API function user manual | About

API Function User Manual

--Total 278 functions

-- Marcus collected and compiled 2014-07-28

About this handbook | Main page

Copyright Declaration(COPYRIGHT)
This manual was compiled by High Flying Electronics Technology.
You can use,dispatch the manual freely. Any unauthourized revise, decompilation on part or all of the manual for commmercial purpose is not allowed.
The manual is under the protection of copyright law and international conventions. High Flying reserves the final revise right to this manual and declaration.
The manual is based on Mainsoft(R) HTML Help format.
If any suggestions,please contact Marcus

version update(UPDATE)
The current manual version is July 28th 2014.Any revise or update after this time will be updated in the new version.
You can write email to me 关于汉枫(About Hi-Flying)

Any questions, please add QQ number: 1341369633

...hfat get words

---

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

Definition:
Get AT command or each respond parameter value

Paramter:
str:point to AT command request or feedback; related RAM address must be able to read and write;
words: reserve each parameter value;
size:number of word

Feedback value:
<=0 str relative charactor string is not correct AT command or illegal response; >0 the number of Word in relative charactor string;

Remark:
AT command divided by ”,”,”=”,” “,”\r\n”

Example:

Marcus 2014 ShangHai

Demand: The header file: hfgpio.h
The library: libKernel.a
HSF version demand: V1.0以上
Hardware: LPBXX

...hfat send cmd

---

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

Definition:
Send AT commend, the result will feedback to appointed buffer.

Parameter:
cmd_line: include AT command charactor string, format is AT+CMD_NAME[=][arg,]…[argn]
cmd_len: the length of cmd_line, include ending charactor;
rsp: the buffer to reserve AT command execution result;
len:the length of rsp;

feedback value:
HF_SUCCESS:set succeed , HF_FAIL: execution failed

Remark:
Function execution is the same with send AT command via serial, currently don;t support "AT+H" and "AT+WSCAN"; for wifi scan please refer to hfwifi_scan ,AT command result reserved in rsp, rsp is a charactor string,for specific format please refer to AT commmand user manual; via this function, user can get the system configuration setting.
This function can not send the AT command extended from user_define_at_cmds_table, since the AT command extended from itself can call directly, no need to realize via send AT command. If user extended existed AT command such as "AT+VER" from user_define_at_cmds_table such as, send hfuart_send(“AT+VER\r\n”, sizeof ("AT+VER\r\n"),rsp,64);
it will feedback its own AT+VER but not the extended one.

Example:
refer to example/at test

Marcus 2014 ShangHai

Demand: The header file: hfgpio.h
The library: libKernel.a
HSF version demand: V1.0以上
Hardware: LPBXX

...HF Debug

---

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

Definition:
output debug information to serial port

Parameter:
debug_level:debug level can be
#define DEBUG_LEVEL_LOW 1
#define DEBUG_LEVEL_MID 2
#define DEBUG_LEVEL_HI 3
can set debug level via hfdbg_set_level;
format: format output, same as printf;

feedback value:
none

feedback value:
for device without debug serial, debug onformation output to the serial for AT command, so after debug, user should close debug;user can open debug via AT+NDBGL=level and close via AT+NDBGL=0.

Example:
None

Marcus 2014 ShangHai

Demand: The header file: hf_debug.h
The library: libKernel.a
HSF version demand: V1.0以上
Hardware: LPBXX

...hfdbg get level

---

int hfdbg_get_level ();

Definition:
get the current debug level

Parameter:
none

Feedback value:
feedback the current debug level

Remark:
None

Remark:
None

Marcus 2014 ShangHai

Demand: The header file: hf_debug.h
The library: libKernel.a
HSF version demand: V1.0以上
Hardware: LPBXX

...hfdbg set level

---

void hfdbg_set_level (int debug_level);

Definition:
set debug level or close debug

Parameter:
debug_level:debug level can be
#define DEBUG_LEVEL_LOW 1
#define DEBUG_LEVEL_MID 2
#define DEBUG_LEVEL_HI 3

Feedback value:
none

Remark:
none

Example:
none

Marcus 2014 ShangHai

Demand: The header file: hf_debug.h
The library: libKernel.a
HSF version demand: V1.0以上
Hardware: LPBXX

AT Test

```
#include
#include
#include
#include
#include
#include "../example.h"
#include

#if (EXAMPLE_USE_DEMO==USER_AT_CMD_DEMO)

#define HFGPIO_F_ADC_CHANNEL1                   HFGPIO_F_USER_DEFINE
#define HFGPIO_F_ADC_CHANNEL2                   (HFGPIO_F_USER_DEFINE+1)
#define HFGPIO_F_ADC_CHANNEL3                   (HFGPIO_F_USER_DEFINE+2)

const int hf_gpio_fid_to_pid_map_table[HFM_MAX_FUNC_CODE]=
{
        HF_M_PIN(2),        //HFGPIO_F_JTAG_TCK
        HF_M_PIN(3),        //HFGPIO_F_JTAG_TDO
        HF_M_PIN(4),        //HFGPIO_F_JTAG_TDI
        HF_M_PIN(5),        //HFGPIO_F_JTAG_TMS
        HFM_NOPIN,                  //HFGPIO_F_USBDP
        HFM_NOPIN,                  //HFGPIO_F_USBDM
        HF_M_PIN(39),       //HFGPIO_F_UART0_TX
        HF_M_PIN(40),       //HFGPIO_F_UART0_RTS
        HF_M_PIN(41),       //HFGPIO_F_UART0_RX
        HF_M_PIN(42),       //HFGPIO_F_UART0_CTS

        HF_M_PIN(27),       //HFGPIO_F_SPI_MISO
        HF_M_PIN(28),       //HFGPIO_F_SPI_CLK
        HF_M_PIN(29),       //HFGPIO_F_SPI_CS
        HF_M_PIN(30),       //HFGPIO_F_SPI_MOSI

        HFM_NOPIN,          //HFGPIO_F_UART1_TX,
        HFM_NOPIN,          //HFGPIO_F_UART1_RTS,
        HFM_NOPIN,          //HFGPIO_F_UART1_RX,
        HFM_NOPIN,          //HFGPIO_F_UART1_CTS,

        HF_M_PIN(43),       //HFGPIO_F_NLINK
        HF_M_PIN(44),       //HFGPIO_F_NREADY
        HF_M_PIN(45),       //HFGPIO_F_NRELOAD
        HF_M_PIN(7),        //HFGPIO_F_SLEEP_RQ
        HF_M_PIN(8),        //HFGPIO_F_SLEEP_ON

        HFM_NOPIN,                  //HFGPIO_F_RESERVE0
        HFM_NOPIN,                  //HFGPIO_F_RESERVE1
```

```
        HFM_NOPIN,                  //HFGPIO_F_RESERVE2
        HFM_NOPIN,                  //HFGPIO_F_RESERVE3
        HFM_NOPIN,                  //HFGPIO_F_RESERVE4
        HFM_NOPIN,                  //HFGPIO_F_RESERVE5

        HF_M_PIN(11),       //HFGPIO_F_ADC_CHANNEL1
        HF_M_PIN(12),       //HFGPIO_F_ADC_CHANNEL2
        HF_M_PIN(23), //HFGPIO_F_ADC_CHANNEL3
};

static int hf_atcmd_myatcmd(pat_session_t s,int argc,char *argv[],char *rsp,int len);
static int hf_atcmd_adctest(pat_session_t s,int argc,char *argv[],char *rsp,int len);

static int USER_FUNC test_httpc_get(char *purl);
static int USER_FUNC test_httpc_post(char *purl);

const hfat_cmd_t user_define_at_cmds_table[]=
{
        {"UMYATCMD",hf_atcmd_myatcmd,"   AT+UMYATCMD=code\r\n",NULL},
        {"ADCTEST",hf_atcmd_adctest,"   AT+ADCTEST=code,channel\r\n",NULL},
        {NULL,NULL,NULL,NULL} //the last item must be null
};

static uint32_t adc_fid =0;

static int hf_atcmd_adctest(pat_session_t s,int argc,char *argv[],char *rsp,int len)
{
        int code=0;
        uint16_t value;

        if(argc<2)
                return -1;

        code = atoi(argv[0]);
        adc_fid = atoi(argv[1]);

        switch(code)
        {
                case 0:
                        hfgpio_adc_enable(adc_fid);
                        return 0;
                case 1:
                        value = hfgpio_adc_get_value(adc_fid);
                        break;
                case 2:
                        value = hfgpio_adc_get_voltage(adc_fid);
                        break;
                default:
                        return -1;
        }

        sprintf(rsp,"=%d",value);

        return 0;

}


static int USER_FUNC hf_atcmd_myatcmd(pat_session_t s,int argc,char *argv[],char *rsp,int len)
{
        static int test_code=0;

        if(argc<2)
        {
                return -1;
        }

        if(argc==0)
                sprintf(rsp,"=%d",test_code);
        else
        {
                test_code=atoi(argv[0]);
        }
        switch(test_code)
        {
                case 1:
                        test_httpc_get(argv[1]);
                        break;
                case 2:
                        test_httpc_post(argv[1]);
                        break;
                default:
                        break;
        }

        return 0;
}

static int USER_FUNC socketa_recv_callback(uint32_t event,char *data,uint32_t len,uint32_t buf_len)
```

```
{
        return len;
}

static int USER_FUNC socketb_recv_callback(uint32_t event,char *data,uint32_t len,uint32_t buf_len)
{
        return len;
}

static int USER_FUNC uart_recv_callback(uint32_t event,char *data,uint32_t len,uint32_t buf_len)
{
        return len;
}

void USER_FUNC update_timer_callback( hftimer_handle_t htimer )
{
        if(hfgpio_fpin_is_high(HFGPIO_F_NREADY))
                hfgpio_fset_out_low(HFGPIO_F_NREADY);
        else
                hfgpio_fset_out_high(HFGPIO_F_NREADY);
}

static int USER_FUNC test_httpc_get(char *purl)
{
        httpc_req_t  http_req;
        char *content_data=NULL;
        char *temp_buf=NULL;
        parsed_url_t url={0};
        http_session_t hhttp=0;
        int total_size,read_size=0;
        int rv=0;
        tls_init_config_t  *tls_cfg=NULL;
        char *test_url=purl;
        hftimer_handle_t upg_timer=NULL;
        struct MD5Context md5_ctx;
        uint8_t digest[16]={0};

        bzero(&http_req,sizeof(http_req));
        http_req.type = HTTP_GET;
        http_req.version=HTTP_VER_1_1;

        if((temp_buf = (char*)hfmem_malloc(256))==NULL)
        {
                u_printf("no memory\n");
                rv= -HF_E_NOMEM;
                goto exit;
        }
        bzero(temp_buf,sizeof(temp_buf));

        if((rv=hfhttp_parse_URL(test_url,temp_buf , 256, &url))!=HF_SUCCESS)
        {
                goto exit;
        }

        if((rv=hfhttp_open_session(&hhttp,test_url,0,tls_cfg,3))!=HF_SUCCESS)
        {
                u_printf("http open fail\n");
                goto exit;
        }

        hfsys_disable_all_soft_watchdogs();
        hfupdate_start(HFUPDATE_SW);
        http_req.resource = url.resource;
        hfhttp_prepare_req(hhttp,&http_req,HDR_ADD_CONN_CLOSE);
        hfhttp_add_header(hhttp,"Range","bytes=0");
        if((rv=hfhttp_send_request(hhttp,&http_req))!=HF_SUCCESS)
        {
                u_printf("http send request fail\n");
                goto exit;
        }

        content_data = (char*)hfmem_malloc(256);
        if(content_data==NULL)
        {
                rv= -HF_E_NOMEM;
                goto exit;
        }
        total_size=0;
        bzero(content_data,256);

        if((upg_timer = hftimer_create("UPG-TIMER",100,true,1,update_timer_callback,0))==NULL)
        {
                u_printf("create timer 1 fail\n");
                goto exit;
        }

        hftimer_start(upg_timer);
        MD5Init(&md5_ctx);
```

```
        while((read_size=hfhttp_read_content(hhttp,content_data,256))>0)
        {
                hfupdate_write_file(HFUPDATE_SW, total_size,content_data, read_size);
                MD5Update(&md5_ctx,(uint8_t*)content_data,read_size);
                total_size+=read_size;
                //u_printf("download file:[%d] [%d]\r",total_size,read_size);
                u_printf("%s",content_data);
        }
        MD5Final(digest,&md5_ctx);
        u_printf("read_size:%d digest is ",total_size);
        u_printf("%02x%02x%02x%02x",digest[0],digest[1],digest[2],digest[3]);
        u_printf("%02x%02x%02x%02x",digest[4],digest[5],digest[6],digest[7]);
        u_printf("%02x%02x%02x%02x",digest[8],digest[9],digest[10],digest[11]);
        u_printf("%02x%02x%02x%02x\n",digest[12],digest[13],digest[14],digest[15]);

        if(hfupdate_complete(HFUPDATE_SW,total_size)!=HF_SUCCESS)
        {
                u_printf("update software fail\n");
        }
exit:
        if(upg_timer!=NULL)
        {
                hftimer_delete(upg_timer);
                hftimer_delete(upg_timer);
        }
        if(temp_buf!=NULL)
                hfmem_free(temp_buf);
        if(content_data!=NULL)
                hfmem_free(content_data);
        if(hhttp!=0)
                hfhttp_close_session(&hhttp);
        hfgpio_fset_out_low(HFGPIO_F_NREADY);
        hfsys_enable_all_soft_watchdogs();
        return rv;
}

static int USER_FUNC test_httpc_post(char *purl)
{
        httpc_req_t  http_req;
        char content_data[34];
        char *temp_buf=NULL;
        parsed_url_t url={0};
        http_session_t hhttp=0;
        int total_size,read_size=0;
        int rv=0;
        tls_init_config_t  *tls_cfg=NULL;
        char *test_url=purl;

        bzero(&http_req,sizeof(http_req));
        http_req.type = HTTP_POST;
        http_req.version=HTTP_VER_1_1;

        if((temp_buf = (char*)hfmem_malloc(256))==NULL)
        {
                u_printf("no memory\n");
                return -HF_E_NOMEM;
        }

        bzero(temp_buf,sizeof(temp_buf));
        if((rv=hfhttp_parse_URL(test_url,temp_buf , 256, &url))!=HF_SUCCESS)
        {
                hfmem_free(temp_buf);
                return rv;
        }

        if((rv=hfhttp_open_session(&hhttp,test_url,0,tls_cfg,3))!=HF_SUCCESS)
        {
                u_printf("http open fail\n");
                hfmem_free(temp_buf);
                return rv;
        }

        http_req.resource = url.resource;
        http_req.content="POST TEST DATA\r\n";
        http_req.content_len = strlen(http_req.content);
        http_prepare_req(hhttp,&http_req,HDR_ADD_CONN_CLOSE);
        if((rv=hfhttp_send_request(hhttp,&http_req))!=HF_SUCCESS)
        {
                u_printf("http send request fail\n");
                hfmem_free(temp_buf);
                http_close_session(&hhttp);
                return rv;
        }
        total_size=0;
        bzero(content_data,sizeof(content_data));
        while((read_size=hfhttp_read_content(hhttp,content_data,32))>0)
        {
                total_size+=read_size;
```

```
                u_printf("%s",content_data);
        }

        u_printf("read_size:%d\n",total_size);
        hfmem_free(temp_buf);
        hfhttp_close_session(&hhttp);
        return total_size;
}

int USER_FUNC app_main (void)
{
        time_t now=time(NULL);

        HF_Debug(DEBUG_LEVEL,"sdk version(%s),the app_main start time is %d %s[AT DEMO]\n",\
                                                        hfsys_get_sdk_version(),now,ctime(&now));
        if(hfgpio_fmap_check()!=0)
        {
                while(1)
                {
                        HF_Debug(DEBUG_ERROR,"gpio map file error\n");
                        msleep(1000);
                }
                //return 0;
        }
        while(!hfnet_wifi_is_active())
        {
                msleep(50);
        }
        if(hfnet_start_assis(ASSIS_PORT)!=HF_SUCCESS)
        {
                HF_Debug(DEBUG_WARN,"start httpd fail\n");
        }
        if(hfnet_start_httpd(HFTHREAD_PRIORITIES_MID)!=HF_SUCCESS)
        {
                HF_Debug(DEBUG_WARN,"start httpd fail\n");
        }
        if(hfnet_start_uart(HFTHREAD_PRIORITIES_LOW,(hfnet_callback_t)uart_recv_callback)!=HF_SUCCESS)
        {
                HF_Debug(DEBUG_WARN,"start uart fail!\n");
        }
        if(hfnet_start_socketa(HFTHREAD_PRIORITIES_LOW,(hfnet_callback_t)socketa_recv_callback)!=HF_SUCCESS)
        {
                HF_Debug(DEBUG_WARN,"start socketa fail\n");
        }
        if(hfnet_start_socketb(HFTHREAD_PRIORITIES_LOW,(hfnet_callback_t)socketb_recv_callback)!=HF_SUCCESS)
        {
                HF_Debug(DEBUG_WARN,"start socketb fail\n");
        }

        {
                char *words[6]={NULL};
                char rsp[64]={0};
                hfat_send_cmd("AT+WANN\r\n",sizeof("AT+WANN\r\n"),rsp,64);
                if(hfat_get_words(rsp,words, 6)>0)
                {
                        u_printf("\nresult:%s\nmode:%s\nIP:%s\nMASK:%s\nGW:%s\n",\
                                        words[0],words[1],words[2],words[3],words[4]);
                }
        }
        adc_fid = HFGPIO_F_ADC_CHANNEL1;
        hfgpio_adc_enable(adc_fid);
        while(1)
        {
                u_printf("[%u]%u %u\n",adc_fid,hfgpio_adc_get_value(adc_fid),hfgpio_adc_get_voltage(adc_fid));
                //u_printf("[7]%u %u\n",SarAdcChannelGetValue(7),SarAdcGetGpioVoltage(7));
                msleep(5000);
        }

        return 1;

}

#endif
```

WebSide : http://gb.hi-flying.com

File Test

```
#include
#include
#include
#include
//#include
#include
#include
#include "../example.h"

#if (EXAMPLE_USE_DEMO==USER_FILE_DEMO)

const int hf_gpio_fid_to_pid_map_table[HFM_MAX_FUNC_CODE]=
{
        HF_M_PIN(2),        //HFGPIO_F_JTAG_TCK
        HF_M_PIN(3),        //HFGPIO_F_JTAG_TDO
        HF_M_PIN(4),        //HFGPIO_F_JTAG_TDI
        HF_M_PIN(5),        //HFGPIO_F_JTAG_TMS
        HFM_NOPIN,                  //HFGPIO_F_USBDP
        HFM_NOPIN,                  //HFGPIO_F_USBDM
        HF_M_PIN(39),       //HFGPIO_F_UART0_TX
        HF_M_PIN(40),       //HFGPIO_F_UART0_RTS
        HF_M_PIN(41),       //HFGPIO_F_UART0_RX
        HF_M_PIN(42),       //HFGPIO_F_UART0_CTS

        HF_M_PIN(27),       //HFGPIO_F_SPI_MISO
        HF_M_PIN(28),       //HFGPIO_F_SPI_CLK
        HF_M_PIN(29),       //HFGPIO_F_SPI_CS
        HF_M_PIN(30),       //HFGPIO_F_SPI_MOSI

        HFM_NOPIN,          //HFGPIO_F_UART1_TX,
        HFM_NOPIN,          //HFGPIO_F_UART1_RTS,
        HFM_NOPIN,          //HFGPIO_F_UART1_RX,
        HFM_NOPIN,          //HFGPIO_F_UART1_CTS,

        HF_M_PIN(43),       //HFGPIO_F_NLINK
        HF_M_PIN(44),       //HFGPIO_F_NREADY
        HF_M_PIN(45),       //HFGPIO_F_NRELOAD
        HF_M_PIN(7),        //HFGPIO_F_SLEEP_RQ
        HF_M_PIN(8),        //HFGPIO_F_SLEEP_ON

        HFM_NOPIN,                  //HFGPIO_F_WPS
        HF_M_PIN(15),               //HFGPIO_F_IR
        HFM_NOPIN,                  //HFGPIO_F_RESERVE2
        HFM_NOPIN,                  //HFGPIO_F_RESERVE3
        HFM_NOPIN,                  //HFGPIO_F_RESERVE4
        HFM_NOPIN,                  //HFGPIO_F_RESERVE5

        HFM_NOPIN,          //HFGPIO_F_USER_DEFINE
};
static int USER_FUNC hf_atcmd_filetest(pat_session_t s,int argc,char *argv[],char *rsp,int len);
const hfat_cmd_t user_define_at_cmds_table[]=
{
        {"FTEST",hf_atcmd_filetest,"   AT+FTEST=code,offset,value\r\n",NULL},
        {NULL,NULL,NULL,NULL} //the last item must be null
};

#define PRINTF(...)HF_Debug(DEBUG_LEVEL,__VA_ARGS__)

static int USER_FUNC hf_atcmd_filetest(pat_session_t s,int argc,char *argv[],char *rsp,int len)
{
        int code,offset,rlen,i;
        uint32_t temp;
        char cc;
        uint32_t file_size;

        if(argc!=3)
                return -1;

        code = atoi(argv[0]);
        offset = atoi(argv[1]);
        file_size = hffile_userbin_size();
        if(code==0)
        {
                sprintf(rsp,"=%d",file_size);
        }
        else if(code==1||code==5)
        {
                rlen = atoi(argv[2]);
                for(i=0;i < rlen; i++)
                {
                        hffile_userbin_read(offset+i,&cc,1);
```

```
                        if(code ==1)
                        {
                                u_printf("%c",cc);
                        }
                        else
                                u_printf("%02X",(uint8_t)cc);
                }
        }
        else if(code==2)
        {
                hffile_userbin_write(offset,argv[2],strlen(argv[2]));
        }
        else if(code==3)
        {
                u_printf("userbin file size=%d\n",file_size);
                for(i=0;i < file_size;)
                {
                        hffile_userbin_write(i,(char*)&i,sizeof(i));
                        i+=sizeof(i);
                }
                for(i=0;i < file_size;)
                {
                        hffile_userbin_read(i,(char*)&temp,sizeof(temp));
                        //u_printf("file[%d]=%08x\n",i,temp);
                        if(temp!=i)
                        {
                                u_printf("test fail\n");
                        }
                        i+=sizeof(temp);
                }
        }
        else if(code==4)
        {
                hffile_userbin_zero();
        }

        return 0;
}

#define CFG_HDR_FILE_OFFSET                     0
#define CFG_BRMID_FILE_OFFSET                   32
#define CFG_BRMADDR_FILE_OFFSET                 64
#define CFG_BRMPORT_FILE_OFFSET                 96

static char *strnstr(const char *s, const char *find, size_t slen)
{
        int i,flen;

        flen = strlen(find);
        if(flen>slen)
                return NULL;

        for(i=0;i<=slen-flen;i++)
        {
                if(s[i]==*find)
                {
                        if(strncmp(s+i,find,slen-i)==0)
                                return (char*)(s+i);
                }
        }

        return NULL;
}


int hfhttpd_user_nvset( char * cfg_name,int name_len,char* value,int val_len)
{
        char temp[20];

        bzero(temp,sizeof(temp));
        if(val_len>=20)
        {
                return 0;
        }
        memcpy(temp,value,val_len);

        if(strnstr(cfg_name,"CFG_BRMID",name_len)!=NULL)
        {
                hffile_userbin_write(CFG_BRMID_FILE_OFFSET,temp,sizeof(temp));
                return 0;
        }
        else if(strnstr(cfg_name,"CFG_BRMADDR",name_len)!=NULL)
        {
                hffile_userbin_write(CFG_BRMADDR_FILE_OFFSET,temp,sizeof(temp));
                return 0;
        }
        else if(strnstr(cfg_name,"CFG_BRMPORT",name_len)!=NULL)
        {
```

```
                hffile_userbin_write(CFG_BRMPORT_FILE_OFFSET,temp,sizeof(temp));
                return 0;
        }

        return -1;
}

int hfhttpd_user_nvget( char *cfg_name,int name_len,char *value,int val_len)
{
        char temp[20];

        bzero(temp,sizeof(temp));
        if(strnstr(cfg_name,"CFG_BRMID",name_len)!=NULL)
        {
                hffile_userbin_read(CFG_BRMID_FILE_OFFSET,temp,19);
                strcpy(value,temp);
                return 0;
        }
        else if(strnstr(cfg_name,"CFG_BRMADDR",name_len)!=NULL)
        {
                hffile_userbin_read(CFG_BRMADDR_FILE_OFFSET,temp,19);
                strcpy(value,temp);
                return 0;
        }
        else if(strnstr(cfg_name,"CFG_BRMPORT",name_len)!=NULL)
        {
                hffile_userbin_read(CFG_BRMPORT_FILE_OFFSET,temp,19);
                strcpy(value,temp);
                return 0;
        }

        return -1;
}

USER_FUNC void test_file_thread_start(void)
{
        uint8_t hdr[8]={0};


        if(hffile_userbin_read(CFG_HDR_FILE_OFFSET,(char*)hdr,8)>0)
        {
                if(hdr[0]!=0x00||hdr[1]!=0x01)
                {
                        bzero(hdr,8);
                        hdr[0]=0x00;
                        hdr[1]=0x01;
                        hffile_userbin_zero();
                        hffile_userbin_write(CFG_HDR_FILE_OFFSET,(char*)hdr,8);
                }
        }

        hfnet_httpd_set_get_nvram_callback(hfhttpd_user_nvset,hfhttpd_user_nvget);
}


int USER_FUNC app_main (void)
{
        time_t now=time(NULL);

        HF_Debug(DEBUG_LEVEL,"[FILE DEMO]sdk version(%s),the app_main start time is %d %s\n",\
                                                hfsys_get_sdk_version(),now,ctime(&now));
        if(hfgpio_fmap_check()!=0)
        {
                while(1)
                {
                        HF_Debug(DEBUG_ERROR,"gpio map file error\n");
                        msleep(1000);
                }
                //return 0;
        }

        while(!hfnet_wifi_is_active())
        {
                msleep(50);
        }

        if(hfnet_start_assis(ASSIS_PORT)!=HF_SUCCESS)
        {
                HF_Debug(DEBUG_WARN,"start httpd fail\n");
        }

        //if(hfnet_start_httpd(HFTHREAD_PRIORITIES_MID)!=HF_SUCCESS)
        //{
        //        HF_Debug(DEBUG_WARN,"start httpd fail\n");
        //}
        if(hfnet_start_uart(HFTHREAD_PRIORITIES_LOW,NULL)!=HF_SUCCESS)
        {
                HF_Debug(DEBUG_WARN,"start uart fail!\n");
```

```
        }
        if(hfnet_start_socketa(HFTHREAD_PRIORITIES_LOW,(hfnet_callback_t)NULL)!=HF_SUCCESS)
        {
                HF_Debug(DEBUG_WARN,"start socketa fail\n");
        }
        //if(hfnet_start_socketb(HFTHREAD_PRIORITIES_LOW,(hfnet_callback_t)NULL)!=HF_SUCCESS)
        //{
        //        HF_Debug(DEBUG_WARN,"start socketb fail\n");
        //}

        test_file_thread_start();

        return 1;

}

#endif
```

