API 接口说明

变更记录

变更时间	变更内容	版本号	提交变更者
2019.3.11	初版	0.0.1	Ender.Wigg
2019.3.15	支持多FPGA固化，增加lib配置宏	0.1.0	Ender.Wigg
2019.3.20	将CAN中断函数放置到hhd32f10x_it.c	0.2.0	Ender.Wigg
2019.3.25	增加切换网口说明	0.3.0	Ender.Wigg
2019.4.11	增加通过上位机选择FPGA功能	0.4.0	Ender.Wigg
2019.4.25	增加RTT nano 内核，提供BSD 网络API	1.0.0	Ender.Wigg

基本信息描述

基本参数

系统主频	**60MHZ**
网络连接速度	**100M/10M**
系统调度定时器中断周期	**10ms**
CAN波特率	**1Mbps/500Kbps**
操作系统	**RTT nano**
网络协议栈	**LWIP1.3.2**
默认IP	**192.168.2.198**

可用外设

- GPIO
- MAC
- CAN
- SPI

系统基本功能初始化

```
1  void base_init(void)
```

描述：完成系统运行环境的构建，具体包括，相关外设的初始化，根据GPIO 引脚分配表初始化相关引脚，设置系统调度定时器中断等功能。在 main 函数中**必须**首先调用该函数。

初始化的外设如下：

GPIO（根据`pins_table`进行初始化）

MAC

SPI

SystemTick

构建运行环境：

LWIP 协议栈初始化

创建 FPGA 远程调试 网络服务器

创建 FPGA 配置文件固化 网络服务器

参数：void

返回值： void

GPIO

GPIO 初始化

GPIO初始化采用 pin描述结构，即期望使用的GPIO引脚需要在 `pins_table` pin描述结构中进行描述，库会自动的对该引脚进行初始化, pin描述接口需要借助宏`__HHD_PIN`来完成填充。例如：

```
1     __HHD_PIN(C,    7,    0,    INPUT,
   GPIO_Mode_DEF)  //ARM_FPGA1_IO1
```

原型：

```
#define __HHD_PIN(gpio, gpio_index, af, dir,mode)\
            {RCC_APB2Periph_GPIO##gpio,\
            GPIO##gpio,\
            PIN##gpio_index,\
            &IOCON->P##gpio##gpio_index,\
            af,\
            dir,\
            mode\
            }
```

参数：

gpio：引脚所在的PORT,[A,B,C,D,E,F]

gpio_index：引脚在组内的序号[0,15]

af：引脚服用功能，一般作为GPIO af 填0

dir：引脚方向，输入INPUT，输出 OUTPUT

mode：此处填默认值 GPIO_Mode_DEF

例如：将PD10 配置为输出

```
__HHD_PIN(D,    10,    0,    OUTPUT,
GPIO_Mode_DEF)
```

GPIO引脚控制

```
void GPIO_WriteBit(HHD32F_GPIO_TypeDef* port, uint16_t
GPIO_Pin, BitAction BitVal)
```

描述：对指定的引脚控制输出高低电频

参数：

port, 该GPIO所在的组，[GPIOA, GPIOB, GPIOC, GPIOD, GPIOE, GPIOF]

GPIO_Pin, 引脚组内编号，[0,15]

BitVal，输出电平，输出低电平Bit_RESET，输出高电平 Bit_SET 其中 枚举 BitAction定义如下：

```
typedef enum
{ Bit_RESET = 0,
  Bit_SET
}BitAction;
```

返回值：void

SPI

使用引脚

针对采集板

ARM 引脚名称	SPI 功能	FPGA 引脚名称
PC8	CS	ARM_FPGA1_IO1
PC10	CLK	ARM_FPGA1_IO2
PC11	MISO	ARM_FPGA1_IO3
PC12	MOSI	ARM_FPGA1_IO4

针对4个FPGA板

ARM 引脚名称	SPI 功能	FPGA 引脚名称
PC7	CS	ARM_FPGA1_IO1
PC8	CLK	ARM_FPGA1_IO2
PC10	MISO	ARM_FPGA1_IO3
PD11	MOSI	ARM_FPGA1_IO4

SPI 接口初始化

```
1  void SPI_To_FPGA_Init(void)
```

描述：函数用于初始化用于与FPGA通信的SPI接口

参数：void

返回值：void

SPI 连续写据到FPGA

```
1  int SPI_To_FPGA_Wirte(uint8_t fpga, uint8_t addr,
   uint8_t *data, int len)
```

