

High-Flying Software Framework (HSF) 用户手册 V1.3x

版本 1.3x
2014 年 12 月

更新记录:

修改时间	作者	修改	删除
2013.08.28	Jim	初版	
2013.09.16	Jim	HSF-V1.03,添加 example 说明, 和 SDK 历史变更	
2013.10.17	Jim	HSF-V1.1x,添加对 LPB100 的支持	
2013.11.26	Jim	添加软件看门狗功能。 添加 PWM 接口	
2014.02.18	Jim	发布最终 V1.17 版本	
2014.03.19	Jim	发布 V1.18 版本	
2014.09.23	Jim	发布 V1.20 版本	
2014.12.05	Jim	发布 V1.30 版本	

目录

1	SDK 历史变更及注意事项	4
2	编译环境安装.....	6
2.1	基于 Keil MDK 的 SDK 环境安装(LPB100).....	6
3	资源分配.....	10
3.1	LPB100 资源分配	10
4	用户怎么样添加自己的源代码	11
4.1	用户函数定义约定.....	11
4.2	用户添加源代码文件	11
5	用户怎么样自定义 GPIO	12
5.1	PIN 脚的定义	12
5.2	系统固定功能码	13
5.3	怎么样修改映射表.....	14
5.4	怎样操作 PIN 脚.....	18
6	用户怎样添加自定义 AT 命令	19
7	怎么样通过串口打印调式信息	20
8	Keil-MDK 中用 ULINK 在线调式程序	21
9	硬件推荐连接.....	23
9.1	硬件推荐连接.....	23
10	怎样升级程序.....	24
10.1	通过串口升级.....	24
10.2	通过 WEB 升级	24
10.3	通过 ULINK 升级	24
10.4	通过 HF 生产工具批量升级.....	24
10.5	升级注意事项.....	24
11	SDK 升级	25
12	Example	27
12.1	通过 AT 命令发送 HTTP 请求	27
12.2	自定义 RELOAD, NLINK.....	28
12.3	定时器控制 nLink,nReady 灯闪烁	28
12.4	网络回调机制控制 NLINK 灯状态.....	28
12.5	通过 AT 命令操作用户配置文件.....	29
13	Flash 分布图.....	30
13.1	LPB100 Flash Layout.....	30

1 SDK 历史变更及注意事项

1,HSF-V1.00

第一版基于 HSF SDK 发布,支持硬件 LPB

2,HSF-V1.03

- 1,增加串口收发 API;
- 2,增加定时器 API
- 3,增加 SOCKETA/B 连接成功回调事情
- 4,SDK 中 example 和正式代码分开;
- 5,添加 SOCKETA,SOCKETB,发送 API;
- 6,增加 DEMO 工程
- 7,网络 SOCKET 回调增加 TCP 连接和断开时间

2,HSF-V1.13

- 1,增加对 LPB100 的支持;
- 2,中断模式增加边沿触发方式;
- 3,增加用户文件操作(保存用户配置)API;
- 4,解决 GPIO 中断配置不触发 Bug;
- 5,解决协议栈存在的一些 Bug;
- 6,解决 wifi 驱动存在的一些 Bug;

3,HSF-V1.17

- 1,增加线程看门狗接口;
- 2,支持 smartlink
- 3,支持 wps
- 4,添加修改定时器周期接口
- 5,添加 nvm 接口
- 6,添加自动升级功能(量产)
- 7,支持 2MB flash SDK,代码区扩展到 512KB
- 8,提供用户 128KB flash 操作接口
- 9,支持 tcp keepalive;
- 10, 解决 ULINK2 download 的总是失败问题;
- 11,支持 2MB SDK 小于 400KB 时候可以直接从 1MB SDK(1.03a)升级
- 12,发布 HSF-V1.17-201402141623 版本
- 13,优化 smartlink 流程;
- 14,添加支持 UART1

4,HSF-V1.18

- 1,解决 SDK 中用户 flash(228KB),当擦除后 100KBflash 的时候会把 wifi 固件擦除 Bug;
- 2,解决自动升级后会把用户 flash 擦除 Bug;
- 3,采用新的 freertos.lib 库文件;
- 4,优化 WIFI 稳定性;

5, 更新新的 ULINK 烧写 flash 配置文件 MV18X_16MB_V1.4.2.FLM(旧的文件为 MV18X_16MB_V1.4.1.FLM)

5, HSF-V1.20

- 1, 模块工作在 AP 模式, 长时间空闲放置, 连接 AP 小概率连不成功问题;
- 2, 模块工作在 STA 模式, 长时间空闲放置, 模块小概率出现不能正常工作问题;
- 3, 大量模块同时上下电, 同时去连接同一个路由器, 可能会概率出现连接不上问题;
- 4, 增加 AT+WIFI 命令, 控制 WIFI 开关;
- 5, 当模块 WIFI 异常的时候, 不再重启模块, 可以通过重启 WIFI 恢复;
- 6, 优化 TCP/IP 协议栈, 解决 TCP 多个连接同时连接问题, 不再有创建 socket 和接收 socket 数据一定在同一个线程的限制;
- 7, 解决模块发送网络数据包(数据部分全 0, 或者有很多连续 0, 长度超过 512) 导致模块死机问题;
- 8, 优化 wps, 增强兼容性;

6, HSF-V1.30

1. TCP 稳定性:
 - 改进 TCP 长期连接情况下的稳定性, 用户建立的 TCP 可长期保持连接和数据传输;
 - 频繁 TCP 反复连接断开情况下的稳定性, 用户可频繁连接断开 TCP 连接, TCP 每次可以连接成功并传输数据;
2. AP 稳定性:
 - AP, AP+STA 长期放置情况下可正常工作; (出于安全考虑, 此时 AP 应设加密方式)
3. Smartlink 改进:
 - 在失败时, 模块会自动重试;
 - 进入 smartlink, 放置一段时间(如 2 分钟以上), 仍可以工作, 不需重启;
4. 模块串口在模块出厂时会校准, 保证波特率的准确性;
5. 提高产测软件执行效率;
6. WPS 功能改进, 增加兼容性;
7. 新增 Websocket 库(client), 可以更好支持云应用;
8. 其他 Bug;

注意事项:

从 1.06 版本开始, 将支持 LPT100, LPT200 的 SDK 开发, LPT100, LPT200, LPB100 的升级文件不再通用, 每个类型只能支持对应的升级文件。每种类型都对应一个工程, 每个工程生成的串口升级我自动升级文件都不一样, 对应关系:

模块类型	工程名	串口升级文件	自动升级文件
LPB100	LPBS2W	LPBS2W.bin	LPBS2W_UPGARDE.bin
LPT100	LPT100S2W	LPT100S2W.bin	LPT100S2W_UPGARDE.bin
LPT200	LPT200S2W	LPT200S2W.bin	LPT200S2W_UPGARDE.bin