WebSide : http://gb.hi-flying.com

Gpio Test

```
#include
#include
#include
//#include
#include
#include "../example.h"

#if (EXAMPLE_USE_DEMO==USER_GPIO_DEMO)

#define HFGPIO_F_TCP_LINK                       (HFGPIO_F_USER_DEFINE+0)
#define HFGPIO_F_USER_RELOAD            (HFGPIO_F_USER_DEFINE+1)


const int hf_gpio_fid_to_pid_map_table[HFM_MAX_FUNC_CODE]=
{
        HF_M_PIN(2),        //HFGPIO_F_JTAG_TCK
        HF_M_PIN(3),        //HFGPIO_F_JTAG_TDO
        HF_M_PIN(4),        //HFGPIO_F_JTAG_TDI
        HF_M_PIN(5),        //HFGPIO_F_JTAG_TMS
        HFM_NOPIN,                  //HFGPIO_F_USBDP
        HFM_NOPIN,                  //HFGPIO_F_USBDM
        HF_M_PIN(39),       //HFGPIO_F_UART0_TX
        HF_M_PIN(40),       //HFGPIO_F_UART0_RTS
        HF_M_PIN(41),       //HFGPIO_F_UART0_RX
        HF_M_PIN(42),       //HFGPIO_F_UART0_CTS

        HF_M_PIN(27),       //HFGPIO_F_SPI_MISO
        HF_M_PIN(28),       //HFGPIO_F_SPI_CLK
        HF_M_PIN(29),       //HFGPIO_F_SPI_CS
        HF_M_PIN(30),       //HFGPIO_F_SPI_MOSI

        HFM_NOPIN,          //HFGPIO_F_UART1_TX,
        HFM_NOPIN,          //HFGPIO_F_UART1_RTS,
        HFM_NOPIN,          //HFGPIO_F_UART1_RX,
        HFM_NOPIN,          //HFGPIO_F_UART1_CTS,

        HFM_NOPIN,//HF_M_PIN(43),     //HFGPIO_F_NLINK
        HF_M_PIN(44),       //HFGPIO_F_NREADY
        HFM_NOPIN,//HF_M_PIN(45),     //HFGPIO_F_NRELOAD
        HF_M_PIN(7),        //HFGPIO_F_SLEEP_RQ
        HF_M_PIN(8),        //HFGPIO_F_SLEEP_ON

        HFM_NOPIN,                  //HFGPIO_F_RESERVE0
        HFM_NOPIN,                  //HFGPIO_F_RESERVE1
        HFM_NOPIN,                  //HFGPIO_F_RESERVE2
        HFM_NOPIN,                  //HFGPIO_F_RESERVE3
        HFM_NOPIN,                  //HFGPIO_F_RESERVE4
```

```
        HFM_NOPIN,                      //HFGPIO_F_RESERVE5

        HF_M_PIN(43),       //HFGPIO_F_USER_DEFINE,HFGPIO_F_TCP_LINK
        HF_M_PIN(45)    //HFGPIO_F_USER_RELOAD
};

static void USER_FUNC do_user_reload(uint32_t arg1,uint32_t arg2);
static char at_cmd_rsp[128]={0};
static int press_reload_key=0;

static void USER_FUNC do_user_reload(uint32_t arg1,uint32_t arg2)
{
        time_t now=time(NULL);

        if(hfgpio_fpin_is_high(HFGPIO_F_NREADY))
        {
                hfgpio_fset_out_low(HFGPIO_F_NREADY);
        }
        else
        {
                hfgpio_fset_out_high(HFGPIO_F_NREADY);
        }
        //pull up the RELOD pin of LPB evaluation board, if high level, release the key.
        if(hfgpio_fpin_is_high(HFGPIO_F_USER_RELOAD))
        {
                press_reload_key=1;
                u_printf("release the reload button!%d %d\n",now,hfgpio_fpin_is_high(HFGPIO_F_SLEEP_RQ));
        }
        else
        {
                press_reload_key=0;
                u_printf("press the reload button!%d %d\n",now,hfgpio_fpin_is_high(HFGPIO_F_SLEEP_RQ));
        }
}


static void USER_FUNC do_user_rq(uint32_t arg1,uint32_t arg2)
{
        time_t now=time(NULL);

        u_printf("press the RQ button!%d\n",now);
        if(hfgpio_fpin_is_high(HFGPIO_F_NREADY))
        {
                hfgpio_fset_out_low(HFGPIO_F_NREADY);
        }
        else
        {
                hfgpio_fset_out_high(HFGPIO_F_NREADY);
        }
}

static void USER_FUNC test_gpio_start()
{
        hfgpio_fset_out_high(HFGPIO_F_TCP_LINK);
        if(hfgpio_configure_fpin_interrupt(HFGPIO_F_USER_RELOAD,HFPIO_IT_EDGE,do_user_reload,1)!=HF_SUCCESS)
        {
                u_printf("configure HFGPIO_F_USER_RELOAD fail\n");
                return;
        }

        if(hfgpio_configure_fpin_interrupt(HFGPIO_F_SLEEP_RQ,HFPIO_IT_FALL_EDGE,do_user_rq,1)!=HF_SUCCESS)
        {
                u_printf("configure HFGPIO_F_SLEEP_RQ fail\n");
                return;
        }

        while(1)
        {
                if(press_reload_key)
                {
                        hfsys_reload();
                        hfat_send_cmd("AT+SMTLK\r\n",sizeof("AT+SMTLK\r\n"),at_cmd_rsp,64);
                }
                else
                        msleep(100);
        }
}

const hfat_cmd_t user_define_at_cmds_table[]=
{
        {NULL,NULL,NULL,NULL} //the last item must be null
};

static int USER_FUNC socketa_recv_callback(uint32_t event,char *data,uint32_t len,uint32_t buf_len)
{
        if(event==HFNET_SOCKETA_CONNECTED)
        {
                hfgpio_fset_out_low(HFGPIO_F_TCP_LINK);
```

```
	}
	else if(event==HFNET_SOCKETA_DISCONNECTED)
	{
		hfgpio_fset_out_high(HFGPIO_F_TCP_LINK);
	}
	else if(event==HFNET_SOCKETA_DATA_READY)
	{
		HF_Debug(DEBUG_LEVEL_LOW,"[%d]socketa recv %d bytes data %d %c\n",event,len,buf_len,data[0]);
	}

	return len;
}

static int USER_FUNC socketb_recv_callback(uint32_t event,char *data,uint32_t len,uint32_t buf_len)
{
	if(event==HFNET_SOCKETB_CONNECTED)
	{
		u_printf("socket b connected!\n");
	}
	else if(event==HFNET_SOCKETB_DISCONNECTED)
	{
		u_printf("socket b disconnected!\n");
	}
	else if(event==HFNET_SOCKETB_DATA_READY)
		HF_Debug(DEBUG_LEVEL_LOW,"[%d]socketb recv %d bytes data %d %c\n",event,len,buf_len,data[0]);

	return len;
}

static int USER_FUNC uart_recv_callback(uint32_t event,char *data,uint32_t len,uint32_t buf_len)
{
	HF_Debug(DEBUG_LEVEL_LOW,"[%d]uart recv %d bytes data %d\n",event,len,buf_len);
	return len;
}

int USER_FUNC app_main (void)
{
	time_t now=time(NULL);

	u_printf("[GPIO DEMO]sdk version(%s),the app_main start time is %d %s\n",\
		hfsys_get_sdk_version(),now,ctime(&now));
	if(hfgpio_fmap_check()!=0)
	{
		while(1)
		{
			HF_Debug(DEBUG_ERROR,"gpio map file error\n");
			msleep(1000);
		}
		return 0;
	}
	while(!hfnet_wifi_is_active())
	{
		msleep(50);
	}
	if(hfnet_start_assis(ASSIS_PORT)!=HF_SUCCESS)
	{
		HF_Debug(DEBUG_WARN,"start httpd fail\n");
	}
	if(hfnet_start_httpd(HFTHREAD_PRIORITIES_MID)!=HF_SUCCESS)
	{
		HF_Debug(DEBUG_WARN,"start httpd fail\n");
	}
	if(hfnet_start_uart(HFTHREAD_PRIORITIES_LOW,(hfnet_callback_t)uart_recv_callback)!=HF_SUCCESS)
	{
		HF_Debug(DEBUG_WARN,"start uart fail!\n");
	}
	if(hfnet_start_socketa(HFTHREAD_PRIORITIES_LOW,(hfnet_callback_t)socketa_recv_callback)!=HF_SUCCESS)
	{
		HF_Debug(DEBUG_WARN,"start socketa fail\n");
	}
	if(hfnet_start_socketb(HFTHREAD_PRIORITIES_LOW,(hfnet_callback_t)socketb_recv_callback)!=HF_SUCCESS)
	{
		HF_Debug(DEBUG_WARN,"start socketb fail\n");
	}

	test_gpio_start();

	return 1;
}

#endif
```

WebSide : http://gb.hi-flying.com

hfsys_get_reset_reason

```
static void show_reset_reason(void)
{
    uint32_t reset_reason=0;


    reset_reason = hfsys_get_reset_reason();

    if(reset_reason&HFSYS_RESET_REASON_ERESET)
    {
        u_printf("HFSYS_RESET_REASON_ERESET\n");
    }
    if(reset_reason&HFSYS_RESET_REASON_IRESET0)
    {
        u_printf("HFSYS_RESET_REASON_IRESET0\n");
    }
    if(reset_reason&HFSYS_RESET_REASON_IRESET1)
    {
        u_printf("HFSYS_RESET_REASON_IRESET1\n");
    }
    if(reset_reason==HFSYS_RESET_REASON_NORMAL)
    {
        u_printf("HFSYS_RESET_REASON_NORMAL\n");
    }
    if(reset_reason&HFSYS_RESET_REASON_WPS)
    {
        u_printf("HFSYS_RESET_REASON_WPS\n");
    }
    if(reset_reason&HFSYS_RESET_REASON_SMARTLINK_START)
    {
        u_printf("HFSYS_RESET_REASON_SMARTLINK_START\n");
    }
    if(reset_reason&HFSYS_RESET_REASON_SMARTLINK_OK)
    {
        u_printf("HFSYS_RESET_REASON_SMARTLINK_OK\n");
    }

    return;
}

int USER_FUNC app_main (void)
{
   show_reset_reason();
          ·····················
}
```


hfsys_register_system_event

```
static int hfsys_event_callback( uint32_t event_id,void * param)
{
    switch(event_id)
    {
        case HFE_WIFI_STA_CONNECTED:
            u_printf("wifi sta connected!!\n");
```

```
            break;
        case HFE_WIFI_STA_DISCONNECTED:
            u_printf("wifi sta disconnected!!\n");
            break;
        case HFE_DHCP_OK:
        {
            uint32_t *p_ip;
            p_ip = (uint32_t*)param;
            u_printf("dhcp ok %08X!\n",*p_ip);
        }
            break;
        case HFE_SMTLK_OK:
            u_printf("smtlk ok!\n");
            return -1;
            break;
        case HFE_CONFIG_RELOAD:
            u_printf("reload!\n");
            break;
        default:
            break;
    }
    return 0;
}

int USER_FUNC app_main (void)
{
    if( hfsys_register_system_event( (hfsys_event_callback_t)hfsys_event_callback)\
      != HF_SUCCESS)
    {
        u_printf("register system event fail\n");
    }
        ..................
}
```

WebSide : http://gb.hi-flying.com

Hfthread_create

```
#include

// thread entrance function
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;
}
```

WebSide : http://gb.hi-flying.com

Ir Test

```
#include
#include
#include
#include
//#include
#include
#include
#include "../example.h"


#if (EXAMPLE_USE_DEMO==USER_IR_DEMO)

#define HFGPIO_F_IRTRNSMITTER                           HFGPIO_F_USER_DEFINE
#define HFGPIO_F_IR_KEY0                                (HFGPIO_F_USER_DEFINE+1)
#define HFGPIO_F_IR_LED                                         (HFGPIO_F_USER_DEFINE+2)

const int hf_gpio_fid_to_pid_map_table[HFM_MAX_FUNC_CODE]=
{
        HF_M_PIN(2),        //HFGPIO_F_JTAG_TCK
        HF_M_PIN(3),        //HFGPIO_F_JTAG_TDO
        HF_M_PIN(4),        //HFGPIO_F_JTAG_TDI
        HF_M_PIN(5),        //HFGPIO_F_JTAG_TMS
        HFM_NOPIN,                  //HFGPIO_F_USBDP
        HFM_NOPIN,                  //HFGPIO_F_USBDM
        HF_M_PIN(39),       //HFGPIO_F_UART0_TX
        HF_M_PIN(40),       //HFGPIO_F_UART0_RTS
        HF_M_PIN(41),       //HFGPIO_F_UART0_RX
        HF_M_PIN(42),       //HFGPIO_F_UART0_CTS

        HF_M_PIN(27),       //HFGPIO_F_SPI_MISO
        HF_M_PIN(28),       //HFGPIO_F_SPI_CLK
        HF_M_PIN(29),       //HFGPIO_F_SPI_CS
        HF_M_PIN(30),       //HFGPIO_F_SPI_MOSI

        HFM_NOPIN,          //HFGPIO_F_UART1_TX,
        HFM_NOPIN,          //HFGPIO_F_UART1_RTS,
        HFM_NOPIN,          //HFGPIO_F_UART1_RX,
        HFM_NOPIN,          //HFGPIO_F_UART1_CTS,

        HF_M_PIN(43),       //HFGPIO_F_NLINK
        HF_M_PIN(44),       //HFGPIO_F_NREADY
        HF_M_PIN(45),       //HFGPIO_F_NRELOAD
        HFM_NOPIN,//HF_M_PIN(7),      //HFGPIO_F_SLEEP_RQ
        HF_M_PIN(8),        //HFGPIO_F_SLEEP_ON

        HFM_NOPIN,                  //HFGPIO_F_WPS
        HF_M_PIN(15),               //HFGPIO_F_IR
        HFM_NOPIN,                  //HFGPIO_F_RESERVE2
        HFM_NOPIN,                  //HFGPIO_F_RESERVE3
        HFM_NOPIN,                  //HFGPIO_F_RESERVE4
        HFM_NOPIN,                  //HFGPIO_F_RESERVE5
        HF_M_PIN(11),       //HFGPIO_F_USER_DEFINE
        HF_M_PIN(7),      // HFGPIO_F_IR_KEY0
        HF_M_PIN(13),
};

static int ir_recv_counter=0;
static int ir_recv_timer_stop=1;
static hftimer_handle_t test_timer_hardware=NULL;
static int record_save_to_file=0;

#define RECORD_FILE_HDR_SIZE            (16)
static char RECORD_FILE_HDR[RECORD_FILE_HDR_SIZE]="ir-records-file";
#define ONE_KEY_RECORD_SIZE             (512)
#define MAX_IR_KEY                                      (6)

typedef struct _IR_RECV_RECORD
{
        //uint32_t recv_time;
        //uint8_t   io_state;
        uint32_t  data;
}IR_RECV_RECORD,*PIR_RECV_RECORD;

#define RECORD_SET_IO_STATE_HI(__p_record)      (__p_record).data |= 0x80000000
#define RECORD_SET_IO_STATE_LO(__p_record)      (__p_record).data &= 0x7FFFFFFF
#define RECORD_GET_IO_STATE(__p_record)         (((__p_record).data>>31)&0x01)
```

```
#define RECORD_GET_RECV_TIME(__p_record)        ((__p_record).data&0x7FFFFFFF)
#define RECORD_SET_RECV_TIME(__p_record,_recv_time)  \
          ((__p_record).data= ((__p_record).data&0x80000000)|(_recv_time&0x7FFFFFFF))


static void USER_FUNC ir_transfer_key(PIR_RECV_RECORD p_key_records,int cnt);
static int hf_atcmd_irsnd(pat_session_t s,int argc,char *argv[],char *rsp,int len);
static int ir_press_key_id =-1;

const hfat_cmd_t user_define_at_cmds_table[]=
{
         {"IRSND",hf_atcmd_irsnd,"   AT+IRSND=code\r\n",NULL},
         {NULL,NULL,NULL,NULL} //the last item must be null
};

struct _IR_RECV_RECORD   ir_recv_record[200]={0};

void USER_FUNC save_key_record_to_file(PIR_RECV_RECORD p_record,int cnt)
{
         int key_cnt=0;
         int offset;

         if(!record_save_to_file)
                  return;

         if(cnt<16)
                  return;

         hffile_userbin_read(RECORD_FILE_HDR_SIZE,(char*)&key_cnt,4);
         if(key_cnt>=MAX_IR_KEY)
         {
                  u_printf("record file is full!\n");
                  return;
         }

         if(cnt*sizeof(IR_RECV_RECORD)>ONE_KEY_RECORD_SIZE)
                  return;

         offset=RECORD_FILE_HDR_SIZE+4+key_cnt*ONE_KEY_RECORD_SIZE;
         hffile_userbin_write(offset,(char*)&cnt,4);
         hffile_userbin_write(offset+4,(char*)p_record,cnt*sizeof(IR_RECV_RECORD));
         key_cnt++;
         hffile_userbin_write(RECORD_FILE_HDR_SIZE,(char*)&key_cnt,4);
         ir_press_key_id = key_cnt-1;

         return;
}

int USER_FUNC read_key_record_from_file(int key_id,PIR_RECV_RECORD p_record,int size)
{
         int key_cnt=0;
         int records_cnt=0;
         int offset=RECORD_FILE_HDR_SIZE+4+key_id*ONE_KEY_RECORD_SIZE;

         hffile_userbin_read(RECORD_FILE_HDR_SIZE,(char*)&key_cnt,4);
         if(key_id>=key_cnt)
                  return -1;

         hffile_userbin_read(offset,(char*)&records_cnt,4);
         hffile_userbin_read(offset+4, (char*)p_record, size*sizeof(IR_RECV_RECORD));

         return records_cnt;
}

static void dump_ir_key_record(PIR_RECV_RECORD p_key_records,int cnt)
{
         int i;

         for(i=0;i < cnt;i++)
         {
                  u_printf("[%d]-->%d %u\n",i,RECORD_GET_IO_STATE(p_key_records[i]),\
                                                     RECORD_GET_RECV_TIME(p_key_records[i]));
         }

         return;
}

static IR_RECV_RECORD ir_key_records[200]={0};


static int hf_atcmd_irsnd(pat_session_t s,int argc,char *argv[],char *rsp,int len)
{
         int key_id=0;
         int code=0;
         int key_records_cnt=0;

         if(argc<2)
                  return -1;
```

```
	code = atoi(argv[0]);
	key_id = atoi(argv[1]);

	if(code==3)
	{
		int i;
		key_records_cnt=100;
		for(i=0;i<100;i++)
		{
			RECORD_SET_RECV_TIME(ir_key_records[i],i*1000);
			if(i%2)
				RECORD_SET_IO_STATE_HI(ir_key_records[i]);
			else
				RECORD_SET_IO_STATE_LO(ir_key_records[i]);
		}
		ir_transfer_key(ir_key_records,key_records_cnt);
		return 0;
	}
	else if(code==2)
	{
		int key_cnt=0;
		record_save_to_file=1;
		hffile_userbin_write(RECORD_FILE_HDR_SIZE,(char*)&key_cnt,4);
		return 0;
	}

	if((key_records_cnt=read_key_record_from_file(key_id,ir_key_records,200))<=0)
	{
		return -1;
	}

	if(code==0)
	{
		dump_ir_key_record(ir_key_records,key_records_cnt);
	}
	else if(code==1)
	{
		ir_transfer_key(ir_key_records,key_records_cnt);
	}

	return 0;

}

void USER_FUNC test_timer_callback( hftimer_handle_t htimer )
{
	ir_recv_timer_stop = 1;
	hftimer_stop(test_timer_hardware);

	if(ir_recv_counter>0)
	{
		dump_ir_key_record(ir_recv_record,ir_recv_counter);
		save_key_record_to_file(ir_recv_record,ir_recv_counter);
	}
	ir_recv_counter=0;
}

static void USER_FUNC do_user_ir(uint32_t arg1,uint32_t arg2)
{
	time_t now=time(NULL);
	uint32_t recv_time;

	if(ir_recv_timer_stop)
	{
		hftimer_start(test_timer_hardware);
		ir_recv_timer_stop=0;
	}

	recv_time = hftimer_get_counter(test_timer_hardware);
	if(hfgpio_fpin_is_high(HFGPIO_F_IR))
	{
		//ir_recv_record[ir_recv_counter].recv_time=hftimer_get_counter(test_timer_hardware);
		//ir_recv_record[ir_recv_counter].io_state=1;
		RECORD_SET_IO_STATE_HI(ir_recv_record[ir_recv_counter]);
		if(hfir_is_key_come())//ENC
		{
			u_printf("key code=%08X\n",hfir_get_key_code());
		}
		//hfgpio_fset_out_high(HFGPIO_F_IR_LED);
	}
	else
	{
		//ir_recv_record[ir_recv_counter].recv_time=hftimer_get_counter(test_timer_hardware);
		//ir_recv_record[ir_recv_counter].io_state=0;
		RECORD_SET_IO_STATE_LO(ir_recv_record[ir_recv_counter]);
		//hfgpio_fset_out_low(HFGPIO_F_IR_LED);
	}
```

```
        RECORD_SET_RECV_TIME(ir_recv_record[ir_recv_counter],recv_time);

        ir_recv_counter ++;

}

static void ir_transfer_key_by_id(int key_id)
{
        int key_cnt=0;
        int key_records_cnt=0;

        hffile_userbin_read(RECORD_FILE_HDR_SIZE,(char*)&key_cnt,4);
        if(key_id>=key_cnt)
        {
                return;
        }

        if((key_records_cnt=read_key_record_from_file(key_id,ir_key_records,200))<=0)
        {
                return ;
        }
        ir_transfer_key(ir_key_records,key_records_cnt);

        return;
}


static void USER_FUNC do_press_irkey0(uint32_t arg1,uint32_t arg2)
{
        int key_cnt=0;
        static int key_id=0;

        if(ir_press_key_id!=-1)
                return;

        hffile_userbin_read(RECORD_FILE_HDR_SIZE,(char*)&key_cnt,4);
        if(key_id>=key_cnt)
        {
                key_id = 0;
        }

        ir_press_key_id=key_id;

        key_id++;

        return;

}


static void ir_start_38K(void)
{
        hfgpio_pwm_enable( HFGPIO_F_IRTRNSMITTER,40000,50);
}

static void ir_stop_38K(void)
{
        hfgpio_pwm_disable(HFGPIO_F_IRTRNSMITTER);
        hfgpio_fset_out_high(HFGPIO_F_IRTRNSMITTER);
}

static void USER_FUNC ir_transfer_key(PIR_RECV_RECORD p_key_records,int cnt)
{
        int i;
        uint32_t recv_time;
        uint32_t io_state;

        hfgpio_fdisable_interrupt(HFGPIO_F_IR);
        hfthread_suspend_all();
        ir_stop_38K();
        hftimer_stop(test_timer_hardware);
        hftimer_start(test_timer_hardware);

        for(i=0;i < cnt;i++)
        {
                recv_time = RECORD_GET_RECV_TIME(p_key_records[i]);
                io_state = RECORD_GET_IO_STATE(p_key_records[i]);

                if(io_state==0)
                {
                        while(hftimer_get_counter(test_timer_hardware) < recv_time);
                        ir_start_38K();
                }
                else
                {
                        while(hftimer_get_counter(test_timer_hardware) < recv_time);
                        ir_stop_38K();
                }
        }
```

```
		ir_stop_38K();
		hfthread_resume_all();
		hfgpio_fenable_interrupt(HFGPIO_F_IR);

		return;
}


void init_ir_key_records_file(void)
{
		char hdr[RECORD_FILE_HDR_SIZE]={0};

		hffile_userbin_read(0,hdr,RECORD_FILE_HDR_SIZE);
		if(memcmp(hdr,RECORD_FILE_HDR,sizeof(RECORD_FILE_HDR))!=0)
		{
			hffile_userbin_zero();
			hffile_userbin_write(0, RECORD_FILE_HDR,RECORD_FILE_HDR_SIZE);
		}

		return;
}

int USER_FUNC app_main (void)
{
		time_t now=time(NULL);

		HF_Debug(DEBUG_LEVEL,"[IR DEMO]sdk version(%s),\
				the app_main start time is %d %s\n",hfsys_get_sdk_version(),now,ctime(&now));
		if(hfgpio_fmap_check()!=0)
		{
			while(1)
			{
				HF_Debug(DEBUG_ERROR,"gpio map file error\n");
				msleep(1000);
			}
			//return 0;
		}

		while(!hfnet_wifi_is_active())
		{
			msleep(50);
		}

		if(hfnet_start_assis(ASSIS_PORT)!=HF_SUCCESS)
		{
			HF_Debug(DEBUG_WARN,"start httpd fail\n");
		}

		if(hfnet_start_httpd(HFTHREAD_PRIORITIES_MID)!=HF_SUCCESS)
		{
			HF_Debug(DEBUG_WARN,"start httpd fail\n");
		}
		if(hfnet_start_uart(HFTHREAD_PRIORITIES_LOW,NULL)!=HF_SUCCESS)
		{
			HF_Debug(DEBUG_WARN,"start uart fail!\n");
		}
		if(hfnet_start_socketa(HFTHREAD_PRIORITIES_LOW,(hfnet_callback_t)NULL)!=HF_SUCCESS)
		{
			HF_Debug(DEBUG_WARN,"start socketa fail\n");
		}
		if(hfnet_start_socketb(HFTHREAD_PRIORITIES_LOW,(hfnet_callback_t)NULL)!=HF_SUCCESS)
		{
			HF_Debug(DEBUG_WARN,"start socketb fail\n");
		}
		hfir_ignore_lead_header(true);

		init_ir_key_records_file();

		if(hfgpio_configure_fpin_interrupt(HFGPIO_F_IR_KEY0,HFPIO_IT_FALL_EDGE,do_press_irkey0,1)!=HF_SUCCESS)
		{
			u_printf("configure HFGPIO_F_IR_KEY0 fail\n");
			return 0;
		}
#if 1
		if(hfgpio_configure_fpin_interrupt(HFGPIO_F_IR,HFPIO_IT_EDGE,do_user_ir,1)!=HF_SUCCESS)
		{
			u_printf("configure HFGPIO_F_IR fail\n");
			return 0;
		}

#else
		while(1)
		{
			if(hfir_is_key_come())
			{
				u_printf("key code=%08X\n",hfir_get_key_code());
			}
			else
```

```
                msleep(100);
        }
#endif

        if((test_timer_hardware=hftimer_create("HDW-TIMER",110000,true,3,\
                        test_timer_callback,HFTIMER_FLAG_HARDWARE_TIMER))==NULL)
        {
                u_printf("create  hardware timer fail\n");
        }

        while(1)
        {
                if(ir_press_key_id>=0)
                {
                        u_printf("press ir_key_id=%d\n",ir_press_key_id);
                        ir_transfer_key_by_id(ir_press_key_id);
                }
                ir_press_key_id=-1;
                msleep(100);
        }
        return 1;

}

#endif
```