描述：将data指向的数据的前len个字节写到指定的FPGA，addr 为写入到FPGA的起始地址，地址会自动向后递增

参数：

fpga，选中的FPGA，[0，3]

addr，起始地址，[0, 0x3F]

data, 待写入的数据指针

len，数据长度，[0, 0x40]

返回值：写入到FPGA的数据长度

SPI 从 FPGA读数据

```
int SPI_To_FPGA_Read(uint8_t fpga, uint8_t addr,
uint8_t *data, int len)
```

描述：从选中的FPGA的addr地址开始，读出len个字节存在data所指向的位置

参数：

fpga，选中FPGA，[0,3]

addr，起始地址，[0, 0x3F]

data, 数据缓存

len，期望读出的数据长度，[0, 0x40]

返回值：从FPGA读出数据的长度，该长度可能小于 len

关于不同板卡使用不同引脚的说明

当前已知 时序板和FPGA板(板上有4个FPGA) 使用的SPI引脚不一致，因此需要特别注意，不同板子的差异，代码库通过文件 `HHD1705_lib.h` 中两个宏来控制，这两个宏必须是互斥的。

```
#define xCFG_SELECT_SPI_IO_0    /* 使用 PC9, PC10,
PC11,  PC12*/
#define CFG_SELECT_SPI_IO_1     /* 使用 PC7, PC8, PC10,
PC11,  该模式支持访问4片FPGA*/
```

时序板时，定义`CFG_SELECT_SPI_IO_0`，FPGA板时，定义FG_SELECT_SPI_IO_1。

CAN

CAN接口初始化

```
1  void can_init(HHD32F1_CAN_TypeDef *can, EN_BAUD baud,
   uint32_t filterID, uint32_t mask);
```

描述：初始化指定的CAN接口

参数：

can ， can接口，即期望使用的can接口，可指定为如下两个宏中的任意一个：

`CAN1`,`CAN2`

baud，接口波特率，当前支持1Mbps和500Kbps,通过如下两个枚举选择：

`CAN_BAUD_500K`,`CAN_BAUD_1M`

filterID，设置过滤ID，取值范围[0, 0x3FFFFFFF]

mask，设置过滤ID的掩码，如果bit置为0，则ID中的该bit会严格比较；为1则该为不经进行比较

返回值：void

CAN发送一帧数据

```
1  int CAN_Transmit(HHD32F1_CAN_TypeDef *can, CanTxMsg
   *TxMessage)
```

描述：从指定的can接口发送一帧数据

参数：can ， can接口，即使用的can接口，可指定为如下两个宏中的任意一个：

`CAN1`,`CAN2`

TxMessage，待发送的数据和帧描述信息，CanTxMsg定义如下：

```
typedef struct
{
  uint32_t StdId;  /*!< 标准帧ID，取值范围[0,7FF].*/

  uint32_t ExtId;  /*!< 扩展帧ID，取值范围
[0,0x1FFFFFFF].*/
  uint8_t IDE;     /*!< 帧类型，标准帧 CAN_Id_Standard
                              扩展帧 CAN_Id_Extended
*/
  uint8_t RTR;     /*!< 传输消息类型   数据 CAN_RTR_Data
                                    远程帧
CAN_RTR_Remote*/
  uint8_t DLC;     /*!< 该帧所携带的数据的长度，取值范围
[0,8] */
  uint8_t Data[8]; /*!< 数据负载 */
} CanTxMsg;
```

返回值：发送状态 `CAN_TxStatus_Failed` 发送失败

`CAN_TxStatus_Ok` 发送成功

CAN接收一帧数据

```
int CAN_Receive(HHD32F1_CAN_TypeDef *can, uint8_t
FIFONumber, CanRxMsg* RxMessage)
```