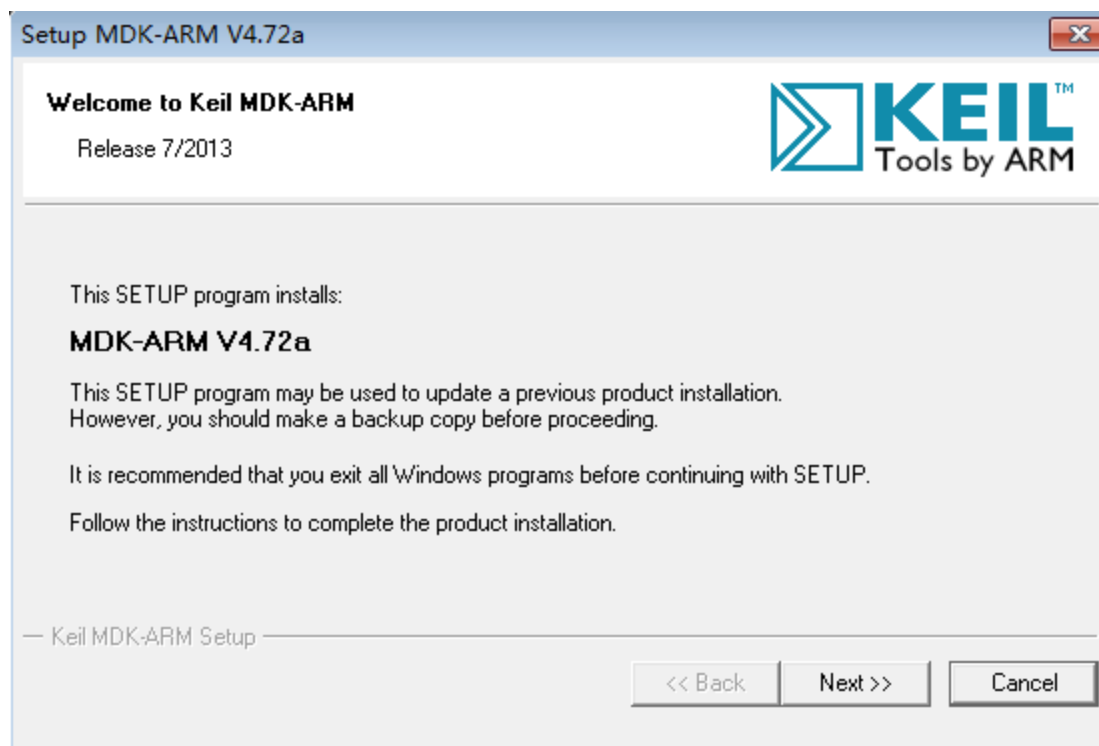
2 编译环境安装

LPB100 采用的是 Keil 的 MDK.

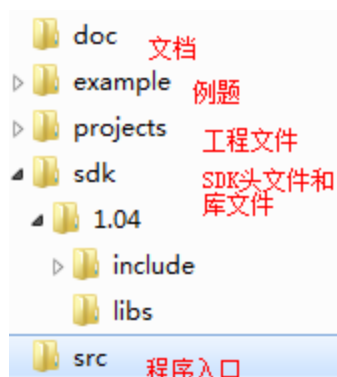
2.1 基于 Keil MDK 的 SDK 环境安装(LPB100)

LPB100 采用的是 Keil MDK.

1, 安装 Keil MDK, SDK 使用的是 MDK-ARM 4.72a 编译, 建议安装 4.72a 版本或者高版本



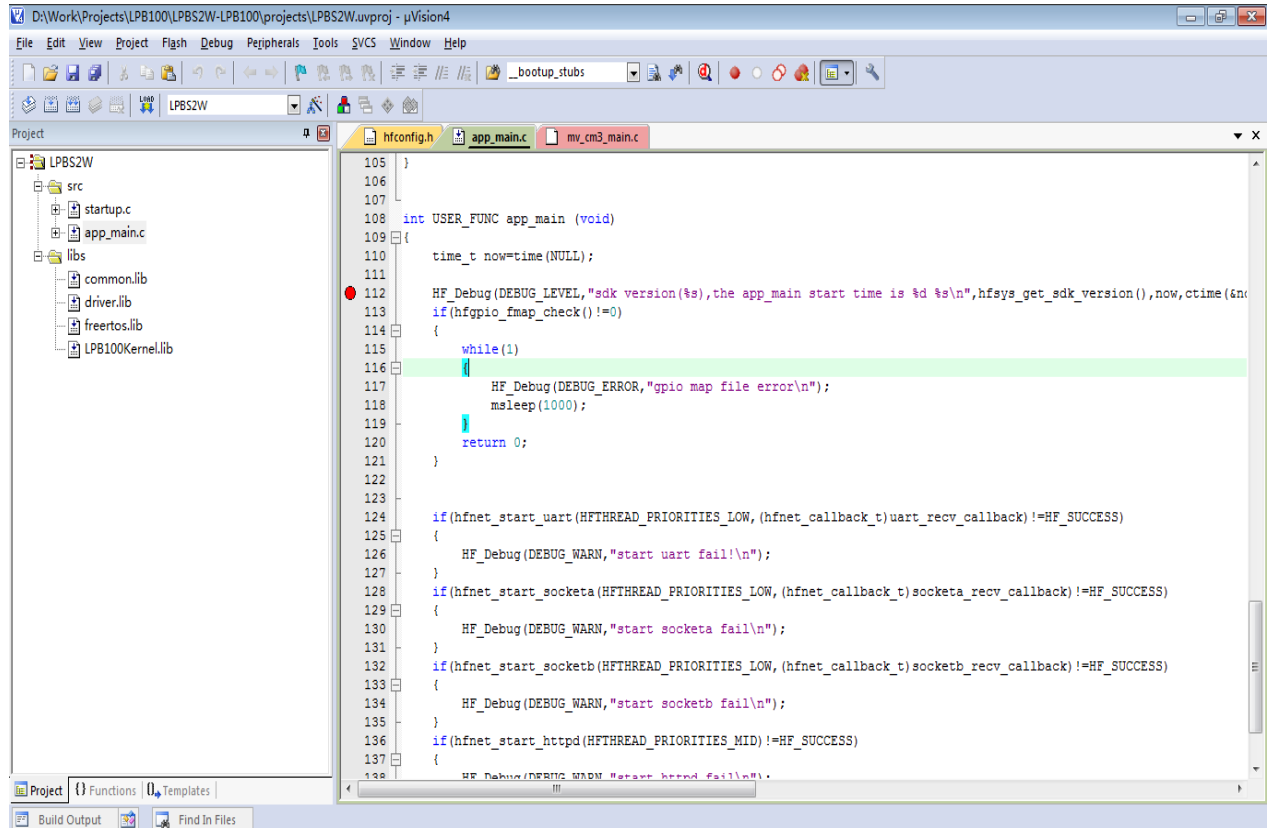
2, 解压 SDK, SDK 的目录如下:



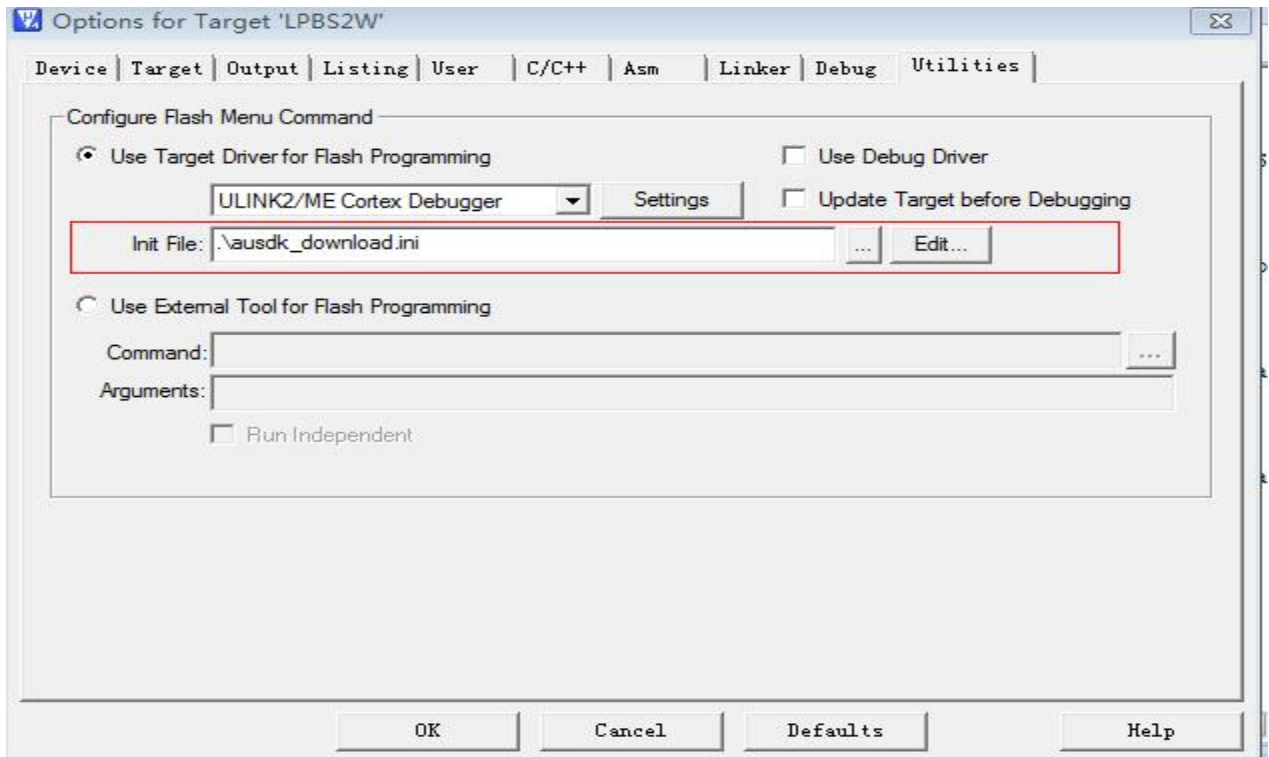
3, 把 doc 目录下的 “MV18X_16MB_V1.4.2.FLM” 拷贝到 “Keil MDK 安装目录\ARM\Firmware\”