WebSide : http://gb.hi-flying.com

Netcallback Test

```
#include
#include
#include
//#include
#include
#include "../example.h"

#if (EXAMPLE_USE_DEMO==USER_CALLBACK_DEMO)

const int hf_gpio_fid_to_pid_map_table[HFM_MAX_FUNC_CODE]=
{
        HF_M_PIN(2),        //HFGPIO_F_JTAG_TCK
        HF_M_PIN(3),        //HFGPIO_F_JTAG_TDO
        HF_M_PIN(4),        //HFGPIO_F_JTAG_TDI
        HF_M_PIN(5),        //HFGPIO_F_JTAG_TMS
        HFM_NOPIN,                  //HFGPIO_F_USBDP
        HFM_NOPIN,                  //HFGPIO_F_USBDM
        HF_M_PIN(39),       //HFGPIO_F_UART0_TX
        HF_M_PIN(40),       //HFGPIO_F_UART0_RTS
        HF_M_PIN(41),       //HFGPIO_F_UART0_RX
        HF_M_PIN(42),       //HFGPIO_F_UART0_CTS

        HF_M_PIN(27),       //HFGPIO_F_SPI_MISO
        HF_M_PIN(28),       //HFGPIO_F_SPI_CLK
        HF_M_PIN(29),       //HFGPIO_F_SPI_CS
        HF_M_PIN(30),       //HFGPIO_F_SPI_MOSI

        HFM_NOPIN,          //HFGPIO_F_UART1_TX,
        HFM_NOPIN,          //HFGPIO_F_UART1_RTS,
        HFM_NOPIN,          //HFGPIO_F_UART1_RX,
        HFM_NOPIN,          //HFGPIO_F_UART1_CTS,

        HF_M_PIN(43),       //HFGPIO_F_NLINK
        HF_M_PIN(44),       //HFGPIO_F_NREADY
        HF_M_PIN(45),       //HFGPIO_F_NRELOAD
        HF_M_PIN(7),        //HFGPIO_F_SLEEP_RQ
        HF_M_PIN(8),        //HFGPIO_F_SLEEP_ON

        HFM_NOPIN,                  //HFGPIO_F_RESERVE0
        HFM_NOPIN,                  //HFGPIO_F_RESERVE1
```

```
        HFM_NOPIN,                  //HFGPIO_F_RESERVE2
        HFM_NOPIN,                  //HFGPIO_F_RESERVE3
        HFM_NOPIN,                  //HFGPIO_F_RESERVE4
        HFM_NOPIN,                  //HFGPIO_F_RESERVE5

        HFM_NOPIN,          //HFGPIO_F_USER_DEFINE
};


const hfat_cmd_t user_define_at_cmds_table[]=
{
        {NULL,NULL,NULL,NULL} //the last item must be null
};

static hftimer_handle_t hnlink_timer=NULL;
#define NLINK_FALSH_TIMER_ID          (1)

void USER_FUNC nlink_falsh_timer_callback( hftimer_handle_t htimer )
{
        if(hftimer_get_timer_id(htimer)==NLINK_FALSH_TIMER_ID)
        {
                if(hfgpio_fpin_is_high(HFGPIO_F_NLINK))
                        hfgpio_fset_out_low(HFGPIO_F_NLINK);
                else
                        hfgpio_fset_out_high(HFGPIO_F_NLINK);
                //hftimer_start(htimer);// when create, the setting of auto_reload is false, restart the timer manually
        }
}

static int USER_FUNC assis_ex_recv_callback(uint32_t event,char *data,uint32_t len,uint32_t buf_len)
{
        if(event == HFNET_ASSIS_DATA_READY)
        {
                char tmp[64]={0};
                char rsp[64]={0};
                char *ip[6]={0};
                char *mac[3]={0};
                char response[40]={0};

                uint32_t ip_addr;
                uint16_t port;

                MEMCPY(&ip_addr, data+len, sizeof(struct ip_addr));
                Memcpy(&port, data+len+sizeof(struct ip_addr), sizeof(port));
                u_printf("ip:%s, port:%d\n", inet_ntoa(ip_addr),port);

                MEMCPY(tmp, data, len);
                if(strcasecmp("HF-A11ASSISTHREAD", tmp)==0)
                {
                        hfat_send_cmd("AT+WANN\r\n", sizeof("AT+WANN\r\n"), rsp, 64);
                        if(hfat_get_words(rsp, ip, 6)>0)
                        {
                                u_printf("local ip:%s\n", ip[2]);
                                sprintf(response,"%s,",ip[2]);
                        }

                        memset(rsp, 0, sizeof(rsp));
                        hfat_send_cmd("AT+WSMAC\r\n", sizeof("AT+WSMC\r\n"), rsp, 64);
                        u_printf("AT+WSMAC's response:%s\n",rsp);
                        if(hfat_get_words(rsp, mac, 3)>0)
                        {
                                u_printf("local mac:%s\n", mac[1]);
                                strcat(response,mac[1]);
                        }

                        hfnet_assis_write(response, sizeof(response), ip_addr, port);

                        return 0;
                }
        }
        return len;
}

static int USER_FUNC socketa_recv_callback(uint32_t event,char *data,uint32_t len,uint32_t buf_len)
{
        if(event==HFNET_SOCKETA_CONNECTED)
        {
                hfnet_socketa_send("CONNECT OK",sizeof("CONNECT OK")-1,1000);
        }
        else if(event==HFNET_SOCKETA_DISCONNECTED)
        {
                hfuart_send(HFUART0,"TCP DISCONNECTED\r\n",sizeof("TCP DISCONNECTED\r\n")-1,1000);
        }
        else if(event==HFNET_SOCKETA_DATA_READY)
        {
                if(len>128)
                {
                        hfnet_socketa_send("INVALID PACKET\n",sizeof("INVALID PACKET\n")-1,1000);
```

```
                }
                data[len]=0;
                if(strcasecmp("GPIO NLINK LOW",data)==0)
                {
                        hftimer_stop(hnlink_timer);
                        hfgpio_fset_out_high(HFGPIO_F_NLINK);
                }
                else if(strcasecmp("GPIO NLINK HIGH",data)==0)
                {
                        hftimer_stop(hnlink_timer);
                        hfgpio_fset_out_low(HFGPIO_F_NLINK);
                }
                else if(strcasecmp("GPIO NLINK FALSH",data)==0)
                {
                        hftimer_start(hnlink_timer);
                }
                else
                {
                        hfuart_send(HFUART0,data,len,1000);
                }
                HF_Debug(DEBUG_LEVEL_LOW,"[%d]socketa recv %d bytes data %d %c\n",event,len,buf_len,data[0]);
        }
        //feedback 0, HSF transparent system will no more send data to serial port
        return 0;
}

static int USER_FUNC socketb_recv_callback(uint32_t event,char *data,uint32_t len,uint32_t buf_len)
{
        if(event==HFNET_SOCKETB_CONNECTED)
        {
                u_printf("socket b connected!\n");
        }
        else if(event==HFNET_SOCKETB_DISCONNECTED)
        {
                u_printf("socket b disconnected!\n");
        }
        else if(event==HFNET_SOCKETB_DATA_READY)
                HF_Debug(DEBUG_LEVEL_LOW,"[%d]socketb recv %d bytes data %d %c\n",event,len,buf_len,data[0]);

        return len;
}

static int USER_FUNC uart_recv_callback(uint32_t event,char *data,uint32_t len,uint32_t buf_len)
{
        HF_Debug(DEBUG_LEVEL_LOW,"[%d]uart recv %d bytes data %d\n",event,len,buf_len);
        return len;
}

static int hfsys_event_callback( uint32_t event_id,void * param)
{
        switch(event_id)
        {
                case HFE_WIFI_STA_CONNECTED:
                        u_printf("wifi sta connected!!\n");
                        break;
                case HFE_WIFI_STA_DISCONNECTED:
                        u_printf("wifi sta disconnected!!\n");
                        break;
                case HFE_DHCP_OK:
                {
                        uint32_t *p_ip;
                        p_ip = (uint32_t*)param;
                        u_printf("dhcp ok %08X!\n",*p_ip);
                }
                        break;
                case HFE_SMTLK_OK:
                        u_printf("smtlk ok!\n");
                        break;
                case HFE_CONFIG_RELOAD:
                        u_printf("reload!\n");
                        break;
                default:
                        break;
        }
        return 0;
}

static void show_reset_reason(void)
{
        uint32_t reset_reason=0;


        reset_reason = hfsys_get_reset_reason();

        if(reset_reason&HFSYS_RESET_REASON_ERESET)
        {
                u_printf("HFSYS_RESET_REASON_ERESET\n");
        }
```

```
	if(reset_reason&HFSYS_RESET_REASON_IRESET0)
	{
		u_printf("HFSYS_RESET_REASON_IRESET0\n");
	}
	if(reset_reason&HFSYS_RESET_REASON_IRESET1)
	{
		u_printf("HFSYS_RESET_REASON_IRESET1\n");
	}
	if(reset_reason==HFSYS_RESET_REASON_NORMAL)
	{
		u_printf("HFSYS_RESET_REASON_NORMAL\n");
	}
	if(reset_reason&HFSYS_RESET_REASON_WPS)
	{
		u_printf("HFSYS_RESET_REASON_WPS\n");
	}
	if(reset_reason&HFSYS_RESET_REASON_SMARTLINK_START)
	{
		u_printf("HFSYS_RESET_REASON_SMARTLINK_START\n");
	}
	if(reset_reason&HFSYS_RESET_REASON_SMARTLINK_OK)
	{
		u_printf("HFSYS_RESET_REASON_SMARTLINK_OK\n");
	}

	return;
}

int USER_FUNC app_main (void)
{
	time_t now=time(NULL);

	u_printf("[CALLBACK DEMO]sdk version(%s),the app_main start time is %d %s\n",\
		hfsys_get_sdk_version(),now,ctime(&now));

	if(hfsys_register_system_event((hfsys_event_callback_t)hfsys_event_callback)!=HF_SUCCESS)
	{
		u_printf("register system event fail\n");
	}

	if(hfgpio_fmap_check()!=0)
	{
		while(1)
		{
			HF_Debug(DEBUG_ERROR,"gpio map file error\n");
			msleep(1000);
		}
		return 0;
	}

	show_reset_reason();

	while(!hfnet_wifi_is_active())
	{
		msleep(50);
	}

	//if(hfnet_start_assis(ASSIS_PORT)!=HF_SUCCESS)
	if(hfnet_start_assis_ex(ASSIS_PORT,(hfnet_callback_t)assis_ex_recv_callback)!=HF_SUCCESS)
	{
		HF_Debug(DEBUG_WARN,"start httpd fail\n");
	}

	if(hfnet_start_httpd(HFTHREAD_PRIORITIES_MID)!=HF_SUCCESS)
	{
		HF_Debug(DEBUG_WARN,"start httpd fail\n");
	}
	if(hfnet_start_uart(HFTHREAD_PRIORITIES_LOW,(hfnet_callback_t)uart_recv_callback)!=HF_SUCCESS)
	{
		HF_Debug(DEBUG_WARN,"start uart fail!\n");
	}
	if(hfnet_start_socketa(HFTHREAD_PRIORITIES_LOW,(hfnet_callback_t)socketa_recv_callback)!=HF_SUCCESS)
	{
		HF_Debug(DEBUG_WARN,"start socketa fail\n");
	}
	if(hfnet_start_socketb(HFTHREAD_PRIORITIES_LOW,(hfnet_callback_t)socketb_recv_callback)!=HF_SUCCESS)
	{
		HF_Debug(DEBUG_WARN,"start socketb fail\n");
	}

	// create a auto-timer, trigger by each seconds.
	if((hnlink_timer = hftimer_create("NLINK-FALSH-TIMER",1000,true,1,nlink_falsh_timer_callback,0))==NULL)
	{
		u_printf("create timer fail\n");
	}

	return 1;
}
```

```
#endif
```

WebSide : http://gb.hi-flying.com

Socket test

```
#include
#include
#include
//#include
#include
#include
#include "../example.h"

#if (EXAMPLE_USE_DEMO==USER_SOCKET_DEMO)

//#define TEST_UART_SELECT

const int hf_gpio_fid_to_pid_map_table[HFM_MAX_FUNC_CODE]=
{
	HF_M_PIN(2),		//HFGPIO_F_JTAG_TCK
	HF_M_PIN(3),		//HFGPIO_F_JTAG_TDO
	HF_M_PIN(4),		//HFGPIO_F_JTAG_TDI
	HF_M_PIN(5),		//HFGPIO_F_JTAG_TMS
	HFM_NOPIN,			//HFGPIO_F_USBDP
	HFM_NOPIN,			//HFGPIO_F_USBDM
	HF_M_PIN(39),		//HFGPIO_F_UART0_TX
	HF_M_PIN(40),		//HFGPIO_F_UART0_RTS
	HF_M_PIN(41),		//HFGPIO_F_UART0_RX
	HF_M_PIN(42),		//HFGPIO_F_UART0_CTS

	HF_M_PIN(27),		//HFGPIO_F_SPI_MISO
	HF_M_PIN(28),		//HFGPIO_F_SPI_CLK
	HF_M_PIN(29),		//HFGPIO_F_SPI_CS
	HF_M_PIN(30),		//HFGPIO_F_SPI_MOSI

	HFM_NOPIN,		//HFGPIO_F_UART1_TX,
	HFM_NOPIN,		//HFGPIO_F_UART1_RTS,
	HFM_NOPIN,		//HFGPIO_F_UART1_RX,
	HFM_NOPIN,		//HFGPIO_F_UART1_CTS,

	HF_M_PIN(43),		//HFGPIO_F_NLINK
	HF_M_PIN(44),		//HFGPIO_F_NREADY
	HF_M_PIN(45),		//HFGPIO_F_NRELOAD
	HF_M_PIN(7),		//HFGPIO_F_SLEEP_RQ
	HF_M_PIN(8),		//HFGPIO_F_SLEEP_ON

	HFM_NOPIN,			//HFGPIO_F_RESERVE0
	HFM_NOPIN,			//HFGPIO_F_RESERVE1
	HFM_NOPIN,			//HFGPIO_F_RESERVE2
	HFM_NOPIN,			//HFGPIO_F_RESERVE3
	HFM_NOPIN,			//HFGPIO_F_RESERVE4
	HFM_NOPIN,			//HFGPIO_F_RESERVE5

	HFM_NOPIN,		//HFGPIO_F_USER_DEFINE
};

const hfat_cmd_t user_define_at_cmds_table[]=
{
	{NULL,NULL,NULL,NULL} //the last item must be null
};

USER_FUNC int tcp_connect_server()
{
	int fd;
	int tmp=1;
	struct sockaddr_in addr;

	memset((char*)&addr,0,sizeof(addr));
```

```
	addr.sin_family = AF_INET;
	addr.sin_port = htons(10001);
	addr.sin_addr.s_addr=inet_addr("10.10.100.150");
	fd = socket(AF_INET, SOCK_STREAM, 0);
	if(fd<0)
		return -1;

	tmp=1;
	if(setsockopt(fd, SOL_SOCKET,SO_KEEPALIVE,&tmp,sizeof(tmp))<0)
	{
		u_printf("set SO_KEEPALIVE fail\n");
	}
	tmp = 60;//60s
	if(setsockopt(fd, IPPROTO_TCP,TCP_KEEPIDLE,&tmp,sizeof(tmp))<0)
	{
		u_printf("set TCP_KEEPIDLE fail\n");
	}
	tmp = 6;
	if(setsockopt(fd, IPPROTO_TCP,TCP_KEEPINTVL,&tmp,sizeof(tmp))<0)
	{
		u_printf("set TCP_KEEPINTVL fail\n");
	}
	tmp = 5;
	if(setsockopt(fd, IPPROTO_TCP,TCP_KEEPCNT,&tmp,sizeof(tmp))<0)
	{
		u_printf("set TCP_KEEPCNT fail\n");
	}

	if (connect(fd, (struct sockaddr *)&addr, sizeof(addr))< 0)
	{
		close(fd);
		return -1;
	}
	u_printf("connect ok!\n");

	return fd;
}

USER_FUNC void test_socket_start(void)
{
	int fd=-1;
	int recv_num=0;
	char recv[32]={0};
	uint8_t mac_addr[6]={0};
	int ufd,ret,maxfd;
	int uart_fd;
	fd_set rset;
	struct timeval timeout;
	struct sockaddr_in addr;
	int alen=sizeof(struct sockaddr_in);

	memset((char*)&addr,0,sizeof(addr));
	addr.sin_family = AF_INET;
	addr.sin_port = htons(10000);
	addr.sin_addr.s_addr=htonl(INADDR_ANY);
	ufd = socket(AF_INET, SOCK_DGRAM, 0);
	if(ufd<0)
	{
		u_printf("create udp socket fail\n");
	}
	bind(ufd, (struct sockaddr*)&addr, sizeof(addr));
	maxfd=ufd;
	FD_ZERO(&rset);
	fd = tcp_connect_server();
	uart_fd = (int)hfuart_open(0);
	while(1)
	{
		maxfd = ufd;
		if(maxfd < fd)
		{
			maxfd = fd;
		}
		FD_ZERO(&rset);
		if(fd >= 0)
		{
			FD_SET(fd,&rset);
		}
		FD_SET(ufd,&rset);
#ifdef TEST_UART_SELECT
		FD_SET(uart_fd,&rset);
		if(maxfd < uart_fd)
			maxfd=uart_fd;
#endif
		timeout.tv_sec= 3;
		timeout.tv_usec= 0;
		ret = hfuart_select(maxfd+1, &rset, NULL, NULL, &timeout);
		if(ret<=0)
```

```
                continue;
        if (FD_ISSET(fd, &rset))
        {
                if((recv_num=recv(fd,recv,sizeof(recv),0))>0)
                {
                        u_printf("recv data bytes:%d\n",recv_num);
                }
                else
                {
                        close(fd);
                        fd=-1;
                        u_printf("tcp disconnectd!\n");
                }
        }
        else if(FD_ISSET(ufd,&rset))
        {
                alen=sizeof(struct sockaddr_in);
                if((recv_num=hfnet_recvfrom(ufd,recv,sizeof(recv),0,(struct sockaddr*)&addr,\
                                                (socklen_t*)&alen,(char*)mac_addr))>0)
                {
                        u_printf("recv data bytes:%d mac:%02X:%02X:%02X:%02X:%02X:%02X\n",
                        recv_num,
                        mac_addr[0],mac_addr[1],mac_addr[2],mac_addr[3],mac_addr[4],mac_addr[5]);
                }
        }
        else if(FD_ISSET(uart_fd,&rset))
        {
                recv_num=hfuart_recv((hfuart_handle_t)uart_fd,recv,sizeof(recv)-1,0);
                if(recv_num>0)
                {
                        recv[recv_num]=0;
                        u_printf("recv data bytes:%d %s\n",recv_num,recv);
                }
        }

    }

    return;

}



int USER_FUNC app_main (void)
{
    time_t now=time(NULL);

    HF_Debug(DEBUG_LEVEL,"[FILE DEMO]sdk version(%s),the app_main start time is %d %s\n",\
                                                        hfsys_get_sdk_version(),now,ctime(&now));
    if(hfgpio_fmap_check()!=0)
    {
        while(1)
        {
            HF_Debug(DEBUG_ERROR,"gpio map file error\n");
            msleep(1000);
        }
        //return 0;
    }

    while(!hfnet_wifi_is_active())
    {
        msleep(50);
    }

    if(hfnet_start_assis(ASSIS_PORT)!=HF_SUCCESS)
    {
        HF_Debug(DEBUG_WARN,"start httpd fail\n");
    }

    if(hfnet_start_httpd(HFTHREAD_PRIORITIES_MID)!=HF_SUCCESS)
    {
        HF_Debug(DEBUG_WARN,"start httpd fail\n");
    }
#ifndef TEST_UART_SELECT
    if(hfnet_start_uart(HFTHREAD_PRIORITIES_LOW,NULL)!=HF_SUCCESS)
    {
        HF_Debug(DEBUG_WARN,"start uart fail!\n");
    }
#endif
    test_socket_start();

    return 1;

}

#endif
```

WebSide : http://gb.hi-flying.com

Socketa_callback

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

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

WebSide : http://gb.hi-flying.com

This file is decompiled from a .CHM file
by an UNREGISTERED version of Easy CHM.
You can download Easy CHM at : http://www.eTextWizard.com