描述：从指定的can接口接收一帧数据

参数：can ，can接口，即使用的can接口，可指定为如下两个宏中的任意一个：

`CAN1`,`CAN2`

FIFONumber，为了兼容ST API，该参数未使用

RxMessage，接收的数据和帧描述信息，CanRxMsg定义如下：

```
typedef struct
{
  uint32_t StdId;  /*!< 标准帧ID，取值范围[0,7FF].*/

  uint32_t ExtId;  /*!< 扩展帧ID，取值范围
[0,0x1FFFFFFF].*/
  uint8_t IDE;     /*!< 帧类型，标准帧 CAN_Id_Standard
                              扩展帧 CAN_Id_Extended
*/
  uint8_t RTR;     /*!< 传输消息类型   数据 CAN_RTR_Data
                                    远程帧
CAN_RTR_Remote*/
  uint8_t DLC;     /*!< 该帧所携带的数据的长度，取值范围
[0,8] */
  uint8_t Data[8]; /*!< 数据负载 */
  uint8_t FMI;     /*!< 该参数未使用*/
} CanRxMsg;
```

返回值：发送状态 `CAN_TxStatus_Ok` 发送成功

CAN接收中断

```
void CAN1_IRQHandler(void)
```

描述：CAN 接收中断处理函数，该函数定义在`hhd32f10x_it.c`

库的配置参数

库的运行相关参数在`HHD1705_lib.h`中定义，包括可选的系统主频，是否启用外设，和功能,以及导出一部分用户可能使用到的API接口。通过修改在文件中的宏，可以轻松编译功能不同库。其中，关于宏的启用和关闭有如下约定：