下；如果没有这个路径，搜索一下 Flash 目录，把这个文件拷贝到 Flash 目录中去，把 ausdk_download.ini 拷贝到和工程文件同一个目录下面。

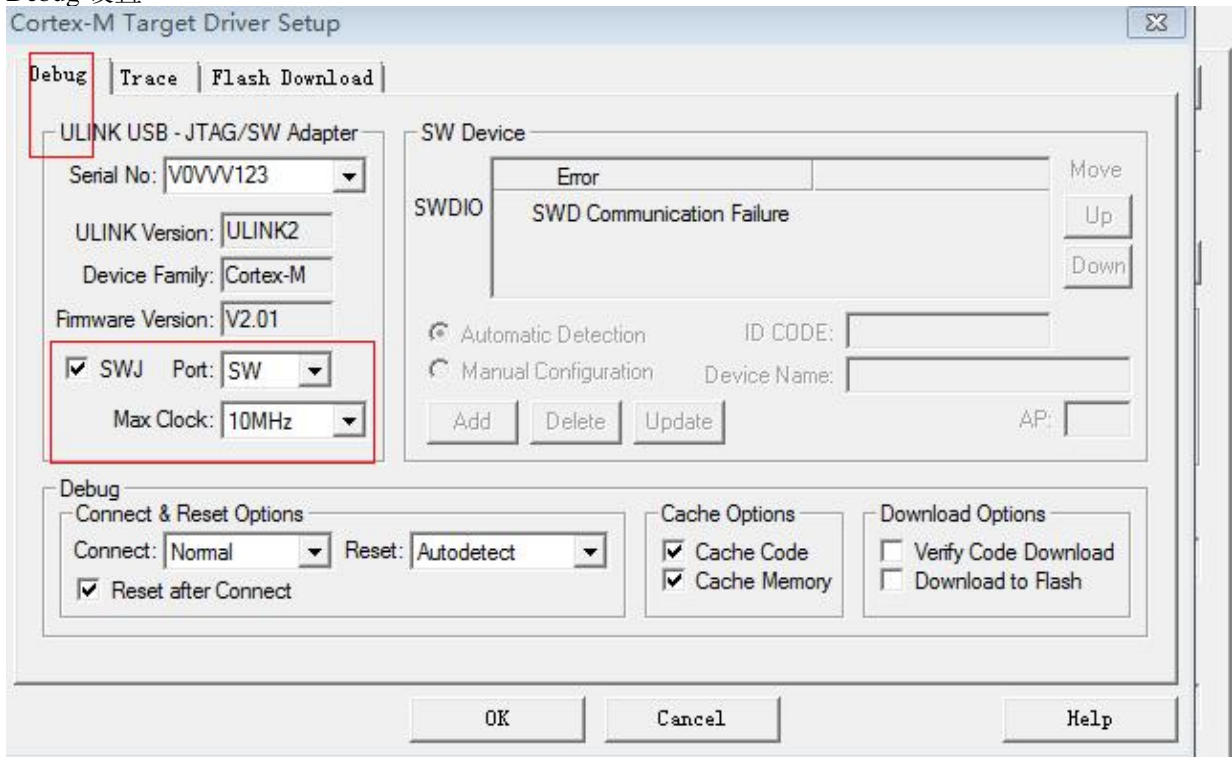
4. 打开 projects 目录下面 “LPBS2W.uvproj”

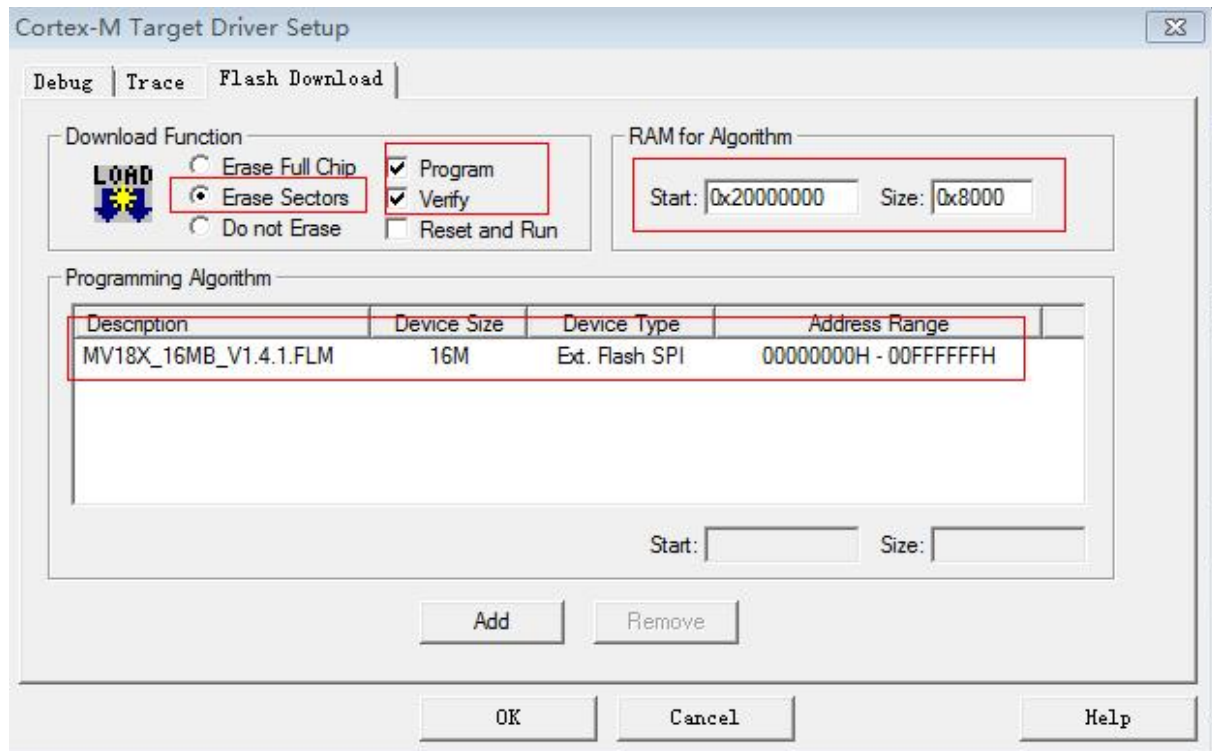


Utilities 设置选项








Debug 设置





4. 点击“build”，生成执行文件

 LPBS2W.axf	→ ULINK 下载文件	2013/10/17 11:00	AXF 文件	3,689 KB
 LPBS2W.bin	→ 通过串口升级文件	2013/10/17 11:00	VLC media file (...)	353 KB
 LPBS2W.fed		2013/9/26 22:48	FED 文件	71 KB
 LPBS2W.hex		2013/10/17 11:00	HEX 文件	992 KB
 LPBS2W.htm		2013/10/17 11:00	Firefox HTML D...	1,659 KB

3 资源分配

3.1 LPB100 资源分配

LPB100 1MB SDK 代码区占 400KB,LPB100 2MB SDK 代码区占 512KB.
如果不使用 SDK 自带机制, 只有 lwip 和 wifi, LPB100 最多有空闲 26KB RAM,和 100-200KB 的 ROM.

LPB100,LPT100 FLASH 都是 2MB, 可以用 2MB 的 SDK, LPT200 为 1MB Flash 只能用 1MB SDK.