SSL Test

```
#include
#include
#include

#include
#include
#include

#include "../example.h"

#if (EXAMPLE_USE_DEMO==SSL_TEST_DEMO)

const int hf_gpio_fid_to_pid_map_table[HFM_MAX_FUNC_CODE]=
{
          HF_M_PIN(2),        //HFGPIO_F_JTAG_TCK
          HFM_NOPIN,          //HFGPIO_F_JTAG_TDO
          HFM_NOPIN,          //HFGPIO_F_JTAG_TDI
          HF_M_PIN(5),        //HFGPIO_F_JTAG_TMS
```

```
        HFM_NOPIN,                  //HFGPIO_F_USBDP
        HFM_NOPIN,                  //HFGPIO_F_USBDM
        HF_M_PIN(39),       //HFGPIO_F_UART0_TX
        HF_M_PIN(40),       //HFGPIO_F_UART0_RTS
        HF_M_PIN(41),       //HFGPIO_F_UART0_RX
        HF_M_PIN(42),       //HFGPIO_F_UART0_CTS

        HF_M_PIN(27),       //HFGPIO_F_SPI_MISO
        HF_M_PIN(28),       //HFGPIO_F_SPI_CLK
        HF_M_PIN(29),       //HFGPIO_F_SPI_CS
        HF_M_PIN(30),       //HFGPIO_F_SPI_MOSI

        HFM_NOPIN,          //HFGPIO_F_UART1_TX,
        HFM_NOPIN,          //HFGPIO_F_UART1_RTS,
        HFM_NOPIN,          //HFGPIO_F_UART1_RX,
        HFM_NOPIN,          //HFGPIO_F_UART1_CTS,

        HF_M_PIN(43),       //HFGPIO_F_NLINK
        HF_M_PIN(44),       //HFGPIO_F_NREADY
        HF_M_PIN(45),       //HFGPIO_F_NRELOAD
        HF_M_PIN(7),        //HFGPIO_F_SLEEP_RQ
        HF_M_PIN(8),        //HFGPIO_F_SLEEP_ON

        HF_M_PIN(15),               //HFGPIO_F_WPS
        HFM_NOPIN,                  //HFGPIO_F_RESERVE1
        HFM_NOPIN,                  //HFGPIO_F_RESERVE2
        HFM_NOPIN,                  //HFGPIO_F_RESERVE3
        HFM_NOPIN,                  //HFGPIO_F_RESERVE4
        HFM_NOPIN,                  //HFGPIO_F_RESERVE5

        HFM_NOPIN,          //HFGPIO_F_USER_DEFINE
};

/* return 1 is a ipaddress */
int addressis_ip(const char * ipaddr)
{
        char ii, ipadd;
        int i, j;

        ii=0;
        for (j= 0; j< 4; j++)
        {
                ipadd=0;
                for (i=0; i< 4; i++, ii++)
                {
                        if (*(ipaddr+ii)=='.')
                                if (i== 0)
                                        return 0;          //the first shall not be '.'
                                else
                                {
                                        ii++;
                                        break;                      //this feild finished
                                }
                        else if ((i==3)&&(j!=3))     //not the last feild, the number shall less than 4 bits
                                return 0;
                        else if ((*(ipaddr+ii) > '9')||(*(ipaddr+ii) < '0'))
                        {
                                if ((*(ipaddr+ii) == '\0')&&(j==3)&&(i!=0))
                                {
                                        break;
                                }
                                else
                                        return 0;                   //pls input number
                        }
                        else
                                ipadd= ipadd*10+(*(ipaddr+ii)-'0');
                        if (ipadd > 255)
                                return 0;
                }
        }
        return 1;
}

int tcp_connect_ssl_server(char *url)
{
        int fd;
        struct sockaddr_in addr;
        char *addrp=url;

        if((memcmp(url, "HTTPS://", 8)==0)||(memcmp(url, "https://", 8)==0))
                addrp= (char *)(url+8);

        ip_addr_t dest_addr;
        if(is_ipaddress((const char *)(addrp)) !=1 )
        {
                if(netconn_gethostbyname((const char *)(addrp), &dest_addr) !=HF_SUCCESS)
                        return -1;
        }
```

```
        else
                inet_aton((char *)(addrp), (ip_addr_t *) &dest_addr);

        memset((char*)&addr,0,sizeof(addr));
        addr.sin_family = AF_INET;
        addr.sin_port = htons(443);
        addr.sin_addr.s_addr=dest_addr.addr;
        fd = socket(AF_INET, SOCK_STREAM, 0);
        if(fd<0)
                return -1;

        if (connect(fd, (struct sockaddr *)&addr, sizeof(addr))< 0)
        {
                close(fd);
                return -1;
        }

        return fd;
}

char ssl_url[101];
char ssl_recvbuf[1000];
static void my_ssl_test(char *url, char *sendbuf, int sendnum)//a SSL test
{
        InitMemoryTracker();//for debug, it can show how many memory used in SSL
        CyaSSL_Debugging_ON();//for debug

        CyaSSL_Init();
        CYASSL_METHOD*  method  = 0;
        CYASSL_CTX*     ctx     = 0;
        CYASSL*         ssl     = 0;
        int sockfd;

        method=CyaSSLv3_client_method();
        if (method == NULL)
                HF_Debug(DEBUG_LEVEL_LOW, "unable to get method");

        ctx = CyaSSL_CTX_new(method);
        if (ctx == NULL)
        {
                HF_Debug(DEBUG_LEVEL_LOW, "unable to get ctx");
                return;
        }

        CyaSSL_CTX_set_verify(ctx, SSL_VERIFY_NONE, 0);//disable verify certificates

        ssl = CyaSSL_new(ctx);
        if (ssl == NULL)
        {
                HF_Debug(DEBUG_LEVEL_LOW, "unable to get SSL object");
                goto FREE_CTX;
        }

        sockfd=tcp_connect_ssl_server(url);
        if(sockfd<0)
        {
                HF_Debug(DEBUG_LEVEL_LOW, "create socket error");
                goto FREE_SSL;
        }

        CyaSSL_set_fd(ssl, sockfd);
        if (CyaSSL_connect(ssl) != SSL_SUCCESS)
        {
                int  err = CyaSSL_get_error(ssl, 0);
                char buffer[80];
                HF_Debug(DEBUG_LEVEL_LOW, "err = %d, %s\n", err,CyaSSL_ERR_error_string(err, buffer));
                HF_Debug(DEBUG_LEVEL_LOW, "SSL_connect failed");
        }

        if (CyaSSL_write(ssl, sendbuf, sendnum) != sendnum)
        HF_Debug(DEBUG_LEVEL_LOW,"SSL_write failed");

        int recvlen;
        recvlen = CyaSSL_read(ssl, ssl_recvbuf, sizeof(ssl_recvbuf)-1);
        if (recvlen > 0)
        {
                HF_Debug(DEBUG_LEVEL_LOW,"Server response: recv start ----------------------------------------\n");
                CyaSSL_Debugging_OFF();
                hfuart_send(HFUART0, ssl_recvbuf, recvlen,1000);

                while (1)
                {
                        recvlen = CyaSSL_read(ssl, ssl_recvbuf, sizeof(ssl_recvbuf)-1);
                        if (recvlen > 0)
                                hfuart_send(HFUART0, ssl_recvbuf, recvlen,1000);
                        else
                                break;
                }
```

```
                CyaSSL_Debugging_ON();
                HF_Debug(DEBUG_LEVEL_LOW,"\n---------------------------------------- recv End!\n");
        }
        else if (recvlen < 0)
        {
                int readErr = CyaSSL_get_error(ssl, 0);
                if (readErr != SSL_ERROR_WANT_READ)
                        HF_Debug(DEBUG_LEVEL_LOW, "CyaSSL_read failed");
        }

FREE_SSL:
        CyaSSL_shutdown(ssl);
        CyaSSL_free(ssl);
FREE_CTX:
        CyaSSL_CTX_free(ctx);
        close(sockfd);

        CyaSSL_Debugging_OFF();//close debug
        ShowMemoryTracker();//peek into how memory was used
}

static USER_FUNC int set_ssl_addr(pat_session_t s,int argc,char *argv[],char *rsp,int len)
{
        if( 0 == argc )
        {
                hffile_userbin_read(0, ssl_url, 100);
                sprintf(rsp, "%s=%s", rsp, ssl_url);
                return 0;
        }
        else if( 1 == argc )
        {
                if((strlen(argv[0]) > 1)&&(strlen(argv[0]) < 100))
                {
                        hffile_userbin_write(0, argv[0], strlen(argv[0])+1);
                        return 0;
                }
                else
                        return -1;
        }
        else
                return -1;

}

const hfat_cmd_t user_define_at_cmds_table[]=
{
        {"SSLADDR", set_ssl_addr, "   AT+SSLADDR: Get/Set address for SSL.\r\n", NULL},//add a AT cmd for SSL
        {NULL,NULL,NULL,NULL} //the last item must be null
};

static int USER_FUNC uart_recv_callback(uint32_t event,char *data,uint32_t len,uint32_t buf_len)
{
        //HF_Debug(DEBUG_LEVEL_LOW,"[%d]uart recv %d bytes data %d\n",event,len,buf_len);
        if((memcmp(data, "POST", 4)==0)||(memcmp(data, "GET", 3)==0))
        {
                hffile_userbin_read(0, ssl_url, 100);
                my_ssl_test(ssl_url, data, len);//do SSL Get/Post
                return 0;
        }
        return len;
}

static void show_reset_reason(void)
{
        uint32_t reset_reason=0;

        reset_reason = hfsys_get_reset_reason();

        if(reset_reason&HFSYS_RESET_REASON_ERESET)
        {
                u_printf("HFSYS_RESET_REASON_ERESET\n");
        }
        if(reset_reason&HFSYS_RESET_REASON_IRESET0)
        {
                u_printf("HFSYS_RESET_REASON_IRESET0\n");
        }
        if(reset_reason&HFSYS_RESET_REASON_IRESET1)
        {
                u_printf("HFSYS_RESET_REASON_IRESET1\n");
        }
        if(reset_reason==HFSYS_RESET_REASON_NORMAL)
        {
                u_printf("HFSYS_RESET_REASON_NORMAL\n");
        }
        if(reset_reason&HFSYS_RESET_REASON_WPS)
        {
                u_printf("HFSYS_RESET_REASON_WPS\n");
```

```
		}
		if(reset_reason&HFSYS_RESET_REASON_SMARTLINK_START)
		{
			u_printf("HFSYS_RESET_REASON_SMARTLINK_START\n");
		}
		if(reset_reason&HFSYS_RESET_REASON_SMARTLINK_OK)
		{
			u_printf("HFSYS_RESET_REASON_SMARTLINK_OK\n");
		}

		return;
	}

	int USER_FUNC app_main (void)
	{
		time_t now=time(NULL);

		u_printf("[CALLBACK DEMO]sdk version(%s),the app_main start time is %d %s\n",\
							hfsys_get_sdk_version(),now,ctime(&now));
		if(hfgpio_fmap_check()!=0)
		{
			while(1)
			{
				HF_Debug(DEBUG_ERROR,"gpio map file error\n");
				msleep(1000);
			}
			return 0;
		}

		show_reset_reason();

		while(!hfnet_wifi_is_active())
		{
			msleep(50);
		}

		if(hfnet_start_assis(ASSIS_PORT)!=HF_SUCCESS)
		{
			HF_Debug(DEBUG_WARN,"start httpd fail\n");
		}

		//this is a new function, can define the stack size for UART thread
		if(hfnet_start_uart_ex(HFTHREAD_PRIORITIES_LOW,(hfnet_callback_t)uart_recv_callback, 1024+256)!=HF_SUCCESS)
		{
			HF_Debug(DEBUG_WARN,"start uart fail!\n");
		}

		return 1;
	}

	#endif
```

WebSide : http://gb.hi-flying.com

This file is decompiled from a .CHM file
by an UNREGISTERED version of Easy CHM.
You can download Easy CHM at : http://www.eTextWizard.com

Thread Test

```
#include
#include
#include
//#include
#include
#include
#include "../example.h"
#include "hfmsgq.h"

#if (EXAMPLE_USE_DEMO==USER_THREAD_DEMO)

const int hf_gpio_fid_to_pid_map_table[HFM_MAX_FUNC_CODE]=
{
	HF_M_PIN(2),		//HFGPIO_F_JTAG_TCK
```

```
        HF_M_PIN(3),        //HFGPIO_F_JTAG_TDO
        HF_M_PIN(4),        //HFGPIO_F_JTAG_TDI
        HF_M_PIN(5),        //HFGPIO_F_JTAG_TMS
        HFM_NOPIN,                  //HFGPIO_F_USBDP
        HFM_NOPIN,                  //HFGPIO_F_USBDM
        HF_M_PIN(39),       //HFGPIO_F_UART0_TX
        HF_M_PIN(40),       //HFGPIO_F_UART0_RTS
        HF_M_PIN(41),       //HFGPIO_F_UART0_RX
        HF_M_PIN(42),       //HFGPIO_F_UART0_CTS

        HF_M_PIN(27),       //HFGPIO_F_SPI_MISO
        HF_M_PIN(28),       //HFGPIO_F_SPI_CLK
        HF_M_PIN(29),       //HFGPIO_F_SPI_CS
        HF_M_PIN(30),       //HFGPIO_F_SPI_MOSI

        HFM_NOPIN,          //HFGPIO_F_UART1_TX,
        HFM_NOPIN,          //HFGPIO_F_UART1_RTS,
        HFM_NOPIN,          //HFGPIO_F_UART1_RX,
        HFM_NOPIN,          //HFGPIO_F_UART1_CTS,

        HF_M_PIN(43),       //HFGPIO_F_NLINK
        HF_M_PIN(44),       //HFGPIO_F_NREADY
        HF_M_PIN(45),       //HFGPIO_F_NRELOAD
        HF_M_PIN(7),        //HFGPIO_F_SLEEP_RQ
        HF_M_PIN(8),        //HFGPIO_F_SLEEP_ON

        HFM_NOPIN,                  //HFGPIO_F_RESERVE0
        HFM_NOPIN,                  //HFGPIO_F_RESERVE1
        HFM_NOPIN,                  //HFGPIO_F_RESERVE2
        HFM_NOPIN,                  //HFGPIO_F_RESERVE3
        HFM_NOPIN,                  //HFGPIO_F_RESERVE4
        HFM_NOPIN,                  //HFGPIO_F_RESERVE5

        HFM_NOPIN,          //HFGPIO_F_USER_DEFINE
};

const hfat_cmd_t user_define_at_cmds_table[]=
{
        {NULL,NULL,NULL,NULL} //the last item must be null
};

static int test_data=0;
hfthread_mutex_t test_lock=NULL_MUTEX;
void test_thread_start(void);

#define PRINTF(...)HF_Debug(DEBUG_LEVEL,__VA_ARGS__)

USER_FUNC void display_mallinfo(void)
{
//      struct mallinfo mi;

//      mi = mallinfo();

//      PRINTF("Total non-mmapped bytes (arena):       %d\n", mi.arena);
//      PRINTF("# of free chunks (ordblks):            %d\n", mi.ordblks);
//      PRINTF("# of free fastbin blocks (smblks):     %d\n", mi.smblks);
//      PRINTF("# of mapped regions (hblks):           %d\n", mi.hblks);
//      PRINTF("Bytes in mapped regions (hblkhd):      %d\n", mi.hblkhd);
//      PRINTF("Max. total allocated space (usmblks):  %d\n", mi.usmblks);
//      PRINTF("Free bytes held in fastbins (fsmblks): %d\n", mi.fsmblks);
//      PRINTF("Total allocated space (uordblks):      %d\n", mi.uordblks);
//      PRINTF("Total free space (fordblks):           %d\n", mi.fordblks);
//      PRINTF("Topmost releasable block (keepcost):   %d\n", mi.keepcost);
}

#define TEST_TIMER_ID                   (1)

void USER_FUNC test_timer_callback1( 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);//when create, the setting of auto_reload is false, restart the timer manually
                }
}

static hftimer_handle_t test_timer=NULL;
static HFMSGQ_HANDLE test_msgq=NULL;

USER_FUNC static void test_thread_func(void* arg)
{
        int fd,id,ret;
        int tmp=1,recv_num=0;
        char recv[32]={0};
```

```
	fd_set rset;
	struct timeval timeout;
	struct sockaddr_in addr;

	id = (int)arg;
	memset((char*)&addr,0,sizeof(addr));

	addr.sin_family = AF_INET;
	addr.sin_port = htons(10001+id);
	addr.sin_addr.s_addr=htonl(INADDR_ANY);
	fd = socket(AF_INET, SOCK_DGRAM, 0);

	tmp=1;
	setsockopt(fd, SOL_SOCKET,SO_BROADCAST,&tmp,sizeof(tmp));
	hfnet_set_udp_broadcast_port_valid(10001,10001+id);
	bind(fd, (struct sockaddr*)&addr, sizeof(addr));
	hftimer_start(test_timer);
	FD_ZERO(&rset);
	//enable current thread watchdog , 30 seconds
	hfthread_enable_softwatchdog(NULL,30);
	//start timer
	while(1)
	{
		void *msg;
		hfthread_mutext_lock(test_lock);
		//comment out below code, module will reboot in 30 seconds
		hfthread_reset_softwatchdog(NULL);
		test_data=id;
		HF_Debug(DEBUG_LEVEL,"thread %d is running\n",test_data);
		hfthread_mutext_unlock(test_lock);

		if(hfmsgq_recv(test_msgq,&msg,10,0)==HF_SUCCESS)
		{
			u_printf("recv msg %p\n",msg);
		}
		FD_SET(fd,&rset);
		timeout.tv_sec= 3;
		timeout.tv_usec= 0;
		ret = select(fd+1, &rset, NULL, NULL, &timeout);
		if(ret<=0)
			continue;

		if (FD_ISSET(fd, &rset))
		{
			tmp = sizeof(addr);
			recv_num = recvfrom(fd, recv, 32, 0, (struct sockaddr *)&addr, (socklen_t*)&tmp);
			if(recv_num>0)
			{
				HF_Debug(DEBUG_LEVEL,"thread %d recvnum=%d\n",id,recv_num);
				sprintf(recv,"thread %d\r\n",id);
				sendto(fd,recv,strlen(recv),0,(struct sockaddr *)&addr,sizeof(addr));
			}
		}

	}

	hftimer_delete(test_timer);
}

USER_FUNC void test_thread_and_timer_start(void)
{
	if(hfthread_mutext_new(&test_lock)!=0)
	{
		HF_Debug(DEBUG_LEVEL,"create mutex fail\n");
		return;
	}
	// create a auto timer, trigger by each 1 seconds.
	if((test_timer = hftimer_create("TEST-TIMER",1000,true,TEST_TIMER_ID,test_timer_callback1,0))==NULL)
	{
		u_printf("create timer fail\n");
	}
	test_msgq = hfmsgq_create(10,4);
	if(test_msgq==NULL)
	{
		u_printf("create msgq fail\n");
	}

	hfthread_create(test_thread_func,"app_main_test",256,(void*)1,1,NULL,NULL);
	//hfthread_create(test_thread_func,"app_main_test1",256,(void*)2,1,NULL,NULL);
}


int USER_FUNC app_main (void)
{
	time_t now=time(NULL);

	HF_Debug(DEBUG_LEVEL,"[THREAD DEMO]sdk version(%s),the app_main start time is %d %s\n",\
                                                        hfsys_get_sdk_version(),now,ctime(&now));
```

```
	if(hfgpio_fmap_check()!=0)
	{
		while(1)
		{
			HF_Debug(DEBUG_ERROR,"gpio map file error\n");
			msleep(1000);
		}
		//return 0;
	}
	while(!hfnet_wifi_is_active())
	{
		msleep(50);
	}

	if(hfnet_start_assis(ASSIS_PORT)!=HF_SUCCESS)
	{
		HF_Debug(DEBUG_WARN,"start httpd fail\n");
	}
	if(hfnet_start_httpd(HFTHREAD_PRIORITIES_MID)!=HF_SUCCESS)
	{
		HF_Debug(DEBUG_WARN,"start httpd fail\n");
	}
	if(hfnet_start_uart(HFTHREAD_PRIORITIES_LOW,NULL)!=HF_SUCCESS)
	{
		HF_Debug(DEBUG_WARN,"start uart fail!\n");
	}
	if(hfnet_start_socketa(HFTHREAD_PRIORITIES_LOW,(hfnet_callback_t)NULL)!=HF_SUCCESS)
	{
		HF_Debug(DEBUG_WARN,"start socketa fail\n");
	}
	if(hfnet_start_socketb(HFTHREAD_PRIORITIES_LOW,(hfnet_callback_t)NULL)!=HF_SUCCESS)
	{
		HF_Debug(DEBUG_WARN,"start socketb fail\n");
	}

	test_thread_and_timer_start();

	return 1;

}

#endif
```