首字母为x的宏，表示未定义该宏

例如：`#define xCOMPILE_TO_LIB` 等价于 `#undef xCOMPILE_TO_LIB`

```
/*****************************************************
*****************************************************
  * Company: Hiwafer Technology Co., Ltd.
  *****************************************************
*****************************************************
  * 文件名称: HHD1705_lib.h
  * 功能说明:
  * 版    本: V1.0
  * 作    者: EnderWigg
  * 日    期: 2019.3.12
```

```
  *
  * 该文件用于配置库文件相关功能
  * 约定："x" 标记表示取消该宏的定义，例如 不启用CAN1，xCFG_USING_CAN1
**************************************************************************************************/
#ifndef __HHD1705_LIB_H__
#define __HHD1705_LIB_H__
#include "hhd32f10x_conf.h"


#define COMPILE_TO_LIB      /* 如果需要将该工程编译成库，必须使能该宏
                               如果编译为直接烧写，则不需要定义该宏*/

#define MII_MODE           /* MII mode   */

#define xBUG_GMII_TO_SGMII  /* 是否监视交换机状态，不是必须要启用*/
#define ETH_100M            /* 确定网口速度，如果定义该宏，网口速度将被配置为100M，否则为10M*/
#define CFG_SYS_60MHZ       /* 指定系统主频 60MHz*/
#define xCFG_USING_MARK     /* 获取 Mark 作为 IP，如果 FPGA 没有实现该功能，则会导致 MCU 启动不成功*/

#define CFG_USING_LED_BLINK /* 是否闪烁LED状态灯*/
#define CFG_USING_CAN1      /* 使用CAN1接口*/
#define CFG_USING_NET       /* 使用网络*/
#define CFG_USING_SPI       /* 使用SPI接口*/
// SPI 引脚选择
#define xCFG_SELECT_SPI_IO_0    /* 使用 PC9，PC10，PC11，PC12*/
#define CFG_SELECT_SPI_IO_1     /* 使用 PC7，PC8，PC10，PC11，该模式支持访问4片FPGA*/

// 网络路径选择
#define ETH_PATH  EN_TO_MDI /* 使用调试板网口*/
//#define ETH_PATH  EN_TO_SWITCH /* 使用交换机*/
// 设备默认IP
#define IP_0            192
#define IP_1            168
#define IP_2            2

// FPGA 固化功能 默认选择的FPGA

#ifdef CFG_SELECT_SPI_IO_1
    #define FPGA1           1
```

```
    #define FPGA2           2
    #define FPGA3           3
    #define FPGA4           4
#else
#define FPGA1           1
#define FPGA2           1
#define FPGA3           1
#define FPGA4           1
#endif


#define DEF_ACCESS_FPGA FPGA0

/*****************************************依赖条件检查
************************************************/



#ifdef CFG_USING_NET
    #ifdef CFG_USING_SPI
        #define CFG_USING_LOAD          // 使用FPGA远程固化网络服务器,  (该功能需要依赖 网络和SPI)
    #endif
    #define CFG_USING_XVC               // 使用FPGA远程调试网络服务器
#endif


#ifdef CFG_USING_CAN1
    #define CFG_USING_CAN1_WRITE_BACK   //MCU将通过CAN1回显收到的数据
#endif

////////////////////////////////////////////////Global
//////////////////////////////////////////////////////////

extern uint8_t Mark;

//////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                  API
//
```

```
83  ////////////////////////////////////////////////////////////////////////////////
    ///////
84
85
86  /*************************************************************************
    *******
87  *
88  *系统基础功能初始化
89  *
90  *************************************************************************
    *******/
91  void base_init(void);
92  /*************************************************************************
    *******
93  *
94  * 应用任务初始化
95  * 该函数初始化如下功能:
96  * 启动 FPGA远程调试功能
97  * 启动 FPGA快速固化功能
98  *************************************************************************
    *******/
99  void application_init(void);
100
101 /*************************************************************************
    *******
102 *
103 *   基于SysTick的延时
104 *
105 *************************************************************************
    ******/
106 void Delay(uint32_t nCount);
107
108 /*************************************************************************
    *******
109 *
110 *   获取SysTick的当前值
111 *
112 *************************************************************************
    ******/
```

```
113
114   uint32_t Get_Tick(void);
115 /*************************************************
    ************************************************
    *******
116 *
117 *   获取板卡Mark，并将该值作为 IP的最后一字节
118 *
119 *************************************************
    ************************************************
    ******/
120 uint8_t get_mark(void);
121
122
123 ////////////////////////////////////////////CAN//
    ////////////////////////////////////////////////
    ///////
124
125 /*************************************************
    ************************************************
    *******
126 *
127 *CAN 接口初始化
128 *
129 *************************************************
    ************************************************
    *******/
130 void can_init(HHD32F1_CAN_TypeDef *can, EN_BAUD
    baud, uint32_t filterID, uint32_t mask);
131
132 /*************************************************
    ************************************************
    *******
133 *
134 * 通过can 接口发送一帧数据
135 *
136 *************************************************
    ************************************************
    *******/
137 int CAN_Transmit(HHD32F1_CAN_TypeDef *can, CanTxMsg
    *TxMessage);
138
139 /*************************************************
    ************************************************
    *******
140 *
141 * 通过can 接口接收一帧数据
142 *
```

```
143 * 其中 FIFONumber 参数未使用
144 ********************************************************************************/
145 int CAN_Receive(HHD32F1_CAN_TypeDef *can, uint8_t FIFONumber, CanRxMsg* RxMessage);
146
147
148
149 ////////////////////////////////////////////SPI////////////////////////////////////////////////
150
151 /*******************************************************************************
152 *
153 *   初始化 SPI接口
154 *
155 *******************************************************************************/
156 void spi_Init(void);
157
158
159 /*******************************************************************************
160 *
161 *   写数据到FPGA
162 *
163 *******************************************************************************/
164 int SPI_To_FPGA_Wirte(uint8_t fpga, uint8_t addr, uint8_t *data, int len);
165
166 /*******************************************************************************
167 *
168 *   连续从FPGA读回数据
169 *
170 *******************************************************************************/
171 int SPI_To_FPGA_Read(uint8_t fpga, uint8_t addr, uint8_t *data, int len);
```

```
172
173
174 #endif
```

