes:

None

Examples:

None

Header file:

hfsys.h

- **hfsys_softreset**

Function prototype:

void HSF_API hfsys_softreset(void);

Description:

Software reset, keep the current IO status

Parameters:

None

Return Value:

None

Notes:

None

Examples:

None

Header file:

hfsys.h

- **hfsys_switch_run_mode**

Function prototype:

int hfsys_switch_run_mode(int mode);

Description:

Switch system running mode.

Parameters:

mode: The following mode is supported.

enum HFSYS_RUN_MODE_E

{

HFSYS_STATE_RUN_THROUGH=0,

```
    HFSYS_STATE_RUN_CMD=1,
    HFSYS_STATE_MAX_VALUE
};
```

HFSYS_STATE_RUN_THROUGH:Throughput mode
HFSYS_STATE_RUN_CMD:Command mode

Return Value:

HF_SUCCESS: success, otherwise HF_FAIL

Header file:

hfsys.h

12. USER FLASH API

- **hfuflash_erase_page**

Function prototype:

int HSF_API hfuflash_erase_page(uint32_t addr, int pages);

Description:

Erase user flash page.

Parameters:

addr: logical address of user flash(not the flash real address).
pages : The page number need to be erased.

Return Value:

Return HF_SUCCESS if success, otherwise return HF_FAIL;

Notes:

The use flash is a 128KB size of flash in the reserved flash real area.

Examples:

example/uflashtest.c

Header file:

hfflash.h

- **hfuflash_read**

Function prototype:

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

Description:

Read data from flash.

Parameters:

addr: The logical address of flash(0- HFUFLASH_SIZE-2)；
data : The received data buffer.
len : The data buffer length;

Return Value:

Return the bytes number read if success, otherwise return <0

Notes:

None

Examples:

example/uflashtest.c

Header file:

hfflash.h

- **hfuflash_write**

Function prototype:

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

Description:

Write data to flash

Parameters:

addr: The logical address of flash(0- HFUFLASH_SIZE-2)；

data : Data buffer；

len : Data buffer length;

Return Value:

the bytes number if write success, otherwise return <0;

Notes:

Need to erase the flash page if the address to be written has previous data in it. The data buffer should be in RAM area, not in ROM. See the following example.:

Error 1:”Test” is in ROM area.

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

Error2:const varible is in ROM area..

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

Correct:

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

Examples:

example/uflashtest.c

Header file:

hfflash.h

13. USER FILE API

- **hffile_userbin_read**

Function prototype:

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

Description:

Read data from user files;

Parameters:

offset: File offset;
data : Save the data from read file to buffer;
len : Size of the buffer;

Return Value:

If return value is less than zero,then it fails.Otherwise,the function returns the number of actual Byte read from the file;

Examples:

None

Header file:

hffile.h

- **hffile_userbin_size**

Function prototype:

int HSF_API hffile_userbin_size(void);

Description:

Read size from user bin's file;

Parameters:

None

Return Value:

Failure is less than zero, otherwise the file size;

Notes:

None

Examples:

None

Header file:

hffile.h

- **hffile_userbin_write**

Function prototype:

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

Description:

Write the data into user file.

Parameters:

offset: File offset;

data : Save the data from read file to buffer;

len : Size of the buffer;

Return Value:

If return value is less than zero,then it fails.Otherwise,the function returns the number of actual Byte written into the file;

Notes:

A user profile is a fixed-size file, the file is stored in flash, you can save user data.User profile has backup function, so users do not need to worry about power outages during programming.If it powers off, it will automatically revert to the content before.

Examples:

None

Header file:

hffile.h

- **hffile_userbin_zero**

Function prototype:

int HSF_API hffile_userbin_zero (void);

Description:

Quickly clear the content of the entire file.

Parameters:

None

Return Value:

Failure is less than zero, otherwise the file size;

Notes:

Calling this function can quickly clear up the entire contnet of file, faster than hffile_userbin_write

Examples:

None

Header file:

hffile.h

14. AUTO-UPDGRADE API

- **hfupdate_complete**

Function prototype:

int hfupdate_complete(HFUPDATE_TYPE_E type,uint32_t file_total_len);

Description:

Upgrade finished

Parameters:

type:upgrade type

file_total_len: upgrade file length

Return Value:

HF_SUCCESS if success, HF_FAIL if fail

Notes:

When the upgrade file has been download into the module, call this function to do the upgrade process(The module need to reboot manually)

Examples:

example/updatetest.c

Header file:

hfupdate.h

- **hfupdate_start**

Function prototype:

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

Description:

Write data to flash

Parameters:

addr: The logical address of flash(0- HFUFLASH_SIZE-2)；

data : Data buffer；

len : Data buffer length;

Return Value:

Return the bytes number if write success, otherwise return <0;

Notes:

Need to erase the flash page if the address to be written has previous data in it. The data buffer should be in RAM area, not in ROM. See the following example.:

Error 1:”Test” is in ROM area.

hfuflash_write (Offset,"Test",4);

Error2:const varible is in ROM area..

const uint8_t Data[] = "Test";

hfuflash_write (Offset,Offset,Data,4);

Correct:

Uint8_t Data[]=”Test”;

hfuflash_write (Offset,Offset,Data,4);

Examples:

example/uflashtest.c

Header file:

hfupdate.h

- **hfupdate_write_file**

Function prototype:

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

Description:

Copy the upgrade file data to upgrade backup flash area.

Parameters:

type: type
offset: The upgrade file offet address
data: the upgrade file data
len: The upgrade file length

Return Value:

>=0 for success, otherwise return HF_FAIL.

Notes:

HFUPDATE_SW is supported currently.

Examples:

example/updatetest.c

Header file:

hfupdate.h

APPENDIX A：HARDWARE TIMER

There is 5 hardware timer in all for MC300 Series, 1 for us level and 4 for ms level timer.

Timer Head file：drv_timer.h

TimerID:

5 timer definition: US_xxx is for us Level timer. MS_xxx for ms level timer. US_TIMER2 is used by OS. MS_TIMER1 is used by OS WatchDog.

```
#define US_TIMER0                    (TU0_US_REG_BASE + 0)
#define US_TIMER2                    (TU0_US_REG_BASE + 0x20)
#define MS_TIMER1                    (TM0_MS_REG_BASE + 0x10)
#define MS_TIMER2                    (TM0_MS_REG_BASE + 0x20)
#define MS_TIMER3                    (TM0_MS_REG_BASE + 0x30)
```

Timer Start API

```
int hwtmr_start(HW_TIMER *tmr, unsigned int count, irq_handler tmr_handle,
                    void *m_data, enum hwtmr_op_mode mode);
```

Parameters：

tmr: is TimerID，US_xxx or MS_xxx

count: is timer value, the maximum is 0xFFFF, so for us level timer the maximum is 65ms, for ms level timer the maximum is 65s.

tmr_handle： is the interrupt callback, it should be put in RAM for fast execute, the execution time shoud be very fast to quit the callback, block operation is not allowed.

m_data： tmr_handle parameters.

mode：HTMR_ONESHOT for single timer callback, HTMR_PERIODIC for cycle timer callback.

Recommend to use US_TIMER0, MS_TIMER2，MS_TIMER3. and the ms level timer value should be more than 100ms.

example

```
// start timer.
static void test_timer_start(void)
{
    u_printf("To init timer...");
    // MS_TIMER2 setting 60s
    hwtmr_start(MS_TIMER2,60000,ms_timer_callback,NULL, HTMR_PERIODIC);
```

```
	// US_TIMER0 setting 50ms
	hwtmr_start(US_TIMER0,50000,us_timer_callback,NULL, HTMR_PERIODIC);
}

// MS_TIMER2 interrupt.
ATTRIBUTE_SECTION_KEEP_IN_SRAM static void ms_timer_callback( void *arg )
{
	u_printf("1");
}

// US_TIMER0 interrupt.
ATTRIBUTE_SECTION_KEEP_IN_SRAM static void us_timer_callback( void *arg )
{
	static int state=0;
	// disable interrupt
	irq_mask_disable(IRQ_US_TIMER0);
	if (state==0)
	{
		drv_gpio_write(GPIO_18,1);
		drv_gpio_Output(GPIO_18,0,0);
		state=1;
	}
	else
	{
		drv_gpio_write(GPIO_18,0);
		drv_gpio_Output(GPIO_18,0,0);
		state=0;
	}
	// enable interrupt
	irq_mask_enable(IRQ_US_TIMER0);
}
```

APPENDIX B：GPIO INTERRUPT

GPIO　header file：

drv_gpio.h
gpio_api.h

GPIO interrupt initialize API.
S32 gpio_irq_enable(GPIO_ID id, GPIO_TRIGGER_MODE mode,
void (*callbackfn)handle, void *data);

Parameters：

id: GPIOID which is defined in drv_gpio.

```
        typedef enum t_GPIO_ID
{
            GPIO_1      = 0,
            GPIO_2,
            GPIO_3,
            GPIO_5,
            GPIO_6,
            GPIO_8,
            GPIO_15,
            GPIO_18,
            GPIO_19,
            GPIO_20,
            GPIO_MAX
} GPIO_ID;
```

mode：interrupt mode, only support RISING_EDGE.

handle：interrupt function, it should be put in RAM. The execution time should be as short as be possible.

data：handle parameters

Note: the GPIO interrupt function can only be one, multiple GPIO interrupt all use the same one and use the parameter data to distinguish which GPIO.