WebSide : http://gb.hi-flying.com

Timer Test

```
#include
#include
#include
//#include
#include
#include "../example.h"
#include

#define HFGPIO_F_TEST_TIMER                    HFGPIO_F_USER_DEFINE

#define REG_TIM0_CNT_INIT   (*(volatile unsigned long *) 0x4001B008)
#define REG_TIM0_CTRL                  (*(volatile unsigned long *) 0x4001B000)
#define REG_TIM0_CNT                   (*(volatile unsigned long *) 0x4001B00C)

#define TIME_CTRL_CNT_CLEAR                    (0x00000001)
#define TIME_CTRL_PAUSE                                (0x00000002)
#define TIME_CTRL_INTC                                         (0x00000004)
#define TIME_CTRL_INTE                                         (0x00000008)
#define TIME_CTRL_CNT_INIT_LOAD                (0x00000010)

#if (EXAMPLE_USE_DEMO==USER_TIMER_DEMO)

const int hf_gpio_fid_to_pid_map_table[HFM_MAX_FUNC_CODE]=
```

```
{
        HF_M_PIN(2),        //HFGPIO_F_JTAG_TCK
        HF_M_PIN(3),        //HFGPIO_F_JTAG_TDO
        HF_M_PIN(4),        //HFGPIO_F_JTAG_TDI
        HF_M_PIN(5),        //HFGPIO_F_JTAG_TMS
        HFM_NOPIN,                  //HFGPIO_F_USBDP
        HFM_NOPIN,                  //HFGPIO_F_USBDM
        HF_M_PIN(39),       //HFGPIO_F_UART0_TX
        HF_M_PIN(40),       //HFGPIO_F_UART0_RTS
        HF_M_PIN(41),       //HFGPIO_F_UART0_RX
        HF_M_PIN(42),       //HFGPIO_F_UART0_CTS
#ifdef __LPB100U__
        HFM_NOPIN,          //HFGPIO_F_SPI_MISO
        HFM_NOPIN,          //HFGPIO_F_SPI_CLK
        HFM_NOPIN,          //HFGPIO_F_SPI_CS
        HFM_NOPIN,          //HFGPIO_F_SPI_MOSI
#else
        HF_M_PIN(27),       //HFGPIO_F_SPI_MISO
        HF_M_PIN(28),       //HFGPIO_F_SPI_CLK
        HF_M_PIN(29),       //HFGPIO_F_SPI_CS
        HF_M_PIN(30),       //HFGPIO_F_SPI_MOSI
#endif
        HFM_NOPIN,          //HFGPIO_F_UART1_TX,
        HFM_NOPIN,          //HFGPIO_F_UART1_RTS,
        HFM_NOPIN,          //HFGPIO_F_UART1_RX,
        HFM_NOPIN,          //HFGPIO_F_UART1_CTS,

        HF_M_PIN(43),       //HFGPIO_F_NLINK
        HF_M_PIN(44),       //HFGPIO_F_NREADY
        HF_M_PIN(45),       //HFGPIO_F_NRELOAD
        HF_M_PIN(7),        //HFGPIO_F_SLEEP_RQ
        HF_M_PIN(8),        //HFGPIO_F_SLEEP_ON

        HFM_NOPIN,                  //HFGPIO_F_RESERVE0
        HFM_NOPIN,                  //HFGPIO_F_RESERVE1
        HFM_NOPIN,                  //HFGPIO_F_RESERVE2
        HFM_NOPIN,                  //HFGPIO_F_RESERVE3
        HFM_NOPIN,                  //HFGPIO_F_RESERVE4
        HFM_NOPIN,                  //HFGPIO_F_RESERVE5

        HF_M_PIN(11),       //HFGPIO_F_USER_DEFINE
};

const hfat_cmd_t user_define_at_cmds_table[]=
{
        {NULL,NULL,NULL,NULL} //the last item must be null
};

static int test_data=0;
hfthread_mutex_t test_lock=NULL_MUTEX;
void test_thread_start(void);

#define PRINTF(...)HF_Debug(DEBUG_LEVEL,__VA_ARGS__)

USER_FUNC void display_mallinfo(void)
{
//      struct mallinfo mi;

//      mi = mallinfo();

//      PRINTF("Total non-mmapped bytes (arena):       %d\n", mi.arena);
//      PRINTF("# of free chunks (ordblks):            %d\n", mi.ordblks);
//      PRINTF("# of free fastbin blocks (smblks):     %d\n", mi.smblks);
//      PRINTF("# of mapped regions (hblks):           %d\n", mi.hblks);
//      PRINTF("Bytes in mapped regions (hblkhd):      %d\n", mi.hblkhd);
//      PRINTF("Max. total allocated space (usmblks):  %d\n", mi.usmblks);
//      PRINTF("Free bytes held in fastbins (fsmblks): %d\n", mi.fsmblks);
//      PRINTF("Total allocated space (uordblks):      %d\n", mi.uordblks);
//      PRINTF("Total free space (fordblks):           %d\n", mi.fordblks);
//      PRINTF("Topmost releasable block (keepcost):   %d\n", mi.keepcost);
}

#define TEST_TIMER_ID                   (1)
#define TEST_TIMER2_ID                  (2)
#define TEST_TIMER3_ID                  (3)

hftimer_handle_t test_timer_hardware=NULL;

void USER_FUNC 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);//when create, the setting of auto_reload is false, restart the timer manually
```

```
                }
                else if(hftimer_get_timer_id(htimer)==TEST_TIMER2_ID)
                {
                        //u_printf("TEST_TIMER_ID active\n");
                        if(hfgpio_fpin_is_high(HFGPIO_F_NLINK))
                        {
                                hftimer_change_period(htimer,1000);
                                hfgpio_fset_out_low(HFGPIO_F_NLINK);
                        }
                        else
                        {
                                hftimer_change_period(htimer,3000);
                                hfgpio_fset_out_high(HFGPIO_F_NLINK);
                        }

                        //hftimer_start(htimer);
                }
                else if(hftimer_get_timer_id(htimer)==TEST_TIMER3_ID)
                {
                        if(hfgpio_fpin_is_high(HFGPIO_F_TEST_TIMER))
                                hfgpio_fset_out_low(HFGPIO_F_TEST_TIMER);
                        else
                                hfgpio_fset_out_high(HFGPIO_F_TEST_TIMER);
                }
                else
                {
                        u_printf("%p\n",htimer);
                }
}


void app_init(void)
{
        u_printf("app_init\n\n");

}


USER_FUNC static void test_thread_func(void* arg)
{
        int fd,id;
        int tmp=1,recv_num=0;
        char recv[32]={0};
        //char *p;
        struct sockaddr_in addr;
        hftimer_handle_t test_timer=NULL;
        hftimer_handle_t test_timer2=NULL;

        id = (int)arg;
        memset((char*)&addr,0,sizeof(addr));

        addr.sin_family = AF_INET;
        addr.sin_port = htons(10001+id);
        addr.sin_addr.s_addr=htonl(INADDR_ANY);
        // = socket()
        fd = socket(AF_INET, SOCK_DGRAM, 0);
        bind(fd, (struct sockaddr*)&addr, sizeof(addr));
        tmp=1;
        setsockopt(fd, SOL_SOCKET,SO_BROADCAST,&tmp,sizeof(tmp));

        // create a auto timer, trigger by each 1 seconds.
        if((test_timer = hftimer_create("TEST-TIMER",1000,true,TEST_TIMER_ID,\
                                                        test_timer_callback,0))==NULL)
        {
                u_printf("create timer 1 fail\n");
        }
        if((test_timer2 = hftimer_create("TEST-TIMER2",1000,false,TEST_TIMER2_ID,\
                                                         test_timer_callback,0))==NULL)
        {
                u_printf("create timer 2 fail\n");
        }

#if 0
        //0.5ms hardware timer,hardware timer calculated by microseconds, only 1 hardware timer can be created.
        if((test_timer_hardware=hftimer_create("HDW-TIMER",500,true,TEST_TIMER3_ID,\
                                test_timer_callback,HFTIMER_FLAG_HARDWARE_TIMER))==NULL)
#else
        if((test_timer_hardware=hftimer_create("HDW-TIMER",110000,true,TEST_TIMER3_ID,\
                                test_timer_callback,HFTIMER_FLAG_HARDWARE_TIMER))==NULL)
#endif
        {
                u_printf("create  hardware timer fail\n");
        }
        // start timer
        hftimer_start(test_timer);
        hftimer_start(test_timer2);
        hftimer_start(test_timer_hardware);
        while(1)
```

```
	{
		REG_TIM0_CTRL |= TIME_CTRL_CNT_CLEAR;
		u_printf("cnt0=%d %08X %d\n",REG_TIM0_CNT,REG_TIM0_CTRL,REG_TIM0_CNT_INIT);
		u_printf("cnt1=%d %08X %d\n",REG_TIM0_CNT,REG_TIM0_CTRL,REG_TIM0_CNT_INIT);
		u_printf("cnt2=%d %08X %d\n",REG_TIM0_CNT,REG_TIM0_CTRL,REG_TIM0_CNT_INIT);
		REG_TIM0_CTRL |= TIME_CTRL_CNT_CLEAR;
		u_printf("cnt3=%d %08X %d\n",REG_TIM0_CNT,REG_TIM0_CTRL);
		msleep(1);
	}
	while(1)
	{
		hfthread_mutext_lock(test_lock);
		test_data=id;
		HF_Debug(DEBUG_LEVEL_LOW,"thread %d is running\n",test_data);
		msleep(3000);
		u_printf("counter:%u\n",hftimer_get_counter(test_timer_hardware));
		hfthread_mutext_unlock(test_lock);
		tmp = sizeof(addr);
		//recv_num = recvfrom(fd, recv, 32, 0, (struct sockaddr *)&addr, (socklen_t*)&tmp);
		if(recv_num>0)
		{
			HF_Debug(DEBUG_LEVEL,"thread %d recvnum=%d\n",id,recv_num);
			sprintf(recv,"thread %d\r\n",id);
			sendto(fd,recv,strlen(recv),0,(struct sockaddr *)&addr,sizeof(addr));
		}
	}

	hftimer_delete(test_timer);
}

USER_FUNC void test_thread_and_timer_start(void)
{
	if(hfthread_mutext_new(&test_lock)!=0)
	{
		HF_Debug(DEBUG_LEVEL,"create mutex fail\n");
		return;
	}

	hfthread_create(test_thread_func,"app_main_test",256,(void*)1,1,NULL,NULL);
	//hfthread_create(test_thread_func,"app_main_test1",256,(void*)2,1,NULL,NULL);

}


int USER_FUNC app_main (void)
{
	time_t now=time(NULL);

	HF_Debug(DEBUG_LEVEL,"sdk version(%s),the app_main start time is %d %s[TIMER DEMO]\n",\
                                                hfsys_get_sdk_version(),now,ctime(&now));
	if(hfgpio_fmap_check()!=0)
	{
		while(1)
		{
			HF_Debug(DEBUG_ERROR,"gpio map file error\n");
			msleep(1000);
		}
		return 0;
	}
	while(!hfnet_wifi_is_active())
	{
		msleep(50);
	}

	if(hfnet_start_assis(ASSIS_PORT)!=HF_SUCCESS)
	{
		HF_Debug(DEBUG_WARN,"start httpd fail\n");
	}
	if(hfnet_start_httpd(HFTHREAD_PRIORITIES_MID)!=HF_SUCCESS)
	{
		HF_Debug(DEBUG_WARN,"start httpd fail\n");
	}
	if(hfnet_start_uart(HFTHREAD_PRIORITIES_LOW,NULL)!=HF_SUCCESS)
	{
		HF_Debug(DEBUG_WARN,"start uart fail!\n");
	}
	if(hfnet_start_socketa(HFTHREAD_PRIORITIES_LOW,(hfnet_callback_t)NULL)!=HF_SUCCESS)
	{
		HF_Debug(DEBUG_WARN,"start socketa fail\n");
	}
	if(hfnet_start_socketb(HFTHREAD_PRIORITIES_LOW,(hfnet_callback_t)NULL)!=HF_SUCCESS)
	{
		HF_Debug(DEBUG_WARN,"start socketb fail\n");
	}

	test_thread_and_timer_start();

	return 1;
```

```
}
#endif
```

WebSide : http://gb.hi-flying.com

Uart Test

```
#include
#include
#include
#include
#include "../example.h"

#if  (EXAMPLE_USE_DEMO==UART_OP_DEMO)

const int hf_gpio_fid_to_pid_map_table[HFM_MAX_FUNC_CODE]=
{
        HF_M_PIN(2),        //HFGPIO_F_JTAG_TCK
        HFM_NOPIN,          //HFGPIO_F_JTAG_TDO
        HFM_NOPIN,          //HFGPIO_F_JTAG_TDI
        HF_M_PIN(5),        //HFGPIO_F_JTAG_TMS
        HFM_NOPIN,                  //HFGPIO_F_USBDP
        HFM_NOPIN,                  //HFGPIO_F_USBDM
        HF_M_PIN(39),       //HFGPIO_F_UART0_TX
        HF_M_PIN(40),       //HFGPIO_F_UART0_RTS
        HF_M_PIN(41),       //HFGPIO_F_UART0_RX
        HF_M_PIN(42),       //HFGPIO_F_UART0_CTS

        HF_M_PIN(27),       //HFGPIO_F_SPI_MISO
        HF_M_PIN(28),       //HFGPIO_F_SPI_CLK
        HF_M_PIN(29),       //HFGPIO_F_SPI_CS
        HF_M_PIN(30),       //HFGPIO_F_SPI_MOSI

        HF_M_PIN(23),       //HFGPIO_F_UART1_TX,
        HFM_NOPIN,          //HFGPIO_F_UART1_RTS,
        HF_M_PIN(8),        //HFGPIO_F_UART1_RX,
        HFM_NOPIN,          //HFGPIO_F_UART1_CTS,

        HF_M_PIN(43),       //HFGPIO_F_NLINK
        HF_M_PIN(44),       //HFGPIO_F_NREADY
        HF_M_PIN(45),       //HFGPIO_F_NRELOAD
        HF_M_PIN(7),        //HFGPIO_F_SLEEP_RQ
        HFM_NOPIN,          //HFGPIO_F_SLEEP_ON

        HF_M_PIN(15),               //HFGPIO_F_WPS
        HFM_NOPIN,                  //HFGPIO_F_RESERVE1
        HFM_NOPIN,                  //HFGPIO_F_RESERVE2
        HFM_NOPIN,                  //HFGPIO_F_RESERVE3
        HFM_NOPIN,                  //HFGPIO_F_RESERVE4
        HFM_NOPIN,                  //HFGPIO_F_RESERVE5

        HFM_NOPIN,          //HFGPIO_F_USER_DEFINE
};

const hfat_cmd_t user_define_at_cmds_table[]=
{
        {NULL,NULL,NULL,NULL} //the last item must be null
};


static void show_reset_reason(void)
{
        uint32_t reset_reason=0;


        reset_reason = hfsys_get_reset_reason();

        if(reset_reason&HFSYS_RESET_REASON_ERESET)
```

```
        {
                u_printf("HFSYS_RESET_REASON_ERESET\n");
        }
        if(reset_reason&HFSYS_RESET_REASON_IRESET0)
        {
                u_printf("HFSYS_RESET_REASON_IRESET0\n");
        }
        if(reset_reason&HFSYS_RESET_REASON_IRESET1)
        {
                u_printf("HFSYS_RESET_REASON_IRESET1\n");
        }
        if(reset_reason==HFSYS_RESET_REASON_NORMAL)
        {
                u_printf("HFSYS_RESET_REASON_NORMAL\n");
        }
        if(reset_reason&HFSYS_RESET_REASON_WPS)
        {
                u_printf("HFSYS_RESET_REASON_WPS\n");
        }
        if(reset_reason&HFSYS_RESET_REASON_SMARTLINK_START)
        {
                u_printf("HFSYS_RESET_REASON_SMARTLINK_START\n");
        }
        if(reset_reason&HFSYS_RESET_REASON_SMARTLINK_OK)
        {
                u_printf("HFSYS_RESET_REASON_SMARTLINK_OK\n");
        }

        return;
}

void uart_test(void* arg)
{
        hfuart_handle_t huart1;
        char *buf;
        int recv_bytes;

        huart1 = hfuart_open(1);
        if(huart1==NULL)
        {
                u_printf("open uart1 fail\n");
                goto exit_thread;
        }
        buf = (char*)hfmem_malloc(1000);
        if(buf==NULL)
        {
                u_printf("memory alloc fail\n");
                goto exit_thread;
        }
        while(1)
        {
                recv_bytes = hfuart_recv(huart1,buf,1000,1000);
                if(recv_bytes>0)
                {
                        //u_printf("uart1 recv %d bytes\n",recv_bytes);
                        hfuart_send(huart1,buf,recv_bytes,100);
                }
        }

exit_thread:
        if(buf!=NULL)
        {
                hfmem_free(buf);
        }
        hfuart_close(huart1);
        hfthread_destroy(NULL);
        return ;
}


int USER_FUNC app_main (void)
{
        time_t now=time(NULL);

        HF_Debug(DEBUG_LEVEL,"sdk version(%s),the app_main start time is %d %s\n",\
                                        hfsys_get_sdk_version(),now,ctime(&now));
        if(hfgpio_fmap_check()!=0)
        {
                while(1)
                {
                        HF_Debug(DEBUG_ERROR,"gpio map file error\n");
                        msleep(1000);
                }
                return 0;
        }

        show_reset_reason();
```

```
	while(!hfnet_wifi_is_active())
	{
		msleep(50);
	}
	if(hfnet_start_assis(ASSIS_PORT)!=HF_SUCCESS)
	{
		HF_Debug(DEBUG_WARN,"start httpd fail\n");
	}

	if(hfnet_start_httpd(HFTHREAD_PRIORITIES_MID)!=HF_SUCCESS)
	{
		HF_Debug(DEBUG_WARN,"start httpd fail\n");
	}
	if(hfnet_start_uart(HFTHREAD_PRIORITIES_LOW,(hfnet_callback_t)NULL)!=HF_SUCCESS)
	{
		HF_Debug(DEBUG_WARN,"start uart fail!\n");
	}

	hfthread_create(uart_test, "uart_demo", 256, NULL, HFTHREAD_PRIORITIES_LOW, NULL,NULL);

	return 1;
}

#endif
```

WebSide : http://gb.hi-flying.com

Uflash test

```
#include
#include
#include
//#include
#include
#include
#include "../example.h"

#if (EXAMPLE_USE_DEMO==USER_FLASH_DEMO)

const int hf_gpio_fid_to_pid_map_table[HFM_MAX_FUNC_CODE]=
{
	HF_M_PIN(2),		//HFGPIO_F_JTAG_TCK
	HF_M_PIN(3),		//HFGPIO_F_JTAG_TDO
	HF_M_PIN(4),		//HFGPIO_F_JTAG_TDI
	HF_M_PIN(5),		//HFGPIO_F_JTAG_TMS
	HFM_NOPIN,			//HFGPIO_F_USBDP
	HFM_NOPIN,			//HFGPIO_F_USBDM
	HF_M_PIN(39),		//HFGPIO_F_UART0_TX
	HF_M_PIN(40),		//HFGPIO_F_UART0_RTS
	HF_M_PIN(41),		//HFGPIO_F_UART0_RX
	HF_M_PIN(42),		//HFGPIO_F_UART0_CTS

	HF_M_PIN(27),		//HFGPIO_F_SPI_MISO
	HF_M_PIN(28),		//HFGPIO_F_SPI_CLK
	HF_M_PIN(29),		//HFGPIO_F_SPI_CS
	HF_M_PIN(30),		//HFGPIO_F_SPI_MOSI

	HFM_NOPIN,		//HFGPIO_F_UART1_TX,
	HFM_NOPIN,		//HFGPIO_F_UART1_RTS,
	HFM_NOPIN,		//HFGPIO_F_UART1_RX,
	HFM_NOPIN,		//HFGPIO_F_UART1_CTS,

	HF_M_PIN(43),		//HFGPIO_F_NLINK
	HF_M_PIN(44),		//HFGPIO_F_NREADY
	HF_M_PIN(45),		//HFGPIO_F_NRELOAD
	HF_M_PIN(7),		//HFGPIO_F_SLEEP_RQ
	HF_M_PIN(8),		//HFGPIO_F_SLEEP_ON

	HFM_NOPIN,			//HFGPIO_F_RESERVE0
	HFM_NOPIN,			//HFGPIO_F_RESERVE1
	HFM_NOPIN,			//HFGPIO_F_RESERVE2
```

```
        HFM_NOPIN,                  //HFGPIO_F_RESERVE3
        HFM_NOPIN,                  //HFGPIO_F_RESERVE4
        HFM_NOPIN,                  //HFGPIO_F_RESERVE5

        HFM_NOPIN,          //HFGPIO_F_USER_DEFINE
};

const hfat_cmd_t user_define_at_cmds_table[]=
{
        {NULL,NULL,NULL,NULL} //the last item must be null
};

#define PRINTF(...)HF_Debug(DEBUG_LEVEL,__VA_ARGS__)


USER_FUNC void test_uflash_start(void)
{
        uint32_t i;
        uint32_t value;
        int pages;

        hfuflash_erase_page(0,1);
        for(i=0;i < HFFLASH_PAGE_SIZE;)
        {
                if(hfuflash_write(i,(char*)&i,sizeof(i)) < sizeof(i))
                {
                        u_printf("uflash eof\n");
                        break;
                }
                i+=sizeof(i);
        }

        for(i=0;i < HFFLASH_PAGE_SIZE;)
        {
                if(hfuflash_read(i,(char*)&value,4) < 4)
                {
                        u_printf("uflash eof\n");
                        break;
                }
                u_printf("%d\n",value);
                i+=4;
        }
        pages = (HFUFLASH_SIZE+HFUFLASH1_SIZE)/HFFLASH_PAGE_SIZE;
        for(i=0;i < pages;i++)
        {
                u_printf("erase test %d\n",i);
                msleep(1000);
                if(hfuflash_erase_page(i*HFFLASH_PAGE_SIZE,pages-i)!=HF_SUCCESS)
                {
                        u_printf("test erase fail!\n");
                        return ;
                }
        }

        for(i=0;i < HFUFLASH_SIZE+HFUFLASH1_SIZE;)
        {
                if(hfuflash_write(i,(char*)&i,sizeof(i)) < sizeof(i))
                {
                        u_printf("uflash eof\n");
                        break;
                }
                if(hfuflash_read(i,(char*)&value,4) < 4)
                {
                        u_printf("uflash eof\n");
                        break;
                }
                if(value!=i)
                {
                        u_printf("test fail %d %d\n",i,value);
                }
                i+=sizeof(i);
        }

}


int USER_FUNC app_main (void)
{
        time_t now=time(NULL);

        HF_Debug(DEBUG_LEVEL,"[FILE DEMO]sdk version(%s),the app_main start time is %d %s\n",\
                                                        hfsys_get_sdk_version(),now,ctime(&now));
        if(hfgpio_fmap_check()!=0)
        {
                while(1)
                {
                        HF_Debug(DEBUG_ERROR,"gpio map file error\n");
                        msleep(1000);
```

```
                }
                //return 0;
        }

        while(!hfnet_wifi_is_active())
        {
                msleep(50);
        }

        if(hfnet_start_assis(ASSIS_PORT)!=HF_SUCCESS)
        {
                HF_Debug(DEBUG_WARN,"start httpd fail\n");
        }

        if(hfnet_start_httpd(HFTHREAD_PRIORITIES_MID)!=HF_SUCCESS)
        {
                HF_Debug(DEBUG_WARN,"start httpd fail\n");
        }
        if(hfnet_start_uart(HFTHREAD_PRIORITIES_LOW,NULL)!=HF_SUCCESS)
        {
                HF_Debug(DEBUG_WARN,"start uart fail!\n");
        }
        if(hfnet_start_socketa(HFTHREAD_PRIORITIES_LOW,(hfnet_callback_t)NULL)!=HF_SUCCESS)
        {
                HF_Debug(DEBUG_WARN,"start socketa fail\n");
        }
        if(hfnet_start_socketb(HFTHREAD_PRIORITIES_LOW,(hfnet_callback_t)NULL)!=HF_SUCCESS)
        {
                HF_Debug(DEBUG_WARN,"start socketb fail\n");
        }

        test_uflash_start();

        return 1;

}

#endif
```

WebSide : http://gb.hi-flying.com

Update Test

```
#include
#include
#include
#include
#include
#include "../example.h"
#include

#if (EXAMPLE_USE_DEMO==UPDATE_TEST_DEMO)

#define HFGPIO_F_UPGRADE_LED          HFGPIO_F_USER_DEFINE
#define HFGPIO_F_UPGRADE_GPIO                 (HFGPIO_F_USER_DEFINE+1)

const int hf_gpio_fid_to_pid_map_table[HFM_MAX_FUNC_CODE]=
{
        HF_M_PIN(2),        //HFGPIO_F_JTAG_TCK
        HF_M_PIN(3),        //HFGPIO_F_JTAG_TDO
        HF_M_PIN(4),        //HFGPIO_F_JTAG_TDI
        HF_M_PIN(5),        //HFGPIO_F_JTAG_TMS
        HFM_NOPIN,                  //HFGPIO_F_USBDP
        HFM_NOPIN,                  //HFGPIO_F_USBDM
        HF_M_PIN(39),       //HFGPIO_F_UART0_TX
        HF_M_PIN(40),       //HFGPIO_F_UART0_RTS
        HF_M_PIN(41),       //HFGPIO_F_UART0_RX
        HF_M_PIN(42),       //HFGPIO_F_UART0_CTS

        HF_M_PIN(27),       //HFGPIO_F_SPI_MISO
```

```
        HF_M_PIN(28),       //HFGPIO_F_SPI_CLK
        HF_M_PIN(29),       //HFGPIO_F_SPI_CS
        HF_M_PIN(30),       //HFGPIO_F_SPI_MOSI

        HF_M_PIN(23),       //HFGPIO_F_UART1_TX,
        HFM_NOPIN,          //HFGPIO_F_UART1_RTS,
        HF_M_PIN(8),        //HFGPIO_F_UART1_RX,
        HFM_NOPIN,          //HFGPIO_F_UART1_CTS,

        HF_M_PIN(43),       //HFGPIO_F_NLINK
        HFM_NOPIN,//HF_M_PIN(44),     //HFGPIO_F_NREADY
        HF_M_PIN(45),       //HFGPIO_F_NRELOAD
        HFM_NOPIN,//HF_M_PIN(7),      //HFGPIO_F_SLEEP_RQ
        HFM_NOPIN,//HF_M_PIN(8),      //HFGPIO_F_SLEEP_ON

        HFM_NOPIN,                  //HFGPIO_F_RESERVE0
        HFM_NOPIN,                  //HFGPIO_F_RESERVE1
        HFM_NOPIN,                  //HFGPIO_F_RESERVE2
        HFM_NOPIN,                  //HFGPIO_F_RESERVE3
        HFM_NOPIN,                  //HFGPIO_F_RESERVE4
        HFM_NOPIN,                  //HFGPIO_F_RESERVE5

        HF_M_PIN(44),       //HFGPIO_F_UPGRADE_LED
        HF_M_PIN(7),        //HFGPIO_F_UPGRADE_GPIO

};

static int USER_FUNC hf_atcmd_upgrade_sw(pat_session_t s,int argc,char *argv[],char *rsp,int len);
static int USER_FUNC test_update_as_http(char *purl,char *type);

const hfat_cmd_t user_define_at_cmds_table[]=
{
        {"UPGRADESW",hf_atcmd_upgrade_sw,"   AT+UPGRADESW=url\r\n",NULL},
        {NULL,NULL,NULL,NULL} //the last item must be null
};


static int USER_FUNC hf_atcmd_upgrade_sw(pat_session_t s,int argc,char *argv[],char *rsp,int len)
{
        if(argc<2)
        {
                return -1;
        }

        test_update_as_http(argv[0],argv[1]);

        return 0;
}


static int USER_FUNC uart_recv_callback(uint32_t event,char *data,uint32_t len,uint32_t buf_len)
{
        return len;
}

void USER_FUNC update_timer_callback( hftimer_handle_t htimer )
{
        if(hfgpio_fpin_is_high(HFGPIO_F_NREADY))
                hfgpio_fset_out_low(HFGPIO_F_NREADY);
        else
                hfgpio_fset_out_high(HFGPIO_F_NREADY);
}

static int USER_FUNC test_update_as_http(char *purl,char *type)
{
        httpc_req_t  http_req;
        char *content_data=NULL;
        char *temp_buf=NULL;
        parsed_url_t url={0};
        http_session_t hhttp=0;
        int total_size,read_size=0;
        int rv=0;
        tls_init_config_t  *tls_cfg=NULL;
        char *test_url=purl;
        hftimer_handle_t upg_timer=NULL;
        struct MD5Context md5_ctx;
        uint8_t digest[16]={0};
        HFUPDATE_TYPE_E  upg_type;

        bzero(&http_req,sizeof(http_req));
        http_req.type = HTTP_GET;
        http_req.version=HTTP_VER_1_1;

        if(strcasecmp(type,"wifi")==0)
        {
                upg_type = HFUPDATE_WIFIFW;
        }
        else
                upg_type = HFUPDATE_SW;
```

```
	if((temp_buf = (char*)hfmem_malloc(256))==NULL)
	{
		u_printf("no memory\n");
		rv= -HF_E_NOMEM;
		goto exit;
	}
	bzero(temp_buf,sizeof(temp_buf));
	if((rv=hfhttp_parse_URL(test_url,temp_buf , 256, &url))!=HF_SUCCESS)
	{
		goto exit;
	}

	if((rv=hfhttp_open_session(&hhttp,test_url,0,tls_cfg,3))!=HF_SUCCESS)
	{
		u_printf("http open fail\n");
		goto exit;
	}

	hfsys_disable_all_soft_watchdogs();
	hfupdate_start(upg_type);
	http_req.resource = url.resource;
	hfhttp_prepare_req(hhttp,&http_req,HDR_ADD_CONN_CLOSE);
	hfhttp_add_header(hhttp,"Range","bytes=0");
	if((rv=hfhttp_send_request(hhttp,&http_req))!=HF_SUCCESS)
	{
		u_printf("http send request fail\n");
		goto exit;
	}

	content_data = (char*)hfmem_malloc(256);
	if(content_data==NULL)
	{
		rv= -HF_E_NOMEM;
		goto exit;
	}
	total_size=0;
	bzero(content_data,256);

	if((upg_timer = hftimer_create("UPG-TIMER",100,true,1,update_timer_callback,0))==NULL)
	{
		u_printf("create timer 1 fail\n");
		goto exit;
	}

	hftimer_start(upg_timer);
	MD5Init(&md5_ctx);
	while((read_size=hfhttp_read_content(hhttp,content_data,256))>0)
	{
		hfupdate_write_file(upg_type, total_size,content_data, read_size);
		MD5Update(&md5_ctx,(uint8_t*)content_data,read_size);
		total_size+=read_size;
		u_printf("download file:[%d] [%d]\r",total_size,read_size);
	}
	MD5Final(digest,&md5_ctx);
	u_printf("read_size:%d digest is ",total_size);
	u_printf("%02x%02x%02x%02x",digest[0],digest[1],digest[2],digest[3]);
	u_printf("%02x%02x%02x%02x",digest[4],digest[5],digest[6],digest[7]);
	u_printf("%02x%02x%02x%02x",digest[8],digest[9],digest[10],digest[11]);
	u_printf("%02x%02x%02x%02x\n",digest[12],digest[13],digest[14],digest[15]);

	if(hfupdate_complete(upg_type,total_size)!=HF_SUCCESS)
	{
		u_printf("update software fail\n");
	}
exit:
	if(upg_timer!=NULL)
	{
		hftimer_delete(upg_timer);
		hftimer_delete(upg_timer);
	}
	if(temp_buf!=NULL)
		hfmem_free(temp_buf);
	if(content_data!=NULL)
		hfmem_free(content_data);
	if(hhttp!=0)
		hfhttp_close_session(&hhttp);
	hfgpio_fset_out_low(HFGPIO_F_NREADY);
	hfsys_enable_all_soft_watchdogs();
	return rv;
}

void USER_FUNC user_upgrade(void)
{
	int result=0;

	hfgpio_configure_fpin(HFGPIO_F_UPGRADE_GPIO,HFM_IO_TYPE_INPUT|HFPIO_DEFAULT);
	msleep(10);
```

```
        if(hfgpio_fpin_is_high(HFGPIO_F_UPGRADE_GPIO)==0)
        {
                result = hfupdate_auto_upgrade(0x20000000);
        }
        else
        {
                result = hfupdate_auto_upgrade(0);
        }

        if(result<0)
        {
                u_printf("no need to upgrade\n");
                return ;
        }
        else if(result==0)
        {
                u_printf("upgrade success!\n");
                while(1)
                {
                        hfgpio_fset_out_high(HFGPIO_F_UPGRADE_LED);
                        msleep(200);
                        hfgpio_fset_out_low(HFGPIO_F_UPGRADE_LED);
                        msleep(200);
                }
        }
        else
        {
                u_printf("upgrade fail %d\n",result);
                while(1)
                {
                        hfgpio_fset_out_low(HFGPIO_F_UPGRADE_LED);
                        msleep(1000);
                }
        }
}

int USER_FUNC app_main (void)
{
        time_t now=time(NULL);

        HF_Debug(DEBUG_LEVEL,"sdk version(%s),the app_main start time is %d %s[AT DEMO]\n",\
                                                hfsys_get_sdk_version(),now,ctime(&now));
        if(hfgpio_fmap_check()!=0)
        {
                while(1)
                {
                        HF_Debug(DEBUG_ERROR,"gpio map file error\n");
                        msleep(1000);
                }
        }
        while(!hfnet_wifi_is_active())
        {
                msleep(50);
        }
        user_upgrade();
        if(hfnet_start_assis(ASSIS_PORT)!=HF_SUCCESS)
        {
                HF_Debug(DEBUG_WARN,"start httpd fail\n");
        }
        if(hfnet_start_httpd(HFTHREAD_PRIORITIES_MID)!=HF_SUCCESS)
        {
                HF_Debug(DEBUG_WARN,"start httpd fail\n");
        }
        if(hfnet_start_uart(HFTHREAD_PRIORITIES_LOW,(hfnet_callback_t)uart_recv_callback)!=HF_SUCCESS)
        {
                HF_Debug(DEBUG_WARN,"start uart fail!\n");
        }


        return 1;

}

#endif
```