4 用户怎么样添加自己的源代码

4.1 用户函数定义约定

返回类型 + **USER_FUNC** + 函数名称 + 参数

例如:

`void USER_FUNC test_func1(char *a);` `USER_FUNC` 为函数修饰符号, 为了更好的兼容性请加上 `USER_FUNC` 这个标识, 如果不加在有的平台编译出来的程序将无法运行

4.2 用户添加源代码文件

添加.c 文件, 基于 HSF 的源文件都要包含 `<hsf.h>` 头文件, 包含这个头文件后, 源代码里面可以调用基于 HSF 的 API 函数, 标准 Socket 函数; 如果要使用 libc 接口函数, 请 `#include` 相关的头文件, 例如如果调用字符串操作函数 `#include <string.h>`, 调用时间函数 `#include <time.h>` 等。

为了利于 SDK 升级, 请把不要在 `app_main.c` 文件里面添加太多代码, 最好只需要添加自己的一个入口函数。其它的源文件都放在自己的目录下面, 可以在 `src` 目录下面建一个目录放置自己的源代码, 这样升级的时候, 只需要把 `app_main.c` 文件修改一下, 其它的 SDK 文件全部用新的替换就可以了。

5 用户怎么样自定义 GPIO

5.1 PIN 脚的定义

HSF 采用一级映射表，功能码到 PIN 脚的映射；用户可以根据自己的需求随意定义每一个 pin 脚(除一些只能用做外设的 pin 脚除外)的功能。

兼容 HSF 架构的模块 PIN 脚属性定义,具体可以参考 `hfgpio.h` 文件。

当前 PIN 脚具有的属性有：

- F_GPI: 可以做输入脚;
- F_GPO: 可以做输出脚;
- F_GPIO: 既可以做输入脚, 又可以做输出脚;
- F_GND: 地;
- F_VCC: 电源脚;
- F_NC: 空脚;
- F_RST: 硬件复位脚;
- F_IT: 可以用做中断输入;
- F_Pt: 某一个外设接口的脚, 例如这个 PIN 为 SPI 中的 MOSI 脚;
- F_PWM: 可以用做 PWM.
- F_ADC: 可以用做 ADC

用户程序可以通过 `HF_M_PINNO(_pinno)` 来读取模块每一个 PIN 脚的属性, 其中 `_pinno` 为 pin 脚号, 模块的 PIN 脚号可以参考相关的数据手册, 也可以参考 `hfgpio.h` 文件, 里面有对每一个 PIN 脚进行定义。映射表以 LPB100 为准, LPT100 和 LPT200 引脚和 LPB100 对应关系:

	HF-LPB100	HF-LPT100	HF-LPT200
1	GND	1	16
2	SWCLK		
3	N.C		
4	N.C		
5	SWD		
6	N.C		
7	GPIO7		14
8	GPIO8		
9	DVDD	2	15
10	N.C		
11	GPIO11	10	
12	GPIO12	9	
13	GPIO13		
14	N.C		
15	GPIO15 (WPS)		
16	N.C		

17	GND		
18	GPIO18	8	
19	N.C		
20	GPIO20		
21	N.C		
22	N.C		
23	GPIO23		9
24	N.C		
25	PWR_SW	7	
26	N.C		
27	SPI_MISO		3
28	SPI_CLK		2
29	SPI_CS		4
30	SPI_MOSI		1
31	DVDD		
32	GND		
33	N.C		
34	DVDD		
35	N.C		
36	N.C		
37	N.C		
38	N.C		
39	UART0_TX	6	5
40	UART0_RTS		8
41	UART0_RX	5	6
42	UART0_CTS		7
43	nLink		13
44	nReady		11
45	nReload	3	12
46	N.C		
47	EXT_RESETn	4	10
48	GND		

5.2 系统固定功能码

当前 HSF 架构中存在的功能码有：

固定 PIN 脚的功能码有：

HFGPIO_F_JTAG_TCK
 HFGPIO_F_JTAG_TDO,
 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

```

上面的 功能码只能对应固定的PIN脚，以LPB100为例，
HFGPIO_F_JTAG_TCK只能对应HF_M_PIN(2),模块的第二个PIN脚，如果它对应的PIN脚可以做GPIO使用，我们可以把功能码对应的PIN脚设置为HFM_NOPIN，这样系统自动会把这个PIN脚配置成GPIO模式，以HFGPIO_F_UART0_TX为例，如果程序不使用串口，我们可以把HFGPIO_F_UART0功能码对应的PIN脚设置为HFM_NOPIN,这样这个脚可以配置成用户自定义的功能码。

下面功能码可以配置成任意有F_GPIO属性的PIN脚，或者HFM_NOPIN，如果配置成HFM_NOPIN说明
程序不使用这个功能.

```

//////////
HFGPIO_F_NLINK
HFGPIO_F_NREADY
HFGPIO_F_NRELOAD
HFGPIO_F_SLEEP_RQ //当前不支持这个功能
HFGPIO_F_SLEEP_ON //当前不支持这个功能

```

5.3 怎么样修改映射表

映射表在 app_main.c 中定义
const int hf_gpio_fid_to_pid_map_table[HFM_MAX_FUNC_CODE]

```

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, //
HFM_NOPIN, //
};

```

功能码实际上为 `hf_gpio_fid_to_pid_map_table` 的索引值，而数组的每一项值对应的为这个功能码实际对应的 PIN 脚。例如 `HFGPIO_F_NLINK` 功能码在 LPB 通用版模块中对应的 PIN 脚为第 43 脚。

如果用户要自定义功能码，请不要使用系统已经定义的值，用户功能码从 `HFGPIO_F_USER_DEFINE` 开始。

例1:

以LPB硬件为例，我们用一个脚来控制进入命令模式，定义为

`#define USERGPIO_F_ATCMD_MODE (HFGPIO_F_USER_DEFINE+0)`

对应的映射表:

```

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(16), // USERGPIO_F_ATCMD_MODE, LPB第16脚可以用来做中断输入
HFM_NOPIN,    //
};

源程序:
void hfgpio_interrupt_func(uint32_t,uint32_t)
{
    if(hfgpio_fpin_is_high(USERGPIO_F_ATCMD_MODE))
    {
        hfsys_switch_run_mode(HFSYS_STATE_RUN_CMD);
    }
    else
    {
        hfsys_switch_run_mode(HFSYS_STATE_RUN_THROUGH);
    }
}

int USER_FUNC app_main (void)
{
    //把USERGPIO_F_ATCMD_MODE对应的PIN脚配置成中断，上升沿触发。
    if(hfgpio_configure_fpin_interrupt(USERGPIO_F_ATCMD_MODE, HFPIO_IT_RISE_EDGE,
hfgpio_interrupt_func,1)!=HF_SUCCESS)
    {
        return -1;
    }
}

```

例2:

把评估版本上面nLink灯和nReady对换

```

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(44), //HFGPIO_F_NLINK
HF_M_PIN(43), //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(16), // USERGPIO_F_ATCMD_MODE, LPB第16脚可以用来做中断输入
HFM_NOPIN, //
};

```

修改后，编译后升级可以发现nLink灯和nReady等对换了。

例3:

把评估版本上面nLink,nReady指示灯用做别的，不使用当前的定义

```
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, //HFGPIO_F_NLINK
HFM_NOPIN, //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(16), // USERGPIO_F_ATCMD_MODE, LPB第16脚可以用来做中断输入
HFM_NOPIN, //
};
```

修改后，编译后升级可以发现nLink灯和nReady不管怎么样都不亮了。

5.4 怎样操作 PIN 脚

HSF 没有提供直接操作 PIN 脚 API，要操作 PIN 脚，只能通过操作功能码。因此要操作某一个 PIN 脚，先要定义一个功能码，在 hf_gpio_fid_to_pid_map_table 表中把这个功能码对应的 PIN 脚的属性写入。

操作功能码对应的 PIN 脚 API 可以参考《HSF1.xx API 参考手册》

6 用户怎样添加自定义 AT 命令

HSF 提供 AT 命令解析引擎，用户可以快速的添加自己的 AT 命集；用户可以通过下面表来添加自己的 AT 命令集

```
const hfat_cmd_t user_define_at_cmds_table[]=
{
    {"UMYATCMD",hf_atcmd_myatcmd," AT+UMYATCMD=code\\n",NULL},
    {NULL,NULL,NULL,NULL} //最后一项一定为NULL,解析引擎通过NULL来判断这个表格的大小
};
```

user_define_at_cmds_table中每一项代表一条AT命令，每一条AT命令表项有4个属性，分别为：

- AT 命令的名称；
- AT命令的入口函数
- AT命令的帮助说明(AT+H的时候显示)
- 最后一项保留以后使用

用户要添加自定义AT命令步骤：

- 1， 添加AT命令名称；
- 2， 添加AT命令入口函数的实现；
- 3， 添加对这条命令的帮助说明

提供的 SDK 中的 example/attest.c，编译出来会添加一条 AT+UMYATCMD 的命令，通过 AT+H 可以看到对 AT+UMYATCMD 的说明,如下图