特别说明

```
1 #define xCFG_USING_MARK
```

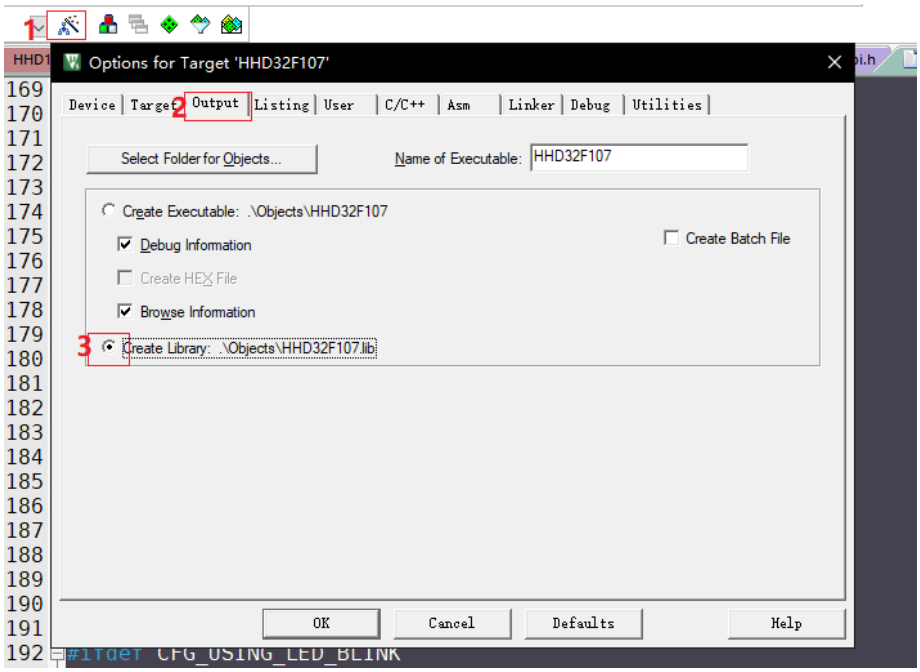
如果使能该宏，MCU 启动时会等待FPAG启动完成后从FPGA获取 Mark ，将其作为 网络 IP 的最后一字节，如果 FPGA 不支持该功能，则会导致 MCU 一直处于等待 FPGA 启动完成状态，及启动不成功，慎重使用。