Example：

```
// Initialize GPIO interrupt
void gpio_interrupt_init()
{
    u_printf("To init gpio interrupt...\n");
    gpio_irq_enable(GPIO_2,RISING_EDGE,gpio_interrupt_cb,(void*)GPIO_2);
```

```
}
// GPIO interrupt
ATTRIBUTE_SECTION_KEEP_IN_SRAM static void gpio_interrupt_cb(void *data)
{
	static int state=0;
	irq_mask_disable(IRQ_GPI0);
	u_printf("1");
	if (state==0)
	{
		drv_gpio_write(GPIO_18,1);
		drv_gpio_Output(GPIO_18,0,0);
		state=1;
	}
	else
	{
		drv_gpio_write(GPIO_18,0);
		drv_gpio_Output(GPIO_18,0,0);
		state=0;
	}
irq_mask_enable(IRQ_GPI0);
}

// key jitter process.
ATTRIBUTE_SECTION_KEEP_IN_SRAM static void gpio_interrupt_cb(void *data)
{
	irq_mask_disable(IRQ_GPI0);
	//create 50 ms timer
	hwtmr_stop(US_TIMER0);
	hwtmr_start(US_TIMER0,50000,us_timer_callback,NULL,HTMR_ONESHOT);
irq_mask_enable(IRQ_GPI0);
}

// US_TIMER0 timer interrupt
ATTRIBUTE_SECTION_KEEP_IN_SRAM static void us_timer_callback( void *arg )
{
	static int state=0;
	// disable interrupt
	irq_mask_disable(IRQ_US_TIMER0);
	if (state==0)
	{
		drv_gpio_write(GPIO_18,1);
		drv_gpio_Output(GPIO_18,0,0);
		state=1;
	}
```

```
    else
    {
        drv_gpio_write(GPIO_18,0);
        drv_gpio_Output(GPIO_18,0,0);
        state=0;
    }
    // enable interrupt
    irq_mask_enable(IRQ_US_TIMER0);
}
```

Long key press example

```
static int long_press=0;

ATTRIBUTE_SECTION_KEEP_IN_SRAM static void ms_timer_callback( void *arg )
{
    static int count=0;
    irq_mask_disable(IRQ_MS_TIMER2);
    if (drv_gpio_read(GPIO_2)==0)
    {
        if (long_press>0)
        {
            if (count==0)
            {
                u_printf("lp:%d s\n", long_press);
            }
            count++;
            if (count>10)          // 1s
            {
                count=0;
                long_press++;
            }
        }
        else
        {
            count++;
            if (count>=10)        // 1s
            {
                long_press=1;
                count=0;
            }
        }
    }
    else
    {
```

```
            count=0;
        }
        irq_mask_enable(IRQ_MS_TIMER2);
}

static void test_timer_start(void)
{
        // MS_TIMER2    100ms
        hwtmr_start(MS_TIMER2,100,ms_timer_callback,NULL, HTMR_PERIODIC);
}

ATTRIBUTE_SECTION_KEEP_IN_SRAM static void gpio_interrupt_cb(void *data)
{
        irq_mask_disable(IRQ_GPI0);
        // 50 ms timer
        hwtmr_stop(US_TIMER0);
        hwtmr_start(US_TIMER0,50000,us_timer_callback,NULL,HTMR_ONESHOT);
irq_mask_enable(IRQ_GPI0);
}

// US_TIMER0 interrupt
ATTRIBUTE_SECTION_KEEP_IN_SRAM static void us_timer_callback( void *arg )
{
        static int state=0;
        irq_mask_disable(IRQ_US_TIMER0);
        if (long_press==0)
                u_printf("sp\n");
        else
                u_printf("lp rls\n");
        irq_mask_enable(IRQ_US_TIMER0);
}

void gpio_interrupt_init()
{
        gpio_irq_enable(GPIO_2,RISING_EDGE,gpio_interrupt_cb,(void*)GPIO_2);
}
```