```
AT+SOCKB: Set/Get Parameters of socket_b.
AT+TCPLKB: Get The state of TCP_B link.
AT+TCPTOB: Set/Get TCP_B time out.
AT+TCPDISB: Connect/Dis-connect the TCP_B client link.
AT+RCVB: Recv data from socket_b
AT+SNDB: Send data to socket_b
AT+MDCH: Put on/off automatic switching WIFI mode.
AT+RELD: Reload the default setting and reboot.
AT+SLPEN: Put on/off the GPIO7.
AT+RLDEN: Put on/off the GPIO45.
AT+Z: Reset the Module.
AT+MID: Get The Module ID.
AT+VER: Get application version.
AT+UMYATCMD=code
AT+NTIME:set/get system time
AT+NMEM:show system memory stat
AT+NDBG: set/get debug level
AT+H:show help
+ok
```

自定义AT命令帮助说明

7 怎么样通过串口打印调式信息

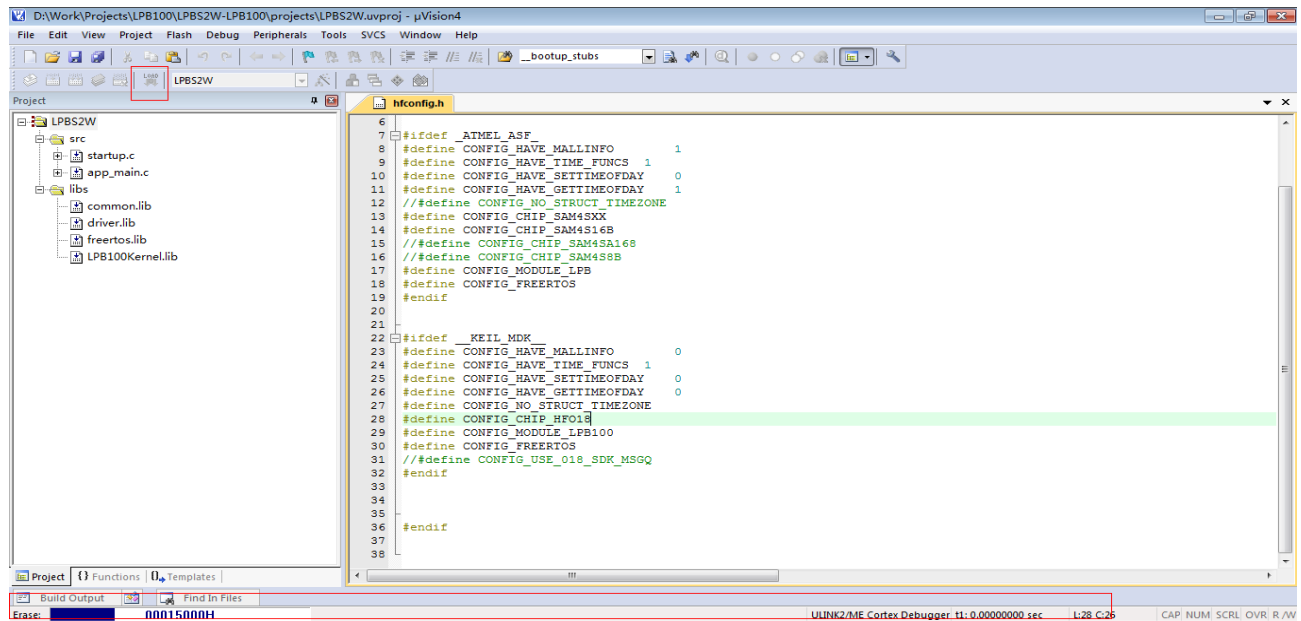
如果程序想通过串口打印调式信息，HSF 中提供了 `u_printf` 和 `HF_Debug` 两个 API 函数，默认情况下程序中调用这两个函数是不会有打印信息出来的，因为默认调式是关闭的，要通过 `hfdbg_set_level` 打开或者调式串口输出，或者 AT 命令 "AT+NDBGL=2" 打开，"AT+NDBGL=0" 关闭

(注.程序最后发布的时候要把 **debug** 模式关闭，因为 **debug** 模式模式下硬件看门狗会被关掉)