WebSide : http://gb.hi-flying.com

Urlcallback Test

```
#include
#include
#include
#include
//#include
#include
#include
#include "../example.h"

#if (EXAMPLE_USE_DEMO==USER_URL_DEMO)

static char user_define = -1;

const int hf_gpio_fid_to_pid_map_table[HFM_MAX_FUNC_CODE]=
{
          HF_M_PIN(2),        //HFGPIO_F_JTAG_TCK
          HFM_NOPIN,          //HFGPIO_F_JTAG_TDO
          HFM_NOPIN,          //HFGPIO_F_JTAG_TDI
          HF_M_PIN(5),        //HFGPIO_F_JTAG_TMS
          HFM_NOPIN,                  //HFGPIO_F_USBDP
          HFM_NOPIN,                  //HFGPIO_F_USBDM
          HF_M_PIN(39),       //HFGPIO_F_UART0_TX
          HF_M_PIN(40),       //HFGPIO_F_UART0_RTS
          HF_M_PIN(41),       //HFGPIO_F_UART0_RX
          HF_M_PIN(42),       //HFGPIO_F_UART0_CTS

          HF_M_PIN(27),       //HFGPIO_F_SPI_MISO
          HF_M_PIN(28),       //HFGPIO_F_SPI_CLK
          HF_M_PIN(29),       //HFGPIO_F_SPI_CS
          HF_M_PIN(30),       //HFGPIO_F_SPI_MOSI

          HFM_NOPIN,          //HFGPIO_F_UART1_TX,
          HFM_NOPIN,          //HFGPIO_F_UART1_RTS,
          HFM_NOPIN,          //HFGPIO_F_UART1_RX,
          HFM_NOPIN,          //HFGPIO_F_UART1_CTS,

          HF_M_PIN(43),       //HFGPIO_F_NLINK
          HF_M_PIN(44),       //HFGPIO_F_NREADY
          HF_M_PIN(45),       //HFGPIO_F_NRELOAD
          HF_M_PIN(7),        //HFGPIO_F_SLEEP_RQ
          HF_M_PIN(8),        //HFGPIO_F_SLEEP_ON

          HF_M_PIN(15),               //HFGPIO_F_WPS
          HFM_NOPIN,                  //HFGPIO_F_RESERVE1
          HFM_NOPIN,                  //HFGPIO_F_RESERVE2
          HFM_NOPIN,                  //HFGPIO_F_RESERVE3
          HFM_NOPIN,                  //HFGPIO_F_RESERVE4
          HFM_NOPIN,                  //HFGPIO_F_RESERVE5

          HFM_NOPIN,          //HFGPIO_F_USER_DEFINE
};
static int web_user_define(pat_session_t s,int argc,char *argv[],char *rsp,int len);

const hfat_cmd_t user_define_at_cmds_table[]=
{
          {"URT", web_user_define, "   AT+URT=Set/Get the return value of user defined webpage.\r\n", NULL},
          {NULL,NULL,NULL,NULL} //the last item must be null
};

static int USER_FUNC socketa_recv_callback(uint32_t event,char *data,uint32_t len,uint32_t buf_len)
{
          return len;
}

static int USER_FUNC socketb_recv_callback(uint32_t event,char *data,uint32_t len,uint32_t buf_len)
{
          return len;
}

static int USER_FUNC uart_recv_callback(uint32_t event,char *data,uint32_t len,uint32_t buf_len)
{
          return len;
}

static void show_reset_reason(void)
{
          uint32_t reset_reason=0;


          reset_reason = hfsys_get_reset_reason();
```

```
	if(reset_reason&HFSYS_RESET_REASON_ERESET)
	{
		u_printf("HFSYS_RESET_REASON_ERESET\n");
	}
	if(reset_reason&HFSYS_RESET_REASON_IRESET0)
	{
		u_printf("HFSYS_RESET_REASON_IRESET0\n");
	}
	if(reset_reason&HFSYS_RESET_REASON_IRESET1)
	{
		u_printf("HFSYS_RESET_REASON_IRESET1\n");
	}
	if(reset_reason==HFSYS_RESET_REASON_NORMAL)
	{
		u_printf("HFSYS_RESET_REASON_NORMAL\n");
	}
	if(reset_reason&HFSYS_RESET_REASON_WPS)
	{
		u_printf("HFSYS_RESET_REASON_WPS\n");
	}
	if(reset_reason&HFSYS_RESET_REASON_SMARTLINK_START)
	{
		u_printf("HFSYS_RESET_REASON_SMARTLINK_START\n");
	}
	if(reset_reason&HFSYS_RESET_REASON_SMARTLINK_OK)
	{
		u_printf("HFSYS_RESET_REASON_SMARTLINK_OK\n");
	}

	return;
}

static int web_user_define(pat_session_t s,int argc,char *argv[],char *rsp,int len)
{
	if(argc == 0)
		sprintf(rsp, "=%d", user_define);
	else
		user_define = atoi(argv[0]);
	return 0;
}

static char *strnstr(const char *s, const char *find, size_t slen)
{
	int i,flen;

	flen = strlen(find);
	if(flen>slen)
		return NULL;

	for(i=0;i<=slen-flen;i++)
	{
		if(s[i]==*find)
		{
			if(strncmp(s+i,find,slen-i)==0)
				return (char*)(s+i);
		}
	}

	return NULL;
}
static int http_get_value(char *str, char *value)
{
	char *tmp = str;
	char i = 0;

	while((*tmp != '&') && (*tmp != '\0'))
	{
		i++;
		tmp++;
	}

	strncpy(value, str, i);
	if(value[0] == '?')
		return 0;
	else
		return 1;
}
/*

User defined parameter supports a maximum of 70 bytes.
Rsp supports a maximum of 1400 bytes.

*/
static int hfhttpd_url_callback_test(char *url, char *rsp)
{
```

```
	char i = 0;
	char *p1, *fname_tmp;
	char value[30] = {0};
	char at_rsp[64] = {0};
	char at_cmd[50] = {0};
	char ret = -1;

	fname_tmp = url;


	if((p1 = strnstr(fname_tmp, "/AT+WSSSID=", strlen("/AT+WSSSID="))) != NULL)
	{
		ret = http_get_value(p1+strlen("/AT+WSSSID="), value);

		if(ret == 0)
		{
			hfat_send_cmd("AT+WSSSID\r\n", strlen("AT+WSSSID\r\n"), at_rsp, sizeof(at_rsp));
			sprintf(rsp, at_rsp);
		}
		else
		{
			sprintf(at_cmd, "%s%s\r\n", "AT+WSSSID=", value);
			hfat_send_cmd(at_cmd, strlen(at_cmd), at_rsp, sizeof(at_rsp));
			if(0 == strcmp("+ok", at_rsp))
				sprintf(rsp, "set ok.\r\n");
			else
				sprintf(rsp, "set fail.\r\n");
		}
		return 0;
	}
	else if((p1 = strnstr(fname_tmp, "/AT+WSKEY=", strlen("/AT+WSKEY="))) != NULL)
	{
		ret = http_get_value(p1+strlen("/AT+WSKEY="), value);

		if(ret == 0)
		{
			hfat_send_cmd("AT+WSKEY\r\n", strlen("AT+WSKEY\r\n"), at_rsp, sizeof(at_rsp));
			sprintf(rsp, at_rsp);
		}
		else
		{
			sprintf(at_cmd, "%s%s\r\n", "AT+WSKEY=", value);
			hfat_send_cmd(at_cmd, strlen(at_cmd), at_rsp, sizeof(at_rsp));
			if(0 == strcmp("+ok", at_rsp))
				sprintf(rsp, "set ok.\r\n");
			else
				sprintf(rsp, "set fail.\r\n");
		}
		return 0;
	}
	else if((p1 = strnstr(fname_tmp, "/AT+WMODE=", strlen("/AT+WMODE="))) != NULL)
	{
		ret = http_get_value(p1+strlen("/AT+WMODE="), value);

		if(ret == 0)
		{
			hfat_send_cmd("AT+WMODE\r\n", strlen("AT+WMODE\r\n"), at_rsp, sizeof(at_rsp));
			sprintf(rsp, at_rsp);
		}
		else
		{
			sprintf(at_cmd, "%s%s\r\n", "AT+WMODE=", value);
			hfat_send_cmd(at_cmd, strlen(at_cmd), at_rsp, sizeof(at_rsp));
			if(0 == strcmp("+ok", at_rsp))
				sprintf(rsp, "set ok.\r\n");
			else
				sprintf(rsp, "set fail.\r\n");
		}

		return 0;
	}
	else if((p1 = strnstr(fname_tmp, "/iweb.html", strlen("/iweb.html"))) != NULL)
	{
		/*************************************
		"iweb.html" is our internal upgrade webpage.
		If you want to replace it ,please return 0;
		If not, just return -1.
		*************************************/
		if( 0 == user_define)
			sprintf(rsp, "user define page.");
		return user_define;
	}

	return -1;
}

int USER_FUNC app_main (void)
{
```

```
	time_t now=time(NULL);

	HF_Debug(DEBUG_LEVEL,"[URL_CALLBACK DEMO]sdk version(%s),\
		the app_main start time is %d %s\n",hfsys_get_sdk_version(),now,ctime(&now));
	if(hfgpio_fmap_check()!=0)
	{
		while(1)
		{
			HF_Debug(DEBUG_ERROR,"gpio map file error\n");
			msleep(1000);
		}
		return 0;
	}

	show_reset_reason();

	while(!hfnet_wifi_is_active())
	{
		msleep(50);
	}
	if(hfnet_start_assis(ASSIS_PORT)!=HF_SUCCESS)
	{
		HF_Debug(DEBUG_WARN,"start httpd fail\n");
	}

	if(hfnet_start_uart(HFTHREAD_PRIORITIES_LOW,(hfnet_callback_t)uart_recv_callback)!=HF_SUCCESS)
	{
		HF_Debug(DEBUG_WARN,"start uart fail!\n");
	}
	if(hfnet_start_socketa(HFTHREAD_PRIORITIES_LOW,(hfnet_callback_t)socketa_recv_callback)!=HF_SUCCESS)
	{
		HF_Debug(DEBUG_WARN,"start socketa fail\n");
	}
	if(hfnet_start_socketb(HFTHREAD_PRIORITIES_LOW,(hfnet_callback_t)socketb_recv_callback)!=HF_SUCCESS)
	{
		HF_Debug(DEBUG_WARN,"start socketb fail\n");
	}
	if(hfnet_start_httpd(HFTHREAD_PRIORITIES_MID)!=HF_SUCCESS)
	{
		HF_Debug(DEBUG_WARN,"start httpd fail\n");
	}

	if(HF_SUCCESS != (hfhttpd_url_callback_register(hfhttpd_url_callback_test, 0)))
	{
		HF_Debug(DEBUG_LEVEL, "register url callback fail\r\n");
	}

	return 1;
}

#endif
```

WebSide : http://gb.hi-flying.com

This file is decompiled from a .CHM file
by an UNREGISTERED version of Easy CHM.
You can download Easy CHM at : http://www.eTextWizard.com

Wifi Test

```
#include
#include
#include
#include
#include "../example.h"

#if  (EXAMPLE_USE_DEMO==WIFI_TEST_DEMO)

const int hf_gpio_fid_to_pid_map_table[HFM_MAX_FUNC_CODE]=
{
	HF_M_PIN(2),		//HFGPIO_F_JTAG_TCK
	HFM_NOPIN,		//HFGPIO_F_JTAG_TDO
	HFM_NOPIN,		//HFGPIO_F_JTAG_TDI
	HF_M_PIN(5),		//HFGPIO_F_JTAG_TMS
```

```
        HFM_NOPIN,                  //HFGPIO_F_USBDP
        HFM_NOPIN,                  //HFGPIO_F_USBDM
        HF_M_PIN(39),       //HFGPIO_F_UART0_TX
        HF_M_PIN(40),       //HFGPIO_F_UART0_RTS
        HF_M_PIN(41),       //HFGPIO_F_UART0_RX
        HF_M_PIN(42),       //HFGPIO_F_UART0_CTS

        HF_M_PIN(27),       //HFGPIO_F_SPI_MISO
        HF_M_PIN(28),       //HFGPIO_F_SPI_CLK
        HF_M_PIN(29),       //HFGPIO_F_SPI_CS
        HF_M_PIN(30),       //HFGPIO_F_SPI_MOSI

        HFM_NOPIN,          //HFGPIO_F_UART1_TX,
        HFM_NOPIN,          //HFGPIO_F_UART1_RTS,
        HFM_NOPIN,          //HFGPIO_F_UART1_RX,
        HFM_NOPIN,          //HFGPIO_F_UART1_CTS,

        HF_M_PIN(43),       //HFGPIO_F_NLINK
        HF_M_PIN(44),       //HFGPIO_F_NREADY
        HF_M_PIN(45),       //HFGPIO_F_NRELOAD
        HF_M_PIN(7),        //HFGPIO_F_SLEEP_RQ
        HF_M_PIN(8),        //HFGPIO_F_SLEEP_ON

        HF_M_PIN(15),               //HFGPIO_F_WPS
        HFM_NOPIN,                  //HFGPIO_F_RESERVE1
        HFM_NOPIN,                  //HFGPIO_F_RESERVE2
        HFM_NOPIN,                  //HFGPIO_F_RESERVE3
        HFM_NOPIN,                  //HFGPIO_F_RESERVE4
        HFM_NOPIN,                  //HFGPIO_F_RESERVE5

        HFM_NOPIN,          //HFGPIO_F_USER_DEFINE
};

const hfat_cmd_t user_define_at_cmds_table[]=
{
        {NULL,NULL,NULL,NULL} //the last item must be null
};

static int hfsys_event_callback( uint32_t event_id,void * param)
{
        switch(event_id)
        {
                case HFE_WIFI_STA_CONNECTED:
                        u_printf("wifi sta connected!!\n");
                        break;
                case HFE_WIFI_STA_DISCONNECTED:
                        u_printf("wifi sta disconnected!!\n");
                        break;
                case HFE_DHCP_OK:
                {
                        uint32_t *p_ip;
                        p_ip = (uint32_t*)param;
                        u_printf("dhcp ok %08X!\n",*p_ip);
                }
                        break;
                case HFE_SMTLK_OK:
                        u_printf("smtlk ok!\n");
                        return -1;
                        break;
                case HFE_CONFIG_RELOAD:
                        u_printf("reload!\n");
                        break;
                default:
                        break;
        }
        return 0;
}

static int USER_FUNC socketa_recv_callback(uint32_t event,char *data,uint32_t len,uint32_t buf_len)
{

        if(event==HFNET_SOCKETA_DATA_READY)
                HF_Debug(DEBUG_LEVEL_LOW,"socketa recv %d bytes data %d\n",len,buf_len);
        else if(event==HFNET_SOCKETA_CONNECTED)
                u_printf("socket a connected!\n");
        else if(event==HFNET_SOCKETA_DISCONNECTED)
                u_printf("socket a disconnected!\n");

        return len;
}

static int USER_FUNC socketb_recv_callback(uint32_t event,char *data,uint32_t len,uint32_t buf_len)
{
        if(event==HFNET_SOCKETB_DATA_READY)
                HF_Debug(DEBUG_LEVEL_LOW,"socketb recv %d bytes data %d\n",len,buf_len);
        else if(event==HFNET_SOCKETB_CONNECTED)
                u_printf("socket b connected!\n");
        else if(event==HFNET_SOCKETB_DISCONNECTED)
```

```
                u_printf("socket b disconnected!\n");

        return len;
}

static int USER_FUNC uart_recv_callback(uint32_t event,char *data,uint32_t len,uint32_t buf_len)
{
        return len;
}

static void show_reset_reason(void)
{
        uint32_t reset_reason=0;


        reset_reason = hfsys_get_reset_reason();

        if(reset_reason&HFSYS_RESET_REASON_ERESET)
        {
                u_printf("HFSYS_RESET_REASON_ERESET\n");
        }
        if(reset_reason&HFSYS_RESET_REASON_IRESET0)
        {
                u_printf("HFSYS_RESET_REASON_IRESET0\n");
        }
        if(reset_reason&HFSYS_RESET_REASON_IRESET1)
        {
                u_printf("HFSYS_RESET_REASON_IRESET1\n");
        }
        if(reset_reason==HFSYS_RESET_REASON_NORMAL)
        {
                u_printf("HFSYS_RESET_REASON_NORMAL\n");
        }
        if(reset_reason&HFSYS_RESET_REASON_WPS)
        {
                u_printf("HFSYS_RESET_REASON_WPS\n");
        }
        if(reset_reason&HFSYS_RESET_REASON_SMARTLINK_START)
        {
                u_printf("HFSYS_RESET_REASON_SMARTLINK_START\n");
        }
        if(reset_reason&HFSYS_RESET_REASON_SMARTLINK_OK)
        {
                u_printf("HFSYS_RESET_REASON_SMARTLINK_OK\n");
        }

        return;
}

int hfwifi_scan_test( PWIFI_SCAN_RESULT_ITEM scan_ret)
{
        int i;

        u_printf("%s,", scan_ret->ssid);
        u_printf("%d,", scan_ret->auth);
        u_printf("%d,", scan_ret->encry);
        u_printf("%d,", scan_ret->channel);
        u_printf("%d,", scan_ret->rssi);
        u_printf("");
        for(i=0; i<6; i++)
                u_printf("%X", ((uint8_t*)scan_ret->mac)[i]);
        u_printf("\r\n");
        return 0;
}

void wifi_scan(void *arg)
{
        while(1)
        {
                u_printf("ssid,auth,encry,channel,rssi,mac\r\n\r\n");
                hfwifi_scan(hfwifi_scan_test);
                u_printf("\r\n\r\n**********\r\n\r\n");
                msleep(30000);
        }
}

int USER_FUNC app_main (void)
{
        time_t now=time(NULL);

        HF_Debug(DEBUG_LEVEL,"sdk version(%s),the app_main start time is %d %s\n",\
                                                hfsys_get_sdk_version(),now,ctime(&now));
        if(hfgpio_fmap_check()!=0)
        {
                while(1)
                {
                        HF_Debug(DEBUG_ERROR,"gpio map file error\n");
                        msleep(1000);
```

```
                }
                return 0;
        }

        show_reset_reason();
        if( hfsys_register_system_event( (hfsys_event_callback_t)hfsys_event_callback) != HF_SUCCESS)
        {
                u_printf("register system event fail\n");
        }

        while(!hfnet_wifi_is_active())
        {
                msleep(50);
        }
        if(hfnet_start_assis(ASSIS_PORT)!=HF_SUCCESS)
        {
                HF_Debug(DEBUG_WARN,"start httpd fail\n");
        }

        if(hfnet_start_uart(HFTHREAD_PRIORITIES_LOW,(hfnet_callback_t)uart_recv_callback)!=HF_SUCCESS)
        {
                HF_Debug(DEBUG_WARN,"start uart fail!\n");
        }
        if(hfnet_start_socketa(HFTHREAD_PRIORITIES_LOW,(hfnet_callback_t)socketa_recv_callback)!=HF_SUCCESS)
        {
                HF_Debug(DEBUG_WARN,"start socketa fail\n");
        }
        if(hfnet_start_socketb(HFTHREAD_PRIORITIES_LOW,(hfnet_callback_t)socketb_recv_callback)!=HF_SUCCESS)
        {
                HF_Debug(DEBUG_WARN,"start socketb fail\n");
        }
        if(hfnet_start_httpd(HFTHREAD_PRIORITIES_MID)!=HF_SUCCESS)
        {
                HF_Debug(DEBUG_WARN,"start httpd fail\n");
        }

        hfthread_create(wifi_scan, "scan thread", 256, NULL, HFTHREAD_PRIORITIES_LOW, NULL,NULL);

        return 1;
}

#endif
```

WebSide : http://gb.hi-flying.com

...hfgpio adc enable

---

int HSF_API hfgpio_adc_enable(int fid);

Definition:
enable the ADC function of related PIN .