编译

编译成库

1、在HHD1705_lib.h中，确保 `COMPILE_TO_LIB` 被定义

2、如下图操作，选择编译成库



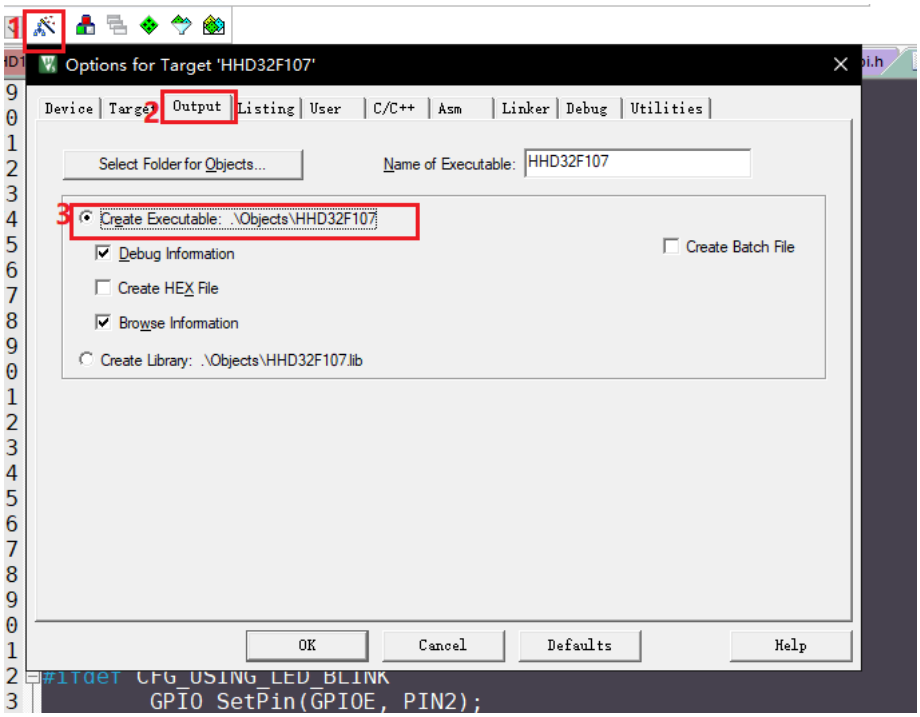
3、使整个工程全部编译

4、生成的文件在`\Project\Objects`,文件名为：**HHD32F107.lib**

编译成直接烧写文件

1、在HHD1705_lib.h中，确保 `COMPILE_TO_LIB` **没有**被定义

2、如下图操作，选择编译成直接烧写文件：



调试网口与背板交换机的切换

`HHD1705_Full_Lib` 工程支持两种网络路径：

- 调试网口
- 背板交换机

两种历经通过一个函数参数来控制，函数定义在如下位置：

文件路径： HHD1705_Full_Lib\User\main.c

在函数 `void peripheral_init(void)` 中

```
35
36  // 网络路径选择
37  #define ETH_PATH  EN_TO_MDI /* 使用调试板网口*/
38  //#define ETH_PATH  EN_TO_SWITCH /* 使用交换机*/
```

函数

```
1  Ethernet_Configuration(EN_TO_SWITCH);
```

当参数为 **EN_TO_MDI** 时 使用调试网口

当参数为 **EN_TO_SWITCH** 时 使用背板交换机

RTT nano 内核

在HHD1705_Full_Lib_V2.0 中引入 RTT nano 内核，提供操作系统的基本功能，包括多任务，信号量，内存管理，任务调度管理等实用功能，用户可以使用内核提的相关机制，实现多任务。内核API使用请参考 《RT-Thread 编程指南.pdf》

Demo 工程中使用 RTT nano 的多任务机制实现了 LED 闪烁功能，简单演示了如何创建任务。HHD1705_Full_Lib_V2.0 中，main函数已经变成了一个初始化任务，改任务在完成初始化后是可以返回的。

BSD 网络 API

HHD1705_Full_Lib_V2.0 中网络不但能够使用LWIP 原始 API 接口创建以太网连接，还能使用标准的 BSD 风格的 API 接口创建以太网连接。提供如下API：

```
int socket(int domain, int type, int protocol);
int bind(int s, const struct sockaddr *name, socklen_t namelen);
int listen(int s, int backlog);
int accept(int s, struct sockaddr *addr, socklen_t *addrlen);
int connect(int s, const struct sockaddr *name, socklen_t namelen);
int send(int s, const void *dataptr, size_t size, int flags);
int recv(int s, void *mem, size_t len, int flags);
int sendto(int s, const void *dataptr, size_t size, int flags, const struct sockaddr
*to, socklen_t tolen);
int recvfrom(int s, void *mem, size_t len, int flags, struct sockaddr *from,
socklen_t *fromlen);
int closesocket(int s);
int setsockopt(int s, int level, int optname, const void *optval, socklen_t optlen);
```

各个 API 的详细使用 请阅读《RT-Thread 编程指南.pdf》的 **25.3 BSD Socket API 介绍** 。

在 `User\load_server_BSD.c` 中创建了一个用于实现 FPAG 配置文件固化的 TCP 服务器，用户可参考该文件来创建自己的网络应用。

HHD1705_Full_Lib_V2.0 工程变化说明

1. HHD1705_Full_Lib_V2.0 中，不再单独创建 Demo 工程，Demo 工程和 Lib 在同一目录下，通过不同的工程入口文件打开不同的工程；
2. 同时由于使用 RTT nano 内核，工程不再支持 Keil4，需要使用Keil5 来打开和编译工程；

3. 对于不必用户修该Lib工程的源文件进行了只读设置，防止用户误操作导致库代码被修改；
4. Lib 文件放置于 `HHD1705_Full_Lib\Project\Objects`路径，及用户在重新编译库生成 Lib 文件后，不需要再次复制到 Demo工程。