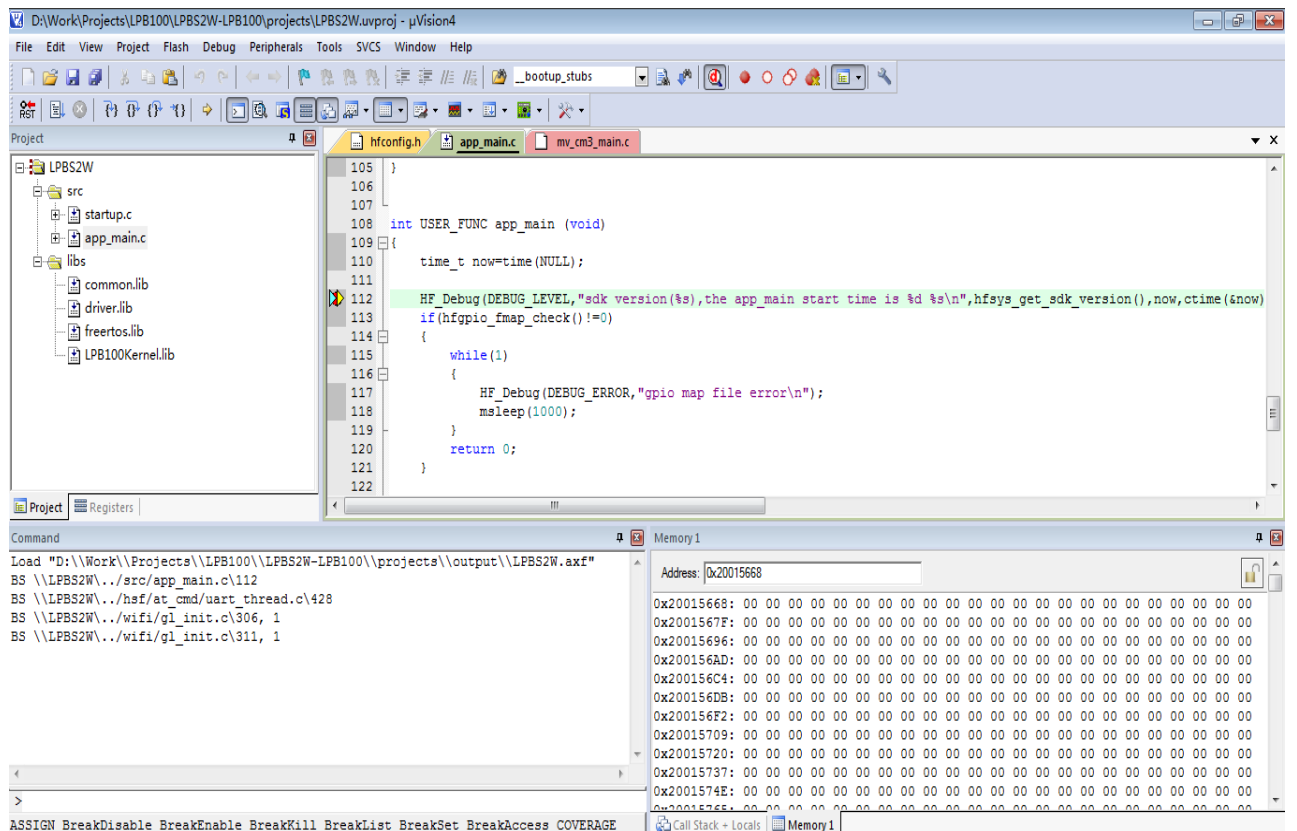
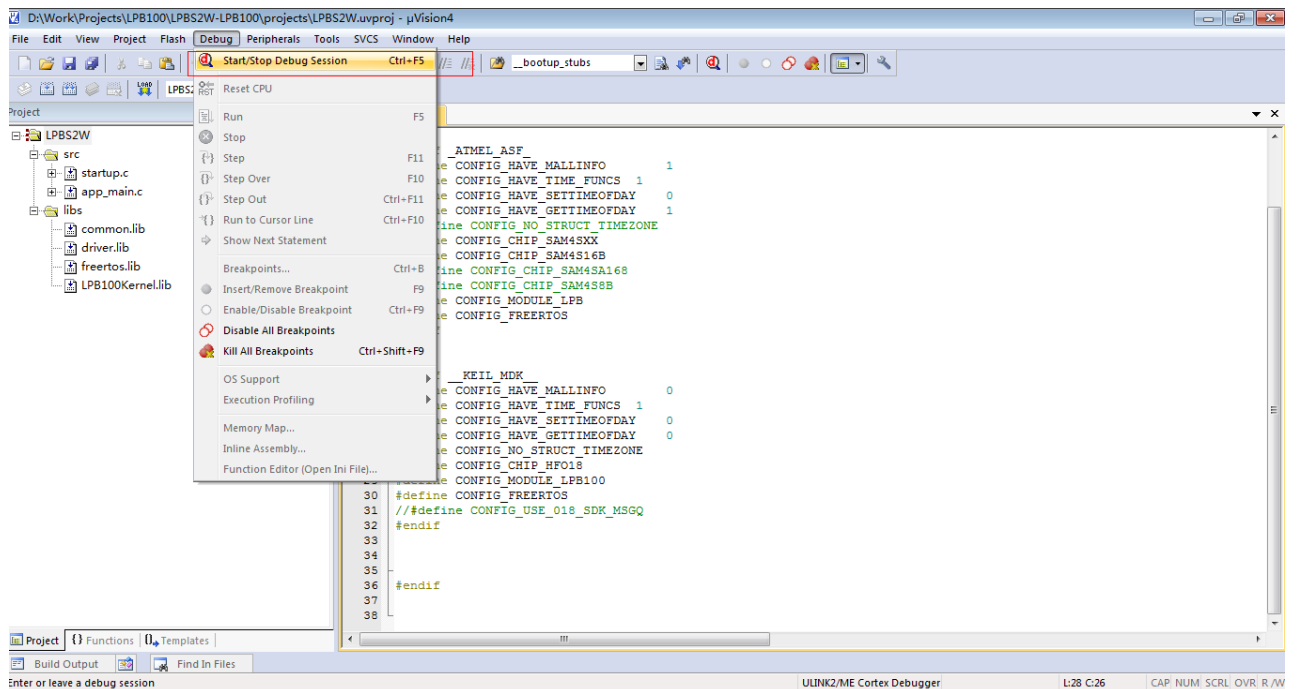
8 Keil-MDK 中用 ULINK 在线调式程序

如果是基于 Keil MDK 的环境，LPB100 评估板一定要采用 ULINK2 来调式。

- 1,先点击”build”按钮（或者按”F7”），生成目标文件；
- 2,确定 ulink 设备已经连接到设备，确保当前固件是以 Debug 模式运行,可以通过 AT+NDBGL 命令来设置；点击”download”按钮，把目标文件下载到 LPB 模块中去；

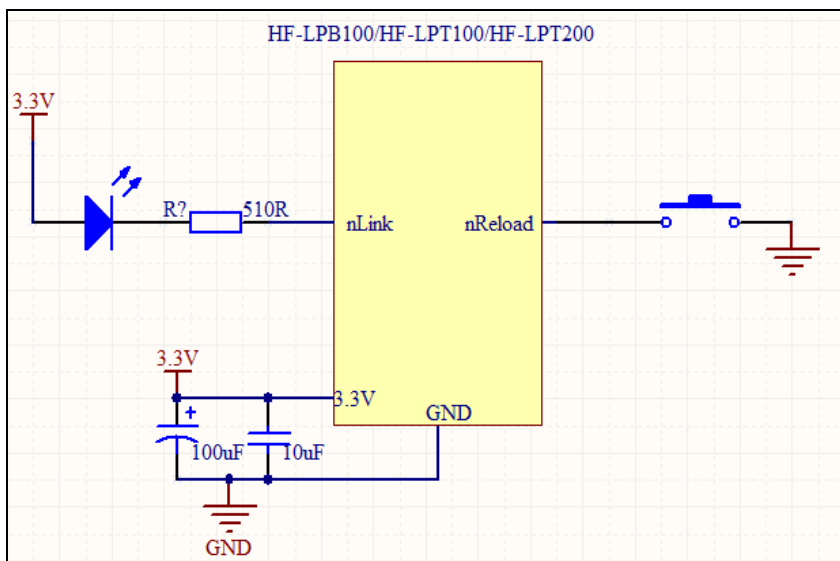


3,选择菜单“Debug”-“Start/Stop Debug Session”,开始调式,现在可以就可以开始调式自己的程序了。



9 硬件推荐连接

9.1 硬件推荐连接



nReload 按键的功能:

1. 模块上电时，如判断该引脚为低（按键按下），则模块进入批量无线升级、配置模式。
(参考用户手册附录 D 从汉枫网站下载生产工具，支持客户用于批量升级、配置)
2. 上电后，短按该键 (<3S)，则模块进入 Smart Link 配置模式，等待 APP 进行密码推送；
(参考用户手册附录 D 从汉枫网站下载 SmartLink APP，用于一键配置模块)
3. 上电后，长按该键 (>=3S)后松开，则模块恢复汉枫出厂设置。

注意：后续客户如需批量配置出厂设置或升级软件，强烈建议引出该引脚。

nLink 指示的功能:

1. 在无线批量升级、配置模式中做 LED 指示，提示配置或升级完成；
2. 在 Smart Link 配置模式，慢闪提示 APP 进行智能联网；
3. 在正常模式，做为 WiFi 的连接状态指示灯；

注意：后续客户如需批量配置出厂设置或升级软件，强烈建议引出该引脚。

10 怎样升级程序

10.1 通过串口升级

按住 **reload** 键下模块断电再上电，同时按下空格键，这个时候就可以进入 **boot** 程序的命令行，选择升级固件来升级