Parameter:
fid: the configure function code can be system fixed ,refer to HF_GPIO_FUNC_E,or can be user defined;

Feedback value:
HF_SUCCESS: set succeed, HF_E_INVAL: fid illegal or its related PIN illegal,
HF_FAIL: set failed ;HF_E_ACCES: related PIN don't have F_ADC attributes,can not configured as ADC mode;

Remark:
The ADC of LPB100 is 12 digital

Example:

None

Marcus 2014 ShangHai

Demand: The header file：hfgpio.h
The library：libKernel.a
HSF version demand：V1.17以上
Hardware：LPBXX

...hfgpio adc get value

---

int HSF_API hfgpio_adc_get_value(int fid);

Definition：
Get sampling value of the function code related PIN .

Parameter：
fid: the configured function code can be system fixed, refer to HF_GPIO_FUNC_E, or can be user defined;

Feedback value：
HF_SUCCESS:set succeed, HF_E_INVAL: fid illegal or its related PIN illegal,
HF_FAIL:set faled ; HF_E_ACCES: related PIN don't have F_ADC attribute,can not configure to ADC mode;

Remark：
LPB100' ADC is 12 digital, user should call hfgpio_adc_enable to open ADC before call this function,sampling value is correspondent to 3.3V.

Example：
None

Marcus 2014 ShangHai

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

...hfgpio configure fpin

---

int hfgpio_configure_fpin(int fid,int flags);

Definition：
as per fid(function code) configure related PIN

Paramter：fid function code

```
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 their own function code, start from HFGPIO_F_USER_DEFINE
flags:PIN attribute, can be one or multiple value as below to run ”|” calculation

HFPIO_DEFAULT	default
HFPIO_PULLUP	inside pullup
HFPIO_PULLDOWN	inside pulldown
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

Feedback value:
HF_SUCCESS: set succeed, HF_E_INVAL: fid illegal or its related PIN illegal, HF_E_ACCES: related PIN do not have the setting attribute(flags), e.g HFGPIO_F_JTAG_TCK related PIN is a peripheral PIN, not GPIO, can't configure any other attribute except HFPIO_DEFAULT .

Remark:
Before setting, user need to figure out each PIN's attribute, please refer to related user manual;if configure the PIN with attribute it doesn't have, will feedback HF_E_ACCES error.

Example:

Marcus 2014 ShangHai

Demand: The header file: hfgpio.h
The library: libKernel.a
HSF version demand: V1.0以上
Hardware: LPBXX

...hfgpio configure fpin interrupt

---

int hfgpio_configure_fpin_interrupt(
int fid,
uint32_t flags,

hfgpio_interrupt_func handle,
int enable);

Definition:
configure the function code related PIN as interrupt input, and appoint interrupt entrance function and interrupt trigger mode

Paramter:
fid: configured function code, system fixed function code refer to HF_GPIO_FUNC_E, or can be user defined;

flags: set interrupt trigger mode, 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 set interrupt mode,flags can logic or operate other value, please refer to flags in hfgpio_configure_fpin;

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

enable:enable interrupt
1,after configuration, enbale interrupt
0,after configuration, disable interrupt, the interrupt will effect unless call hfgpio_fenable_interrupt(fid) ;

Feedback value:
HF_SUCCESS: set succeed, HF_E_INVAL: fid illegal or its related PIN illegal,
HF_FAIL: set failed; HF_E_ACCES: related PIN can not set as interrput PIN;

Remark:

Example:
refr to example/gpio test

Marcus 2014 ShangHai

Demand: The header file: hfgpio.h
The library: libKernel.a
HSF version demand: V1.0以上
Hardware: LPBXX

...hfgpio fconfigure get

---

int HSF_API hfgpio_fconfigure_get(int fid);

Definition:
Get the attribute value of PIN related to function code;

Parameter:
fid: function code, refer to HF_GPIO_FUNC_E, or can be user defined 。

Feedback value:
If succeed, feedback PIN attribute value, refer to hfgpio_configure_fpin, HF_E_INVAL: fid illegal or its related PIN illegal

Remark:
None

Example:
None

Marcus 2014 ShangHai

Demand: The header file: hfgpio.h
The library: libKernel.a
HSF version demand: V1.16以上
Hardware: LPBXX

...hfgpio fdisable interrupt

---

int hfgpio_fdisable_interrupt(int fid);

Definition:
disable function code related PIN interrupt

Paramter:
fid:configured function code, system fixed function code refer to HF_GPIO_FUNC_E, or can be user defined;

Feedback value:
HF_SUCCESS: set secceed, HF_E_INVAL: fid illegal or its related PIN illegal,
HF_FAIL:set failed; HF_E_ACCES: related PIN can not be interrupt PIN;

Remark:
before call the function, must call hfgpio_configure_fpin_interrupt to configure interrupt

Example:

Marcus 2014 ShangHai

Demand: The header file: hfgpio.h
The library: libKernel.a
HSF version demand: V1.0以上
Hardware: LPBXX

...hfgpio fenable interrupt

---

int hfgpio_fenable_interrupt(int fid);

Definition:
Enable function code related PIN interrupt

Paramter:
fid:function code, system fixed function code refer to HF_GPIO_FUNC_E, or can be user defined;

Feedback value:
HF_SUCCESS: set succeed, HF_E_INVAL: fid illegal or its related PIN illegal,
HF_FAIL:set failed ; HF_E_ACCES:related PIN can not be interrupt PIN;

Remark:
Before call this function ,must call hfgpio_configure_fpin_interrupt to configure interrupt

Marcus 2014 ShangHai

Demand: The header file: hfgpio.h
The library: libKernel.a
HSF version demand: V1.0以上
Hardware: LPBXX

...hfgpio fpin add feature

---

int HSF_API hfgpio_fpin_add_feature(int fid,int flags);

Definition:
add attribute value to function code related PIN;

Parameter:
fid: function code, refer to HF_GPIO_FUNC_E, or can be user defined ;
flags:refer to hfgpio_configure_fpin flags;

Feedback value:
HF_SUCCESS:set succeed, HF_E_INVAL: fid illegal or its related PIN illegal

Remark:
None

Example:
None

Marcus 2014 ShangHai

Demand: The header file: hfgpio.h
The library: libKernel.a
HSF version demand: V1.16以上
Hardware: LPBXX

...hfgpio fpin clear feature

---

int HSF_API hfgpio_fpin_clear_feature (int fid,int flags);

Definition:
clear one or multiple attribute value of function code related PIN;

Parameter:
fid: function code, refer to HF_GPIO_FUNC_E, or can be user defined;
flags: refer to hfgpio_configure_fpin flags;

Feedback value:
HF_SUCCESS:set succeed, HF_E_INVAL: fid illegal or its related PIN illegal

Remark:
None

Example:
None

Marcus 2014 ShangHai

Demand: The header file: hfgpio.h
The library: libKernel.a
HSF version demand: V1.16以上
Hardware: LPBXX

...hfgpio fpin is high

---

int hfgpio_fpin_is_high(int fid);

Definition:
Judge if the function code related PIN is high level;

Parameter:
fid: function code, refer to HF_GPIO_FUNC_E, or can be user defined ,fid related PIN have PIN F_GPO or F_GPI attribute.

Feedback value:
If the PIN is low level ,feedback 0, if high level, feedback 1;if ＜0,means fid related PIN ilegal.

Remark:

Example:
refer to example/gpio test

Marcus 2014 ShangHai

Demand: The header file: hfgpio.h
The library: libKernel.a
HSF version demand: V1.0以上
Hardware: LPBXX

...hfgpio fset out high

---

int hfgpio_fset_out_high(int fid);

Definition:
set the function code related PIN as high level output

Parameter:
fid: refer to HF_GPIO_FUNC_E, or can be user defined .

Feedback value:
HF_SUCCESS:set succeed, HF_E_INVAL: fid illegal or its related PIN illegal,
HF_FAIL:set failed; HF_E_ACCES: related PIN can not be output PIN;

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

Example:
refer to example/gpio test

Marcus 2014 ShangHai

Demand: The header file: hfgpio.h
The library: libKernel.a
HSF version demand: V1.0以上
Hardware: LPBXX

...hfgpio fset out low

---

int hfgpio_fset_out_low(int fid);

Definition:
set function code related PIN as low level output;

Parameter:
fid: fid: function code, refer to HF_GPIO_FUNC_E, or can be user defined.

Feedback value:
HF_SUCCESS:set succeed, HF_E_INVAL: fid illegal or its related PIN illegal

Remark:
This fucntion is equal to hfgpio_configure_fpin (fid, HFM_IO_OUTPUT_0| HFPIO_DEFAULT);

Example:
refer to example/gpio test

Marcus 2014 ShangHai

Demand: The header file: hfgpio.h
The library: libKernel.a
HSF version demand: V1.0以上
Hardware: LPBXX

...hfgpio pwm disable

---

int HSF_API hfgpio_pwm_disable(int fid);

Definition:
Disable the PWM function of function code related PIN.

Parameter:
fid:configured function code, system fixed function code refer to HF_GPIO_FUNC_E, or can be user defined.

Feedback value:
HF_SUCCESS:set succeed, HF_E_INVAL: fid illegal or its related PIN illegal.

HF_FAIL:set failed; HF_E_ACCES: related PIN don't have F_PWM attribute,can not confirgure to PWM mode;

Remark:
the frequency of LPBxx is divided from 12MHZ.

Example:
None

Marcus 2014 ShangHai

Demand: The header file: hfgpio.h
The library: libKernel.a
HSF version demand: V1.17以上
Hardware: LPBXX

...hfgpio pwm enable

---

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

Definition:
enbale PWM function of function code related PIN.

Parameter:
fid:configured function code, system fixed function code refer to HF_GPIO_FUNC_E, or can be user defined;
freq: frequency of PWM, the frequency of LPB is divided from 12MHZ..
hrate:the rate of high level in PWM, can be (1-99);

Feedback value:
HF_SUCCESS: set succeed, HF_E_INVAL: fid illegal or its related PIN illegal,
HF_FAIL:set failed; HF_E_ACCES:related PIN don't have F_PWM attribute,can not confirgure to PWM mode;

Remark:
the frequency of LPB is divided from 12MHZ,functione code related PIN have F_PWM attribute.

Example:
None

Marcus 2014 ShangHai

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

...libc 函数

HSF is compatible with standard Lib C function, such as memory management, character string, time, standard output/input etc, the description of function please refer to Lib C.

Marcus 2014 ShangHai

Remark: in Keil MDK system, user can not directly call the memory management function in Lib C, or the connection will be failed, now only three functions available in memory management, refer to hfmem_malloc,hfmem_free,hfmem_realloc.

...hfmsgq_create

HFMSGQ_HANDLE hfmsgq_create(uint32_t size,uint32_t item_size);

Definition:
Create message queue

Parameter:
size:the length of message queue
item_size:the length of message member

Feedback value:
if succeed, feedback a object pointer point to a message queue; it failed feedback NULL ;

Remark:
None

Example:
参考example/thread test

Marcus 2014 ShangHai

Demand: The header file: hfmsgq.h
The library: libKernel.a
HSF version demand: V1.8 above
Hardware: LPBXX

...hfmsgq_destory

Void hfmsgq_destroy(HFMSGQ_HANDLE msgq);

Definition:
destroy created message queue

Parameter:
msgq:created by hfmsgq_create

Feedback value:
None
Remark:
None

Example:
refer to example/thread test

Marcus 2014 ShangHai

Demand: The header file: hfmsgq.h
The library: libKernel.a
HSF version demand: V1.8以上
Hardware: LPBXX

...hfmsgq_create

---

int hfmsgq_recv(HFMSGQ_HANDLE msgq, void * msgq, uint32_t timeouts, uint32_t flags);

Definition:
read message from message queue

Parameter:
msgq: created by hfmsgq_create
timeouts: timeout period
flags: flag bit, reserved

Feedback value:
HF_SUCCESS: succeed, or failed, please refer to HSF error code;
Remark:
None

Example:
refer to example/thread test

Marcus 2014 ShangHai

Demand: The header file: hfmsgq.h
The library: libKernel.a
HSF version demand: V1.8 above
Hardware: LPBXX

...hfmsgq_send

---

int hfmsgq_send(HFMSGQ_HANDLE msgq, void *msgq, uint32_t timeouts, uint32_t flags);

Definition:
send message to message queue

Parameter:
msgq: created by hfmsgq_create
timeouts: timeout period
flags: flag bit, reserved

Feedback value:
HF_SUCCESS: succeed or failed, please refer to HSF error code;

Remark:
None

Example:
refer to example/thread test

Marcus 2014 ShangHai

Demand: The header file: hfmsgq.h
The library: libKernel.a
HSF version demand: V1.8 above
Hardware: LPBXX

...hfsmtlk stop

---

int HSF_API hfsmtlk_stop(void);

Defintion:
stop smartlink.

Parameter:
None

Feedback value:
if succeed, feedback HF_SUCCESS,otherwise failed

Remark:
after call this function, software will soft restart immediately.

Example:
refer to example/wifi test

Marcus 2014 ShangHai

Demand: The header file: hfsmtlk.h
The library: libKernel.a
HSF version demand: V1.17 above
Hardware: LPBXX

...hfwifi scan

---

int HSF_API hfwifi_scan(hfwifi_scan_callback_t p_callback);

Definition:
scan exsited AP nearby.

Parameter:
hfwifi_scan_callback_t: when device find a AP, advise the AP detail information to user via this callback.

```
typedef int (*hfwifi_scan_callback_t)
( PWIFI_SCAN_RESULT_ITEM );
typedef struct _WIFI_SCAN_RESULT_ITEM
{
   uint8_t auth; //authentication method
   uint8_t encry;//encryption method
   uint8_t channel;//work channel
   uint8_t rssi;//signal strength
   char ssid[32+1];//AP SSID
   char mac[6];//AP mac address
}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
```

Feedback value:
if succeed, feedback the number of found AP, if less than 0 means failed;function feedback means the scan is over

Remark:
None

Example:
refer to example/wifi test

Marcus 2014 ShangHai

Demand: The header file: hfwifi.h
The library: libKernel.a
HSF version demand: V1.17 above
Hardware: LPBXX

...hfuart close

---

hfuart_handle_t HSF_API hfuart_close(int uart_no);

Definition:

Parameter:
uart_no: serial number, currently can only be 0,1;

Feedback value:
if succeed, feedback HF_SUCCESS,otherwise HF_FAIL;

Remark:
when serial port is not in use, call hfuart_close to release resource;

Example:
refer to example/uart test

Marcus 2014 ShangHai

Demand: The header file: hfuart.h
The library: libKernel.a
HSF version demand: V1.0 above
Hardware: LPBXX

...hfuart open

---

hfuart_handle_t HSF_API hfuart_open(int uart_no);

Definition:

open serial device

Parameter:
uart_no:serial number, now can only be 0,1;

Feedback value:
if succeed,feedback a pointer point to serial device; otherwise feedback NULL

Remark:
serial API hfuart_open and hfuart_recv must be in the same thread, or hfuart_recv will not receive data, before use uart, must call hfuart_open;

Example:
refer to example/uart test

Marcus 2014 ShangHai

Demand: The header file: hfuart.h
The library: libKernel.a
HSF version demand: V1.0 above
Hardware: LPBXX

...hfuart send

---

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

Definition:
Send data to serial

Parameter:
huart:serial device object, feedback from hfuart_open
data: buffer area for sending data
bytes:the length of sending data
timeouts:timeout period

Feedback value:
if succeed, feedback the actual sending data;if failed, feedback error code.

Remark:

Example:
refer to example/uart test

Marcus 2014 ShangHai

Demand: The header file: hfuart.h
The library: libKernel.a
HSF version demand: V1.0 above
Hardware: LPBXX

...hfuart recv

---

int HSF_API hfuart_recv(

hfuart_handle_t huart,char *recv,
uint32_t bytes,
uint32_t timeouts)

Definition:
receive data from serial

Parameter:
huart:serial device object,feedback from hfuart_open
recv:buffer area for reserve received data;
bytes:the length of buffer area
timeouts:timeout period, when use "select" to operate, timeouts must be 0;

Feedback value:
if succeed, feedback the actual length of received data, or feedback error code;

Remark:
if used system owned serial transparent transmit and command mode, please dont call this function, otherwise would cause serial transparent transmit and command mode abnormal; or call hfnet_start_uart to get the serial data.

Example:
refer to example/uart test

Marcus 2014 ShangHai

Demand: The header file: hfuart.h
The library: libKernel.a
HSF version demand: V1.0 above
Hardware: LPBXX

...hftimer_start

---

int HSF_API hftimer_start(hftimer_handle_t htimer);

Definition:
start a timer

Parameter:
htimer:created by hftimer_create;

Feedback value:
if succeed feedback HF_SUCCESS,otherwise HF_FAIL;

Remark:

Example:
refer to example/timer test

Marcus 2014 ShangHai

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

...hftimer_create

```
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 );
```

Definition:
create a timer

Parameter:
name: the name of timer.

period: the period of trigger the timer, the unit is in ms;
If flags set as HFTIMER_FLAG_HARDWARE_TIMER, the unit is in μs.

auto_reload: appoint as automatically or manually. if true, only need to call hftimer_start one time, after timer triggerred, no need to call hftimer_start again;if false,then user need to call hftimer_start again after trigger.

timer_id: appoint a unique ID represent the timer, when multiple timers use the same callback fuction, the unique ID can be used to distinguish the timers;

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

Feedback value:
if succeed, feedback a pointer point to a timer object, otherwise feedback NULL;

Remark:
after timer created, it will not start immediately untill call hftimer_start. If manually,re-trigger the timer need to call hftimer_start, if automatically, then no need, timer will automatically trigger in next specific period.
If create a hardware timer, set flags as HFTIMER_FLAG_HARDWARE_TIMER,only one hardware timer can be created. The period unit of hardware timer is ms, and it is not accuracy, need adjustment (about %1-%2 allowance) .Only V1.17 and later version able to support hardware timer .

Example:
refer to example/timer test

Marcus 2014 ShangHai

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

...hftimer_change_period

void HSF_API hftimer_change_period(
hftimer_handle_t htimer,
int32_t new_period
);

Definition:
revise the timer period

Parameter:
htimer:created by hftimer_create;
new_period: new period, unit is ms. if it is hardware, then unit is μs

Feedback value:
None;

Remark:
revise the timer period, after call the function, timer will run in new period.

Example:
refer to example/timer test

Marcus 2014 ShangHai

Demand: The header file: hftimer.h
The library: libKernel.a
HSF version demand: V1.17 above
Hardware: LPBXX

...hftimer_delete

---

void HSF_API hftimer_delete(hftimer_handle_t htimer);

Definition:
delete a timer

Parameter:
htimer: the deleting timer, created by hftimer_create;

Feedback value:
None

Example:
refer to example/timer test

Marcus 2014 ShangHai

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

...hftimer get counter

---

void HSF_API hftimer_get_counter (hftimer_handle_t htimer);

Definition:
get the CLK counter which is time spending from start to now of the hardware timer

Parameter:
htimer: point to the hardware timer created by hftimer_create .

Feedback:
feedback the the counter value which is the spend time from start to now of the harfware timer, the frequency of LPB100 is 48MHZ,one CLK is 1/48 us,from start to now,time consuming of timer is counter/48 us; if feedback value is 0, means the timer is time out.

Remark:
if program require accuracy time, can realize by this function plus hardware timer.

Example:
refer to example/timer test

Marcus 2014 ShangHai

Demand: The header file: hftimer.h
The library: libKernel.a
HSF version demand: V1.17 above
Hardware: LPBXX

...hftimer_get_timer_id

---

uint32_t HSF_API hftimer_get_timer_id
(hftimer_handle_t htimer);

Definition:
get the timer ID

Parameter:
htimer:create by hftimer_create;

Feedback value:
if succeed, feedback the timer ID, appoint by hftimer_create; if failed, feedback HF_FAIL;

Remark:
this function is usually used when timer callback, to distinguish multiple timers when they use the same callback function or when timer callback

Example:
refer to example/timer test

Marcus 2014 ShangHai

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

...hftimer_stop

---

void HSF_API hftimer_stop(hftimer_handle_t htimer);

Definition:
stop a timer

Parameter:
htimer:create by hftimer_create;

Feedback value:
None;

Remark:
after call the function, the timer will stop trigger, until call hftimer_start again;

Example:
refer to example/timer test

Marcus 2014 ShangHai

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

...定时器

LPB software timer accuracy is 1 ms, LPB100 software timer accuracy is 10 ms, if more accuracy timer is required, please use hardware timer. the accuracy of hardware is us. user can not operate long time delay in the timer, can not call the API which used the timer, or the timer will shutdown.

Marcus 2014 ShangHai

Demand: The header file: hftimer.h
The library: libKernel.a
Hardware: LPBXX

...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);

Definition:
create a thread

Parameter:
routine : input parameter:the entrance function of the thread,
typedef void (*PHFTHREAD_START_ROUTINE)(void

*);

stack_depth:input parameter: stack depth, 4Bytes as an unit, stack_size = stack_depth*4;

parameters: input parameter,pass to thread entrance function;

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

HFTHREAD_PRIORITIES_LOW	low priority
HFTHREAD_PRIORITIES_MID	middle priority
HFTHREAD_PRIORITIES_NORMAL	normal priority
HFTHREAD_PRIORITIES_HIGH	high priority

user thread usually use HFTHREAD_PRIORITIES_MID, HFTHREAD_PRIORITIES_LOW;

created_thread: optional, if succeedm feedback a pointer point to create thread, if null, no feedback;

stack_buffer: reserve for future use

feedback value:
HF_SUCCESS: succeed ,otherwise failed, , refer to HSF error code;

Remark:
for stability, suggest use to use HFTHREAD_PRIORITIES_LOW and HFTHREAD_PRIORITIES_MID , HFTHREAD_PRIORITIES_NORMAL or above is not suggested, unless most of the thread is under sleep and few things to proceed。

Example:
refer to hfthread_create

Marcus 2014 ShangHai

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

...hfthread_delay

---

void hf_thread_delay(uint32_t ms);

Definition:
suspend current thread, the unit is ms

Parameter:
ms ,the suspend time (the unit is ms);

feedback value:
this function has no feedback value

Remark:
there maybe allowance between the thread sleep time which the function made and the actual time, if accuracy time is required for sleep, please use hfthread_delay (hftimer_get_timer_adjust_factor()*ms), msleep function has this limit, too.

Marcus 2014 ShangHai

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

...hfthread destory

---

void hfthread_destroy(hfthread_hande_t thread);

Definition:
delete the thread created by hfthread_create;

Parameter:
thread: point to the delete thread, if null, delete currect thread.

Feedback value:
the function has no feedback value

Marcus 2014 ShangHai

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

...hfthread disable softwatchdog

---

int HSF_API hfthread_disable_softwatchdog(
hfthread_hande_t thread,
);

Definition:
disable the thread software watchdog.

Parameter:
thread: a pointer point to thread, feedback from hfthread_create, the parameter can be NULL,when feedback NULL,disable the software watchdog of current thread;

feedback value:
HF_SUCCESS: succeed, otherwise failed, refer to HSF error code;

Remark:
in the thread operation process, if one operation tooks too long time (or waitng too long time for a signal) and the time is longer than overtime, user can disable the software watchdog, in case the watchdog effect since the time is yoo long and cause restart; after the operation finished, enable the watchdog.

Example:
refer to example/thread test

Marcus 2014 ShangHai

Demand: The header file: hfthread.h
The library: libKernel.a
HSF version demand: V1.7 above
Hardware: LPBXX

...hfthread enable softwatchdog

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

Definition:
enable the thread software watchdog.

Parameter:
thread: pointer point to thread, feedback from hfthread_create, the parameter can be NULL,when feedback NULL,enable the software watchdog of current thread;
time:software watchdog overtime, unit is second;

feedback value:
HF_SUCCESS: succeed, otherwise failed, refer to HSF error code;

remark:
thread watchdog can be used to check if the thread is shut down. if enable the watchdog, thread did not call hfthread_reset_softwatchdog in stipulated time, LPB module will soft reset. This function can be called many times, can dynamically revise overtime. System will restore the watchdog when call the function.
Defaultly the watchdog is disabled, it will effect after call the function.

Example:
refer to example/thread test

Marcus 2014 ShangHai

Demand: The header file: hfthread.h
The library: libKernel.a
HSF version demand: V1.7 above
Hardware: LPBXX

...hfthread mutex free

---

void hfthread_mutext_free(hfthread_mutex_t mutex);

Definition:
delete the thread created by hfthread_mutext_new;

parameter:
mutex:point to the deleting mutex;

feedback value:
the function has no feedback value;

Example:
refer to hfthread mutex new

Marcus 2014 ShangHai

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

...hfthread mutex lock

---

int hfthread_mutext_lock (hfthread_mutex_t mutex);

Definition:

parameter:
mutex: point to a mutex object created by hfthread_mutext_new;

feedback value:
HF_SUCCESS succeed; HF_FAIL probably cause locked, refer to HSF error code

Remark:
hfthread_mutext_lock and hfthread_mutex_unlock comes out in pairs., if user just call hfthread_mutex_lock, but not call hfthread_mutex_unlock, then when user call hfthread_mutex_lock again, it would locked;

Example:

Marcus 2014 ShangHai

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

...hfthread mutex new

---

int HSF_API hfthread_mutext_new(hfthread_mutex_t *mutex)

Definition:
create a thread mutex ;

Parameter:
mutex: function executed succeedly, point to the created mutex;

feedback value
HF_SUCCESS:succeed, otherwise failed, refer to HSF error code;

remark:
when user no longer use the created mutex, please call hfthread_mutext_free to release resource;

example:

Marcus 2014 ShangHai

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

...hfthread mutex trylock

---

int HSF_API hfthread_mutext_trylock (hfthread_mutex_t mutex)

Definition:
check if mutext is locked.

Parameter:
mutex: point to a mutex object, created by hfthread_mutext_new;

feedback value:
if mutext lock, feedback 0; otherwise mutex is not locked.

remark:

example:

Marcus 2014 ShangHai

Demand: The header file: hfthread.h
The library: libKernel.a
HSF version demand: V1.17 above
Hardware: LPBXX

...hfthread mutex unlock

---

void hfthread_mutext_unlock(hfthread_mutex_t mutex);

Definition:
release the mutex;

parameter:
mutex: point to a mutex object, created by hfthread_mutext_new;

feedback value:
the function has no feedback value;

example:
refer to hfthread mutex new

Marcus 2014 ShangHai

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

...hfthread reset softwatchdog

---

int HSF_API hfthread_disable_softwatchdog(
hfthread_hande_t thread,
);

Definition:
restore software watchdog of the thread (feed the dog)。

parameter:
thread: pointer point to thread, feedback from hfthread_create, the parameter can be NULL, if NULL,restore the software watchdog of current thread;

feedback value:
HF_SUCCESS: succeed, otherwise failed, refer to HSF error code;

Remark:
after enable the watchdog, thread must call the function in the stipulated time and feed the dog; when the watchdog is overtime, module will soft reset.

example:
refer to example/thread test

Marcus 2014 ShangHai

Demand: The header file: hfthread.h
The library: libKernel.a
HSF version demand: V1.7 above
Hardware: LPBXX

...system error code definition

---

system error code definition
API function feedback value (special specification excluded) stipulation: , succeed HF_SUCCESS, >0, failed <0. the size of error code is 4Bytes, with symbol and integer, feedback value is the negative number of error code. 31-24bit is module index, 23-8 for reservation, 7-0 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*/
};
```

Marcus 2014 ShangHai

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

```
...hfhttpd url callback cancel
```

---

int HSF_API hfhttpd_url_callback_cancel(void);

Definition:
cancel url callback function

Parameter:
None

Feedback value:
if succeed, feedback HF_SUCCESS, HF_FAIL means failed;

Remark:
None

Example:
None

Marcus 2014 ShangHai

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

...hfhttpd url callback register

---

int HSF_API hfhttpd_url_callback_register(
hfhttpd_url_callback_t callback,
int flag
);

Definition:
set the data process callback based on webserver url.