```
HF-LPB100 Bootloader v1.0.5, please entry code to choose :
'B': Clean All Config.
'F': Program Firmware. 升级WIFI 固件
'N': Program NVRAM data.
'S': Program application. 升级应用程序
'G': Run applications.
```

10.2 通过 WEB 升级

通过浏览器访问 LPBXX 的 webserver 来升级

10.3 通过 ULINK 升级

参考 Keil MDK 帮助帮助文档

10.4 通过 HF 生产工具批量升级

请参考《HF-LPB100 模组升级流程》

10.5 升级注意事项

由于 LPB100,LPT100 以前都是采用 1MB SDK (1.03a) 编译的，如果要从 1.03a 升到 2MB SDK 升级分两种情况:

- 1, 2MB SDK 编译出来的程序小于 400KB, 可以直接用上面升级程序直接升级.
- 2, 2MB SDK 编译出来的程序大于 400KB, 不能直接充 1.03a 升级, 需要升级一个过度程序 (2MB SDK 编译出来的小于 400KB 的程序), 在 sdk 中 doc 中有两个过度程序

LPBS2W_1MB_TO_2MB.bin 如果是串口升级可以用这个

LPBS2W_1MB_TO_2MB_UPGARDE.bin 如果用 web 或者用 HF 生产工具用这个.

升级过度程序之后再升级大于 400KB 的程序。如果不升级过度程序直接升级模块的 MAC 地址, 配置, 和 WIFI 固件可能会被擦掉, 程序无法正常启动。这个是只能通过串口 bootloader 把 wifi 固件升级进出. 程序才能够正常启动。

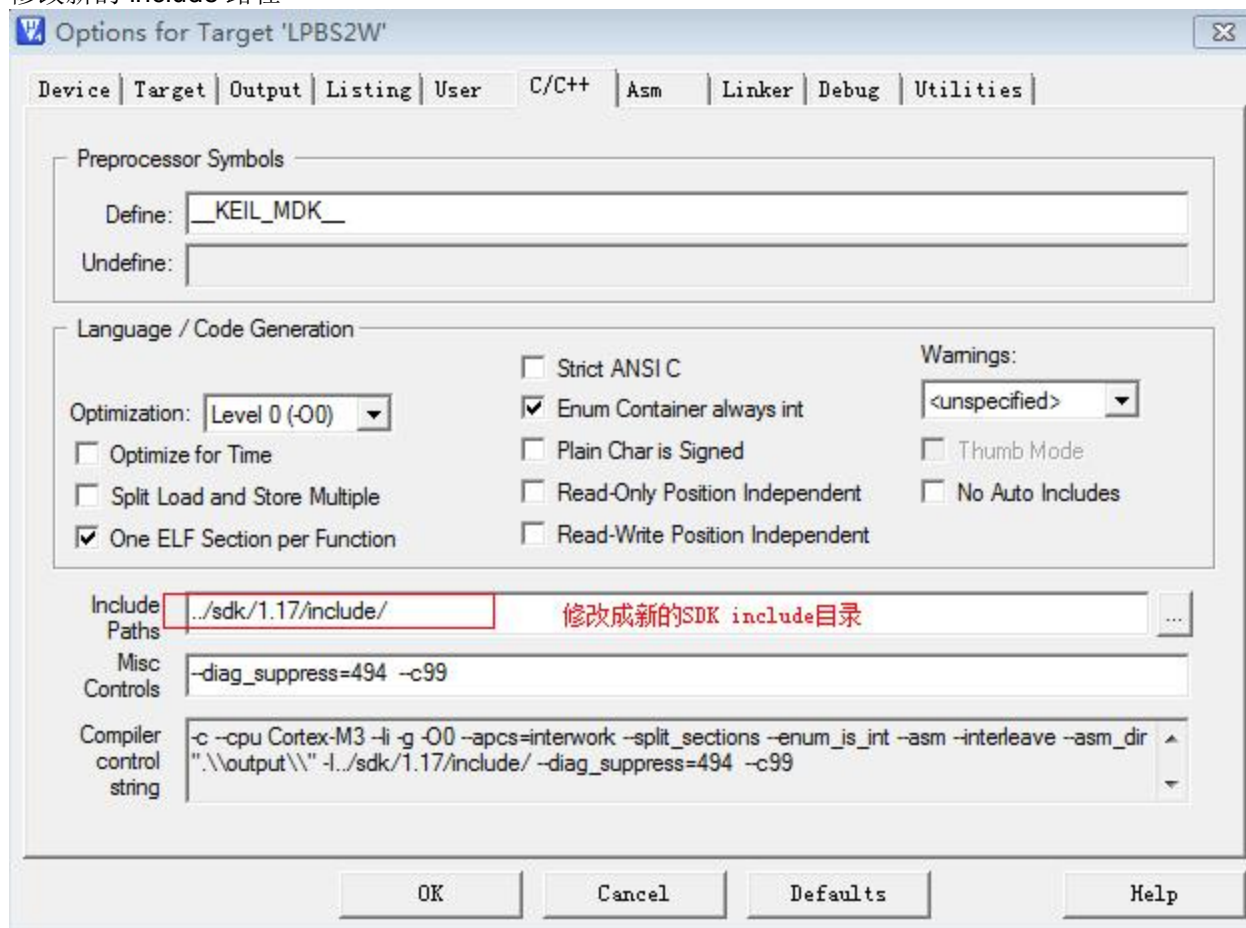
1MB SDK 之间升级, 2MB SDK 之间升级, 没有上面限制, 可以直接升级

11 SDK 升级

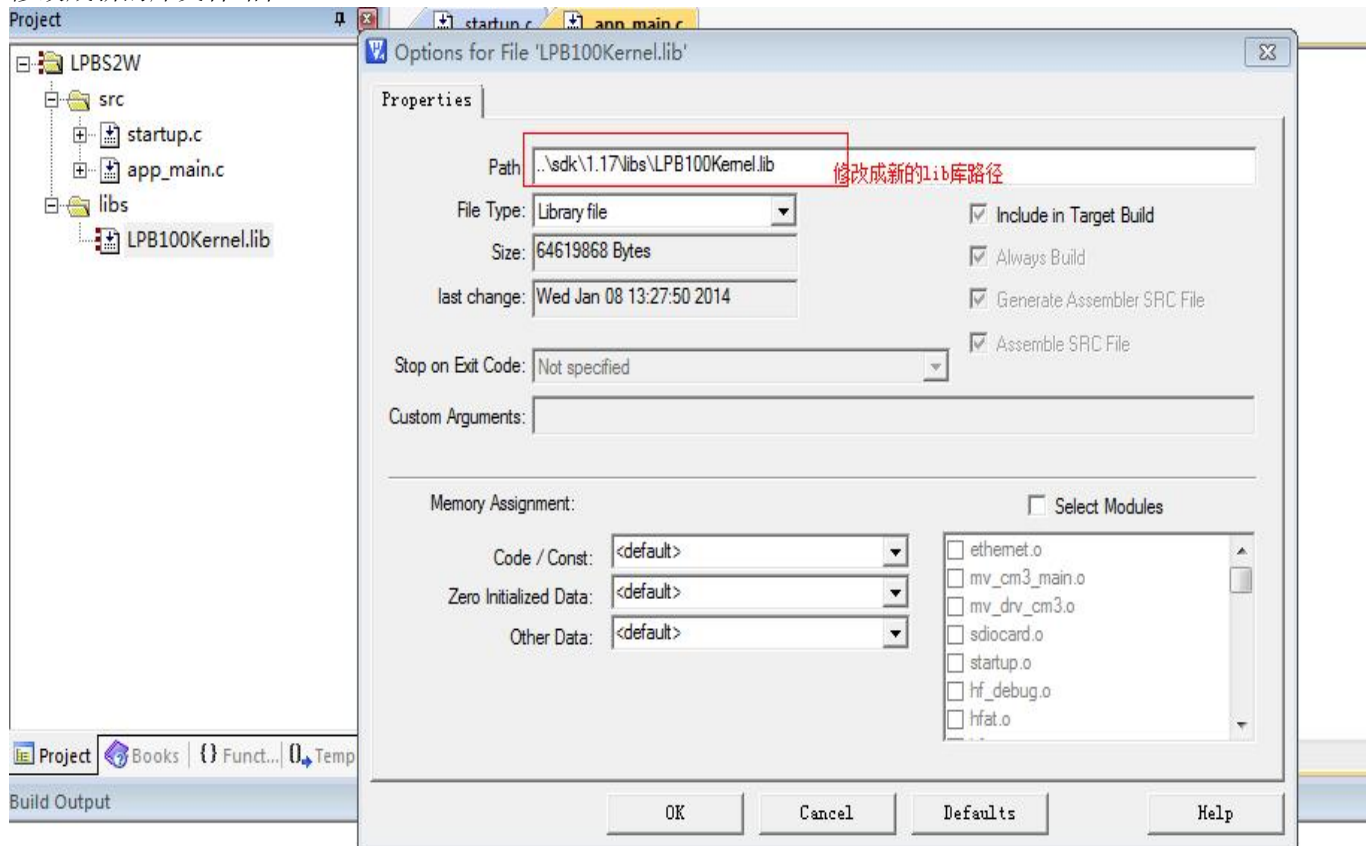
新的 SDK 发布，一般都会兼容老版本 SDK，用户最好采用新的 SDK 工程文件，只需要把自己的代码拷贝到新的 SDK 工程中去，把自己的 `app_main.c` 替换 SDK 的。

或者根据新的 SDK 的工程文件修改自己的工程文件，把 `sdk/` 中新的 `lib` 和 `include` 目录替换自己旧的工程中的 `lib` 和 `include`，把新的 `startup.c` 文件替换旧的 `startup.c`。修改工程配置。（这中方式不推荐，采用上面的兼容性要好）

修改新的 `include` 路径



修改成新的库文件路径



12 Example

Example 对应的工程为 **LPBExample.cproj**,当前有 5 个例子,可以通过 **example.h** 文件宏来选择要编译哪一个例子。如果要编译 **LPBExample.cproj** 工程,请把这个工程设置为启动工程; (如果采用的是 Keil MDK,工程文件放在 **example/projects** 下面,点击运行基于可以了),运行例子需要通过“AT+NDBGL=2”来设置消息显示级别为 2,这样 **u_printf** 函数可以通过串口打印调试信息。

```
#ifndef _H_EXAMPLE_H_H_H_
#define _H_EXAMPLE_H_H_H_

#define USER_AT_CMD_DEMO 1    AT 命令 Demo
#define USER_GPIO_DEMO 2    GPIO 控制 Demo
#define USER_TIMER_DEMO 3    定时器 Demo
#define USER_THREAD_DEMO 4    多线程 Demo
#define USER_CALLBACK_DEMO 5    SOCKET A/SOCKET B 回调机制
#define USER_FILE_DEMO 6    用户配置文件 Demo
#define USER_FLASH_DEMO 7    用户 Flash 操作
#define USER_SOCKET_DEMO 8    标准 socket Demo
#define USER_IR_DEMO 9    红外遥控 Demo

//通过下面可以选择不同的例子进行编译
#define EXAMPLE_USE_DEMO    USER_CALLBACK_DEMO
```

12.1 通过 AT 命令发送 HTTP 请求

代码在 **example/at/attest.c**;对应的宏为 **USER_AT_CMD_DEMO**,通过这个例子可以了解怎么样添加用户自定义 AT 命令,熟悉 **httpc** 接口函数的用法;

运行结果,在命令模式可以通过 AT 命令发送 HTTP 请求,把远程服务器返回的数据通过串口打印出来。

编译升级后通过串口工具执行 **AT+UMYATCMD=1,www.baidu.com** 结果:


```

AT+UMYATCMD=1,www.baidu.com
send_at_cmd [207] name:UMYATCMD [3]
UMYATCMD
1
www.baidu.com
<!DOCTYPE html><!--STATUS OK--><html><head><meta http-equiv="content-type" conte
nt="text/html; charset=utf-8"><title>獨惧害涓€涓嶈紕浣犳氮鑼ラ厶</title><style>h
tml,body{height:100%;html{overflow-y:auto;}#wrapper{position:relative;_position:;
min-height:100%;#content{padding-bottom:100px;text-align:center;}#ftCon{height:10
0px;position:absolute;bottom:44px;text-align:center;width:100%;margin:0 auto;z-i
ndex:0;overflow:hidden;}#ftConw{width:720px;margin:0 auto}body{font:12/20px;margi
px arial;text-align:;background:#fff}body,p,form,ul,li{margin:0;padding:0;list-s
tyle:none}body,form,#fm{position:relative}td{text-align:left}img{border:0}a{colo
r:#00c}a:active{color:#f60}.bg{background-image:url(http://s1.bdstatic.com/r/www
/cache/static/global/img/icons_0bf3824a.patc/glng);background-repeat:no-repeat;
background-image:url(http://s1.bdstatic.com/r/www/cache/static/global/img/icons
_0bf2f8a8.gif)}.c-icon{display:inline-block;width:14px;height:14px;vertical-alig
n:text-bottom;font-style:normal;overflow:hidden;background:url(http://s1.bdstatic
.com/r/www/cache/static/global/img/icons_0bf3824a.png) no-repeat 0 0;background
-image:url(http://s1.bdstatic.com/r/www/cache/static/global/img/icons_0bf2f8a8.
gif)}.c-icon-triangle-down{background-position:480px -24px}.c-icon-chevron
-unfold2{background-position:-504px -24px}#u{color:#999;padding:4px 10px 5px 0;t
ext-align:right}#u a{margin:0 5px}#u .reg{margin:0}#m{width:720px;margin:0 auto}
#nv a,#nv b,.btn,#lk{font-size:14px}#fm{padding-left:110px;text-align:left;z-ig
-left:110px;text-indent:1}input{border:0;padding:0}#nv{height:19px;font-size:16px;m
argin:0 0 4px;text-align:left;text-indent:137px}.s_ipt_wr{width:418px;height:30p
x;display:inline-block;margin-right:5px;background-position:0 -288px;border:1px
solid #b6b6b6;border-color:#9a9a9a #cdcdcd #cdcdcd #9a9a9a;vertical-align:top}.s
_ipt{width:405px;height:22px;font:16px/22px arial;margin:5px 0 0 7px;background:
#fff;outline:0;-webkit-appearance:none}.s_btn{width:95px;height:32px;padding-top
:2px\9;font-size:14px;background-color:#ddd;background-position:0 -240px;cursor:
pointer}.s_btn_h{background-position:-240px -240px}.s_btn_wr{width:97px;height:3
4px;display:inline-block;background-position:-120px -240px;*position:relative;z-
index:0;vertical-align:top}#lg_img{vertical-align:top}#lg_iign:top;margin-b
ottom:3px}#lk{margin:33px 0}#lk span{font:14px "瀹嬩綋"}#lm{height:60px}#lh{marg
in:16px 0 5px;word-spacing:3px}.tools{position:absolute;top:-4px;*top:10px;right

```

12.2 自定义 RELOAD, NLINK

代码在 example/gpio/gpiotest.c,对应的宏为 USER_GPIO_DEMO 通过这个例子可以了解怎么样自定义 GPIO 脚, 修改通用版自带的 PIN 脚功能, 熟悉 GPIO API 函数的用法。

运行结果, 按下 RELOAD 键不再是恢复出厂设置, 而是 nLink 灯开关, nLink 灯在 STA 连接成功后不再点亮, 而是在 tcp 连接成功后点亮, 按下 RQ 按键, 如果在 DEBUG 模式下会打印数据 (DEBUG 模式可以通过 AT+NDBG=2 来设置)

12.3 定时器控制 nLink,nReady 灯闪烁

代码在 example/timer/timertest.c;对应的宏为 USER_TIMER_DEMO,通过这个例子可以了解线程的创建, 定时器的创建, 以及相关的 API 函数的用法

运行结果, nLink,nReady 灯以 1HZ 的频率闪烁。

12.4 网络回调机制控制 NLINK 灯状态

代码在 example/netcallback.c 中, 对应的宏为 USER_CALLBACK_DEMO,通过这个例

子可以熟悉 `socketat/b` 发送 API,串口发送 API,以及网络回调处理机制。

执行结果,当远程和模块通过 `SocketA` 连接成功后,模块主动发送“CONNECT OK”给远程,远程发送“GPIO_NLINK_LOW”(没有换行符号,下面格式都是这样) nLink 灯熄灭,“GPIO_NLINK_HIGH” nlink 灯点亮,“GPIO_NLINK_FLASH” nLink 灯闪烁,其它字符透传给串口。

12.5 通过 AT 命令操作用户配置文件

代码在 `example/filetest.c` 中,对应的宏为 `USER_FILE_DEMO`,通过这个例子我们可以熟悉用户文件 API 的用法,用户自定义 AT 命令的用法;

执行结果,程序多了一条 AT 命令 `AT+FTEST`,通过这条命令可以操作用户文件,
`AT+FTEST=code,offset,len/value`

code: 0 读取文件的长度;`AT+FTEST=0,0,0`

code: 1 读取文件的内容,并通过串口打印出来;`AT+FTEST=1,0,100`

code: 5 读取文件的内容,并通过串口以 hex 格式打印出来;`AT+FTEST=1,100,100`

code: 2 写文件; `AT+FTEST=2,0,test123456789`

code: 3 测试文件接口的正确性;`AT+FTEST=3,0,0`

code: 4 把整个用户文件内容快速清零;`AT+FTEST=4,0,0`

offset: 文件的偏移量;

len:要读取文件的长度

value:要写入到文件的字符串

13 Flash 分布图

13.1 LPB100 Flash Layout

LPB100
2MB Flash Layout

0x00000000	程序代码区 512KB
0x00080000	保留 248KB
0x000BE000	参数配置 16KB
0x000C2000	保留16KB
0x000C6000	WIFI 固件 152KB
0x000EC000	保留28KB
0x000F3000	内部使用
0x000FD000	用户配置文件 8KB
0x000FF000	
0x00100000	WEB文件区 300KB
0x0014B000	保留区 212KB
0x00180000	程序升级备份区 512KB
0x001FFFFF	

© Copyright High-Flying, May, 2011

The information disclosed herein is proprietary to High-Flying and is not to be used by or disclosed to unauthorized persons without the written consent of High-Flying. The recipient of this document shall respect the security status of the information.

The master of this document is stored on an electronic database and is “write-protected” and may be altered only by authorized persons at High-Flying. Viewing of the master document electronically on electronic database ensures access to the current issue. Any other copies must be regarded as uncontrolled copies.

<结束>