Parameter:
callback: callback function for user parameter analysis;
set callback function type as:
int hfhttpd_url_callback (char *url, char *rsp);
url is the url removed ip address; rsp is the required feedback data, support utmost 1400 byte; if callback function feedback -1, then webserver will handle; if feedback 0, webserver will not analysiss this http request, callback function will handle this request by its own.
flag: 0 no authentication is required ; 1 require authentication;

Feedback value:
if succeed feedback HF_SUCCESS, HF_FAIL means failed;

Remark:
after first step analysis, the url has removed ip address; if brower input : 10.10.100.254/abcd, the url callback function got is /abcd。

Example:
refer to example/urlcallback test

Marcus 2014 ShangHai

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

...hfnet httpd set get nvram callback

---

void HSF_API hfnet_httpd_set_get_nvram_callback(
hfhttpd_nvset_callback_t p_set,
hfhttpd_nvget_callback_t p_get);

Definition:
set the webserver to get module parameter callback.

Parameter:
p_set: optional parameter, if no need to extend

WEB to get parameter interface, please set a null, otherwise point to the entrance function setting;
the callback function type has: :

int hfhttpd_nvset_callback(char * cfg_name,int name_len,char* value,int val_len);
cfg_name: the name of the correspondent configuration ;name_len : the length of cfg_name;value : the correspondent configuration value;val_len :the length of value;

p_get: optional parameter, if no need to extend WRB to get parameter interface, please set a null, otherwise point to the entrance function of get parameter;
read the callback function type of parameter :

int hfhttpd_nvget_callback(char *cfg_name,int name_len,char *value,int val_len);
cfg_name: the name of reading parameter, please be aware that cfg_name do not contain character string ending character; name_len: the length of cfg_name ;value: the configuration value of cfg_name; val_len: the length of value.

Feedback value:
None

Remark:

Example:
refer to example/file test

Marcus 2014 ShangHai

Demand: The header file: hfnet.h
The library: libKernel.a
HSF version demand: V1.15 above
Hardware: LPBXX

...hfnet ping

---

int hfnet_ping(const char* ip_address);

Definition:
send PING packet to target address, check if the IP address is reachable.

Parameter:
ip_address: the character string of the checking IP address, address format is xxx.xxx.xxx.xxx, if need to ping a domain , please call hfnet_gethostbyname to get domain IP address;

Feedback value:
if succeed feedback HF_SUCCESS, otherwise failed, refer to HSF error code

Remark:
if network disconnect, DNS server setting error would cause failure.

Example:

Marcus 2014 ShangHai

Demand: The header file: hfnet.h
The library: libKernel.a
HSF version demand: V1.0 above
Hardware: LPBXX

...hfnet set udp broadcast port valid

---

int HSF_API hfnet_set_udp_broadcast_port_valid (
uint16_t start_port,
uint16_t end_port)

Definition:
set the broadcast port range which UDP can receive ;

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

Feedback value
if succeed, feedback HF_SUCCESS,otherwise feedback -HF_E_INVAL;

Remark:
by defaultly, LPB100 will filter brroadcast packet in the network to unburden the system, so if the socket created by user need to receive broadcast packet, user should set the monitor port area through the function.

Example:
refer to example/thread test

Marcus 2014 ShangHai

Demand: The header file: hfnet.h
The library: libKernel.a
HSF version demand: V1.17 above
Hardware: LPBXX

...hfnet socketa fd

---

int HSF_API hfnet_socketa_fd(void);

Definition:
get the standard socket descriptor of socketa;

Parameter:
None

Feedback value:
if succeed, feedback the standard descriptor of socketa;otherwise feedback value < 0.

Remark:
if socketa work under server mode, feedback the monitor socket fd.

Example:
refer to example/netcallback test

Marcus 2014 ShangHai

Demand: The header file: hfnet.h
The library: libKernel.a
HSF version demand: V1.17 above
Hardware: LPBXX

...hfnet socketa get client

---

int HSF_API hfnet_socketa_get_client(int cid,phfnet_socketa_client_t p_client);

Definition:
get the connected clients information when socketa work under TCP server mode;

Parameter:
cid: clients ID, can be 0-4, currently socketa can connect utmost 5 clients;
p_client:can't be NULL,point to clients information structure;

Feedback value:
if succeed, feedback HF_SUCCESS,clients information is reserved in p_client; otherwise failed, if cid corresponding client server dones not exsit, feedback failure. .

Remark:
this function is only effect when socketa work under TCP srerver mode. cid feedback from socketa event callback.

Example:
refer to example/netcallback test

Marcus 2014 ShangHai

Demand: The header file: hfnet.h
The library: libKernel.a
HSF version demand: V1.17 above
Hardware: LPBXX

...hfnet socketa send

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

Definition:
send data to SOCKETA

Parameter:
data:the buffer area for reserve the sending data;
len:the length of buffer area;
timeouts:send overtime, unavailable currently;

Feedback value:
if succeed, feeback the length of actual sending data,otherwise feedback error code;

Remark:

Example:
None

Marcus 2014 ShangHai

Demand: The header file: hfnet.h
The library: libKernel.a
HSF version demand: V1.03 above
Hardware: LPBXX

...hfnet socket fd

---

int HSF_API hfnet_socketb_fd(void);

Definition:
get socketb standard socket descriptor ;

Parameter:
None

Feedback value:
if succeed, feedback standard socket discriptor of socketb, otherwise feedback < 0.

Remark:
user can get standard socket discriptor through this interface, operate socketb via standard lwip function.

Example:
refer to example/netcallback test

Marcus 2014 ShangHai

Demand: The header file: hfnet.h
The library: libKernel.a
HSF version demand: V1.17 above
Hardware: LPBXX

...hfnet socketb send

---

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

Definition:
send data to SOCKETB

Parameter:
data:buffer area for reserve the sending data;
len: the length of buffer area;
timeouts:send overtime, unavailable currently;

Feedback value:
if succeed, feedback the length of actual sending data, otherwise feedback error code;

Remark:

Example:
None

Marcus 2014 ShangHai

Demand: The header file: hfnet.h
The library: libKernel.a
HSF version demand: V1.03 above
Hardware: LPBXX

...hfnet start httpd

---

int hfnet_start_httpd(uint32_t uxpriority);

Definition:
start httpd,a small-sized web server

Parameter:
uxpriority: httpd service priority, please refer to hfthread_create uxpriority parameter;

Feedback value:
if succeed feedback HF_SUCCESS, HF_FAIL means failed

Remark:
if application require to support webpage interface,please call the function when program start;

Example:

Marcus 2014 ShangHai

Demand: The header file: hfnet.h
The library: libKernel.a
HSF version demand: V1.0 above
Hardware: LPBXX

...hfnet start socketa

---

int hfnet_start_socketa(uint32_t uxpriority,hfnet_callback_t p_callback);

Definition :
start HSF own socketa service

Parameter:
uxpriority: socketa service priority, refer to hfthread_create uxpriority parameter;

p_callback: callback function, optional, if callback is no need, set the value as NULL; it can be triggerred when socketa receive data packet or status variation;
int socketa_recv_callback_t(uint32_t event,void *data,uint32_t len,uint32_t buf_len);
event:event ID ;
data: point to the buffer for receiving data, user can revise the buffer value in callback function; when work under UDP mode,the 6 bytes

follow data+len is the 4 bytes Ip address and 2 bytes port number of sending port. if socketa work under TCP server mode, the 4 bytes follow data+len is the cid of client server, user can call hfnet_socketa_get_client to get detail information.
len:the length of received data;
buf_len: the actual length of data point to buffer. the value should ≥len;
the feedback value of callback funtion is the length of processed data,if user just read the data, but not revise, the value should equal to len;

feedback value:
if succeed feedback HF_SUCCESS, HF_FAIL means failed

Remark:
when socketa service received data sending from network, call p_callback, then send the p_callback processed value to uart, user can use p_callback to analysis the received data or secondary treatment, such as encryption, decipher, send the processed data back to socketa service.

Example:
the below example can add the received length to last two bytes of buffer, when socketa (under TCP server mode) service received the data sending from network ;
refer to socketa_callback

Marcus 2014 ShangHai

Demand: The header file: hfnet.h
The library: libKernel.a
HSF version demand: V1.0 above
Hardware: LPBXX

...hfnet start socketb

---

int hfnet_start_socketb(uint32_t uxpriority,hfnet_callback_t p_callback);

Definition:
start HSF own socketb service.

Parameter:
uxpriority:socketb service priority level;refer to hfthread_create uxpriority parameter
p_callback:optional, if not in use, set as NULL, please refer to hfnet_start_socketa

Feedback value:
if succeed feedback HF_SUCCESS, HF_FAIL means failed

Remark:

Example:
refer to hfnet_start_socketa

Marcus 2014 ShangHai

Demand: The header file: hfnet.h
The library: libKernel.a
HSF version demand: V1.0 above
Hardware: LPBXX

...hfnet start uart

---

int hfnet_start_uart(uint32_t uxpriority,hfnet_callback_t p_uart_callback);

Definition:
start HSF own uart service to control receive and send data.

Parameter:
uxpriority:uart service priority,please refer to hfthread_create uxpriority parameter
p_uart_callback: uart callback function, optional. if not need, please set as NULL; call the function when uart received data, the definition and parameter please refer to hfnet_start_socketb;

Feedback value:
if succeed, feeback HF_SUCCESS, HF_FAIL means failed

Remark:
when uart receive data, if p_uart_callback is not NULL,please call p_uart_callback at first.if work under through mode, send the received data to socketa, socketb service (if these two server exsited);if work under command mode, send the received data to command analytical program. under through mode, user can use this callback function and socketa,socketb service callback to realize the data enryption and decipher, or secondary treatment; under command mode, user can realize self-define the name and format of AT command through callback ;

Example:

Marcus 2014 ShangHai

Demand: The header file: hfnet.h
The library: libKernel.a
HSF version demand: V1.0 above
Hardware: LPBXX

...标准socket API

HSF use lwip protocal, compatible with standard socket interface, such as socket,recv,select,sendto,ioctl; if source code use standard socket function, user only need to inputhsf.h and hfnet.h, for use method of standard socket please refer to related unser manual.

Marcus 2014 ShangHai

Remark: since system restriction, when create socket via lwip, create socket and receive data must be in the same thread, send data is not required in the same thread. or module unable to receive data. If UPD need to receive broadcast packet, we have filter the broadcoast for better function, if need to receive broadcast, please refer to hfnet_set_udp_broadcast_port_valid.

...hfmem_free

---

void HSF_API hfmem_free(void *pv);

Definition:
release the memory allocated by hfsys_malloc

Parameter:
pv: point to the address of releasing memory;

Feedback value:
None

Remark:
the function is thread-safe, if multiple thread apply this function, don't use the "free" in libc, it is non-thread-safe function.

Example:
None

Marcus 2014 ShangHai

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

...hfmem_malloc

---

void *hfmem_malloc(size_t size)

Definition:
dynamically allocate memory

Parameter:
size:the size of allocated memory

Feedback value:
If NULL,means system has no free memory, if succeed, feedback the memory address;

Remark:
the function is thread-safe, if multiple thread apply this function,
do not use malloc in libc, it is non-thread-safe function,
in LPB100 series, call the meomry management function in libc is not successful.

Marcus 2014 ShangHai

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

...hfmem realloc

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

Definition:
reallocate memory

Parameter:
pv:point to the previous allocated address by hfmem_malloc;
size:the size of reallocated memmory

Feedback value:
None

Remark:
refer to libc realloc, user can not call realloc function directly, but only use the API。

Example:
None

Marcus 2014 ShangHai

Demand: The header file: hfsys.h
The library: libKernel.a
HSF version demand: V1.7 above
Hardware: LPBXX

...hfsys_get_reset_reason

uint32_t HSF_API hfsys_get_reset_reason (void);

Definition:
get the reason why module reset

Parameter:
None

Feedback value:
the module reset can be one or multiple reasons as below:
.

HFSYS_RESET_REASON_NORMAL	module power off cause reset
HFSYS_RESET_REASON_ERESET	hardware watchdog and reset key cause reset
HFSYS_RESET_REASON_IRESET0	program call hfsys_softreset cause reset （software watchdog reset, or program error, or memory access error）

HFSYS_RESET_REASON_IRESET1	program call hfsys_reset cause reset
HFSYS_RESET_REASON_WPS	press WPS cause reset
HFSYS_RESET_REASON_SMARTLINK_START	launch SmartLink cause reset
HFSYS_RESET_REASON_SMARTLINK_OK	SmartLink configure succeed cause reset
HFSYS_RESET_REASON_WPS_OK	WPS match succeed cause reset

Remark:
user can call this function to judge the reset reason can according to different reason to restore.

Example:
refer to example hfsys_get_reset_reason

Marcus 2014 ShangHai

Demand: The header file: hfsys.h
The library: libKernel.a
HSF version demand: V1.7 above
Hardware: LPBXX

...hfsys_get_run_mode

---

int hfsys_get_run_mode()

Definition:
get system current run mode

Parameter:
None

Feedback value:
feedback 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
};
```

Marcus 2014 ShangHai

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

...hfsys_get_time

---

uint32_t HSF_API hfsys_get_time (void);

Definition:
get the spent time from system bootup to now (ms)

Parameter:
None

Feedback value:
feedback the spent ms value from system bootup to now

Remark:
None

Example:
None

Marcus 2014 ShangHai

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

...hfsys_nvm_read

---

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

Definition:
read data from NVM

Parameter:
nvm_addr:NVM address ,can be(0-99);
buf: buffer area for reserve data read from NVM;
length: length plus nvm_addr is smaller than 100;

Feedback value:
if succeed, feedback HF_SUCCESS, otherwise feedback value < 0 .

Remark:
when module restart, soft-reset, NVM data will not be cleared, LPB provide 100Bytes NVM; if module power off, the NVM data will be cleared .

Example:
None

Marcus 2014 ShangHai

Demand: The header file: hfsys.h
The library: libKernel.a
HSF version demand: V1.7 above
Hardware: LPBXX

...hfsys_nvm_write

---

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

Definition:
write data to NVM

Parameter:
nvm_addr:NVM address, can be(0-99);
buf: buffer area for reserve data read from NVM;
length: length plus nvm_addr is smaller than 100;

Feedback value:
if succeed, feedback HF_SUCCESS, otherwise < 0

Remark:
when module restart, soft reset, NVM data will not be cleared, LPB Provide 100Bytes NVM, if module power off, the NVM data will be cleared

Example:
None

Marcus 2014 ShangHai

Demand: The header file: hfsys.h
The library: libKernel.a
HSF version demand: V1.7 above
Hardware: LPBXX

...hfsys_register_system_event

---

int HSF_API hfsys_register_system_event (hfsys_event_callback_t p_callback);

Definition:
register system event callback

Parameter:
p_callback: point to the callback function address of user defined system event;

Feedback value:
if feedback HF_SUCCESS, system will handle the event by default operation; if feedback < 0, system will not handle the event.

Remark:
in callback function, user can not call the API function with delay. Delay is not allowed, should feedback immediately after handle the event, otherwise will effect the system normal operation . currently supported system event as follow:

HFE_WIFI_STA_CONNECTED	trigger when STA connected
HFE_WIFI_STA_DISCONNECTED	trigger when STA disconnected
HFE_CONFIG_RELOAD	trigger when system execute reload
HFE_DHCP_OK	trigger when STA conneect, and DHCP get address
HFE_SMTLK_OK	trigger when SMTLK configuration get password, defaultly system will resrt; if feedback is not

	HF_SUCCESS, reset is hold, user can reset manually.

Example:
refer to example hfsys_register_sys_event

Marcus 2014 ShangHai

Demand: The header file: hfsys.h
The library: libKernel.a
HSF version demand: V1.7 above
Hardware: LPBXX

...hfsys_reload

---

void HSF_API hfsys_reload();

Definition:
restore system to factory setting

Parameter:
None

Feedback value:
None

Remark:
None

Example:
None

Marcus 2014 ShangHai

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

...hfsys_reset

---

void HSF_API hfsys_reset(void);

Definition:
restart system, IO level not hold

Parameter None

Feedback:
None

Remark:
None

Example:
None

Marcus 2014 ShangHai

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

...hfsys_softreset

---

void HSF_API hfsys_softreset(void);

Definition:
soft reset system, IO level hold

Parameter:
None

Feedback value:
None

Remark:
None

Example:
None

Marcus 2014 ShangHai

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

...hfsys_switch_run_mode

---

int hfsys_switch_run_mode(int mode);

Definition:
switch system run mode

Parameter:
mode:the switched run mode, system currently supportted mode has

```
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: through mode
HFSYS_STATE_RUN_CMD: command mode
HFSYS_STATE_RUN_GPIO:GPIO mode

Feedback value:
HF_SUCCESS: succeed, otherwise failed, refer to HSF error code;

Marcus 2014 ShangHai

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

...hfuflash erase page

---

int HSF_API hfuflash_erase_page(uint32_t addr, int pages);

Definition:
erase the user flash page

Parameter:
addr: logic address of user flash, not physical address;
pages : the erase pages;

Feedback value:
if succeed, feedback HF_SUCCESS; if failed, feedback HF_FAIL;

Remark:
user flash is a 128KB area in physical flash, user can only operate this area via API. API operation address is the logic address of user flash, user don't need to attention to its actual address.

Example:
refer to example/uflash test

Marcus 2014 ShangHai

Demand: The header file: hfflash.h
The library: libKernel.a
HSF version demand: V1.16a above
Hardware: LPBXX

...hfuflash read

---

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

Definition:
read data from user file

Parameter:
addr: logic address of user flash(0-HFUFLASH_SIZE-2);
data : read data from flash buffer area ;
len : the length of buffer area;

Feedback value:
if < 0 means failed, otherwise feedback the actual Bytes number read from flash;

Remark:
None

Example:
refer to example/uflash test

Marcus 2014 ShangHai

Demand: The header file: hfflash.h
The library: libKernel.a
HSF version demand: V1.16a above
Hardware: LPBXX

...hfuflash write

---

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

Definition:
write data to user file

Parameter:
addr: the logic address of user flash(0-HFUFLASH_SIZE-2);
data : the buffer area for reserve the data written to flash;
len : the length of buffer area;

Feedback value:
if < 0 means failed, otherwise feedback the actaul Bytes number written to flash;

Remark:
before write the flash, if the flash address is already has data, must erase the data first。
data address can not be in the ROM, must be in the RAM. Or call the function may cause shut down or the program would feedback - HF_E_INVAL,below code is forbidden:

Wrong code 1: ”Test” in ROM area;
hfuflash_write (Offset,"Test",4);

Wrong code 2: const modified, initialized variable in program area(ROM).
const uint8_t Data[] = "Test";
hfuflash_write (Offset,Offset,Data,4);
Right code:
Uint8_t Data[]=”Test”;
hfuflash_write (Offset,Offset,Data,4);

Example:
refer to example/uflash test

Marcus 2014 ShangHai

Demand: The header file: hfflash.h
The library: libKernel.a
HSF version demand: V1.16a above
Hardware: LPBXX

...hffile userbin read

---

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

Definition:

read data from user file

Parameter:
offset: file offset;
data : reserve the data read from file to buffer area;
len : length of buffer area;

Feedback value:
if < 0 means failed, otherwise feedback the actual Bytes number read from file;

Sample:
refer to example/file test

Marcus 2014 ShangHai

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

...hffile userbin size

---

int HSF_API hffile_userbin_size(void);

Parameter:
None

Feedback value:
if < 0 means failed, otherwise feedback the file size value;

Remark:
None

Example:
refer to example/file test

Marcus 2014 ShangHai

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

...hffile userbin write

---

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

Definition:
write data to user file

Parameter:
offset: file offset;
data : reserve the file data to buffer area;
len : the length of buffer area;

feedback value:
if < 0 means failed, otherwise feedback the

actual Bytes number written to the file;

Remark:
user configuration file is fixed size and reserved in flash, can be used to reserve user data. user configuration file has backup function, if accidently power off during write, it will automatically restore to previous content.

Example:
refer to example/file test

Marcus 2014 ShangHai

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

...hffile userbin zero

---

int HSF_API hffile_userbin_zero (void);

Parameter:
None

Definition:
zero clearing the whole file

feedback value:
< 0 means failed, otherwise feedback the size of the file;

Remark:
call this function can clear the all file to zero quickly, faster than via hffile_userbin_write;

Example:
refer to example/file test

Marcus 2014 ShangHai

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

...hfupdate complete

Int hfupdate_complete(
HFUPDATE_TYPE_E type,
uint32_t file_total_len
);

Definition:
upgrade completed

Parameter:
type: upgrade type
file_total_len: the length of upgrade file

Feedback value:
if succeed, feedback HF_SUCCESS, otherwise failed

Remark:
when all upgrade file downloaded, call this function to run upgrad.

Example:
refer to example/at test

Marcus 2014 ShangHai

Demand: The header file: hfupdate.h
The library: libKernel.a
HSF version demand: V1.17 above
Hardware: LPBXX

...hfupdate start

---

int hfupdate_start(HFUPDATE_TYPE_E type);

Definition:
start upgrade.

Parameter:
type: upgrade type

```
typedef enum HFUPDATE_TYPE
{
   HFUPDATE_SW=0,//upgrade software
   HFUPDATE_CONFIG=1,//upgrade default
configuration
   HFUPDATE_WIFIFW,//upgrade WIFI firmware
   HFUPDATE_WEB,//upgrade web
}HFUPDATE_TYPE_E;
```

Feedback value:
if succeed ,feedback HF_SUCCESS, otherwise failed

Remark:
currently only support HFUPDATE_SW. before download the upgrade file, call this function to initialization.

Example:
refer to example/at test

Marcus 2014 ShangHai

Demand: The header file: hfupdate.h
The library: libKernel.a
HSF version demand: V1.17 above
Hardware: LPBXX

...hfupdate write file

---

int hfupdate_write_file(
HFUPDATE_TYPE_E type ,
uint32_t offset,
char *data,

int len);

Definition:
write the upgrade data to upgrade area.

Parameter:
type: upgrade type
offset:offset of the upgrade file
data:the upgrade file data
len: the length of upgrade data

feedback value:
if > 0 means succeed, otherwise failed.

Remark:
currently only support HFUPDATE_SW.

example:
refer to example/at test

Marcus 2014 ShangHai

Demand: The header file: hfupdate.h
The library: libKernel.a
HSF version demand: V1.17 above
Hardware: LPBXX