|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  | 文档编号： | AR8020-SF-PD-V-00-03 |

AR8020 软件编程指南

|  |
| --- |
| 编写：Artosyn Software Team |
| 公司：Artosyn |
| 版本：Ver 0.3 |
| 日期：2017.02.10 |

目录

[1. AR8020 软件开发包（SDK）说明 2](#_Toc474508101)

[2. AR8020软件系统框架 3](#_Toc474508102)

[3. AR8020开发环境建立 3](#_Toc474508103)

[3.1. Host PC准备 3](#_Toc474508104)

[3.2. 编译说明 4](#_Toc474508105)

[3.2.1. 文件说明 4](#_Toc474508106)

[3.2.2. 编译流程 4](#_Toc474508107)

[4. Debug方法 5](#_Toc474508108)

[4.1.1. 串口Console 5](#_Toc474508109)

[4.1.2. GDB（单核CPU） 6](#_Toc474508110)

[4.1.3. DS5（单核CPU） 9](#_Toc474508111)

[5. Image升级 9](#_Toc474508112)

[5.1.1. 烧录器 9](#_Toc474508113)

[5.2. 进入Bootload bl2阶段升级app或bl2 11](#_Toc474508114)

[5.2.1.1. 进入app升级app 12](#_Toc474508115)

[6. AR8020各个模块的说明与使用 13](#_Toc474508116)

[6.1. libcommon.a 13](#_Toc474508117)

[6.1.1. 通用API 13](#_Toc474508118)

[6.1.2. AR8020定制的基础API 14](#_Toc474508119)

[6.2. libhal.a 22](#_Toc474508120)

[6.2.1. I2C 22](#_Toc474508121)

[6.2.2. UART 25](#_Toc474508122)

[6.2.3. GPIO 27](#_Toc474508123)

[6.2.4. TIMER/PWD 31](#_Toc474508124)

[6.2.5. SPI Master 39](#_Toc474508125)

[6.2.6. Baseband 41](#_Toc474508126)

[6.2.7. System Controller 54](#_Toc474508127)

[6.2.8. Cortex interrupt HAL 57](#_Toc474508128)

[6.2.9. USB\_HOST 66](#_Toc474508129)

[6.2.10. USB\_DEVICE 69](#_Toc474508130)

[6.2.11. USB\_OTG 70](#_Toc474508131)

[6.2.12. SD 71](#_Toc474508132)

[6.2.13. CAN 78](#_Toc474508133)

[6.2.14. ADC 81](#_Toc474508134)

[6.2.15. DMA 82](#_Toc474508135)

[6.3. libfreertos.a 83](#_Toc474508136)

[附录1：函数索引 83](#_Toc474508137)

[附录2：数据结构说明 86](#_Toc474508138)

# AR8020 软件开发包（SDK）说明

本文档主要对AR8020项目的以下几方面进行了说明：

1. PC机上开发环境的搭建。
2. 软件调试方法。
3. 软件升级方法。
4. 包含的库，并按照模块划分，分节对库中每个函数的使用进行详细说明。
5. 在附录中，对库中用到的数据结构进行统一说明。

本文档适合想了解AR8020项目，以及对AR8020项目进行二次开发的人员。文档修改记录如下：

表格 1 文档修改记录

|  |  |  |
| --- | --- | --- |
| 描述 | 版本 | 日期 |
| 创建文档 | 0.1 | 2016-12-05 |
| 细化部分内容 | 0.2 | 2016-12-29 |
| 1、修改Debug、USB、Baseband部分内容。  2、增加USB\_OTG、CAN、ADC、DMA章节 | 0.3 | 2017-02-10 |

# AR8020软件系统框架



图表 1 AR8020软件系统框架

# AR8020开发环境建立

## Host PC准备

1. Host PC 的OS可以选择 64位Redhat或者Ubuntu。推荐版本Red Hat Enterprise Linux Server release 6.8 （Santiago）; Ubuntu 14.04 LTS。
2. Toolchain使用gcc-arm-none-eabi-5\_2-2015q4， 可以从链接<https://launchpad.net/gcc-arm-embedded/5.0/5-2015-q4-major/+download/gcc-arm-none-eabi-5_2-2015q4-20151219-linux.tar.bz2> 获得。
3. 将“gcc-arm-none-eabi-5\_2-2015q4-20151219-linux.tar.bz2”解压到“/opt/toolchain”目录。
4. 若想将编译工具gcc-arm-none-eabi-5\_2-2015q4放在其他地方，可修改配置文件AR8020-XX/Build/config.mk中的CROSS\_COMPILE和CROSS\_COMPILE\_LIB\_PATH，将其修改为gcc-arm-none-eabi-5\_2-2015q4所在路径。

|  |
| --- |
| CROSS\_COMPILE = /myPath/gcc-arm-none-eabi-5\_2-2015q4/  CROSS\_COMPILE\_LIB\_PATH = /myPath/Desktop/gcc-arm-none-eabi-5\_2-2015q4/ |

## 编译说明

### 文件说明

SDK发布包括一个SDK压缩包和一个md5文件（例如AR8020-SDK-2016-12-05.tar.gz和AR8020-SDK-2016-12-05.md5）。

开始使用前请将SDK压缩包解压到工作目录，其中包括Application、Build、Configure、Document、Oputput、Utility。

1. Application是应用层代码，用户定制的代码主要都属于这部分内容。
2. Build是Makefile的模板和配置文件。
3. Configure是系统配置相关的头文件。
4. Document是版本发布说明等文档。
5. Output内包含有应用层代码所需的头文件和库文件，分别在Staging/Inc目录和Staging/Lib目录。Staging/Lib目录中有libAR8020Drv.a、libhal.a、libboard.a、libcommon.a和libfreertos.a。
6. Utility内包含编译过程和image文件打包过程中需要用到的一些工具软件。

### 编译流程

具体编译流程如下：

1. 进入myApp目录：

|  |
| --- |
| $ cd AR8020-XX/Application/ myApp/ |

1. 清除之前编译结果：

|  |
| --- |
| $ make clean |

1. 编译天空端程序：

|  |
| --- |
| $ make sky |

1. 编译地面端程序：

|  |
| --- |
| $ make ground |

其他说明：

1. 第3）步或第4）步编译之后，在当前目录生成3个bin文件app.bin、ar8020.bin、boot.bin，其中：

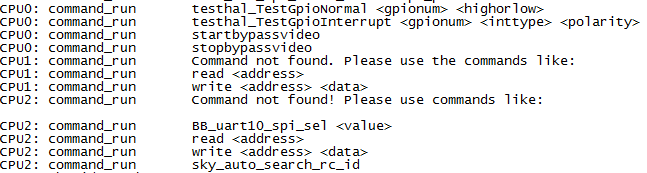
* ar8020.bin是一个包含bootload和应用程序的完整二进制文件，可用于烧录器烧录；
* boot.bin是只包含bootload的二进制文件；
* app.bin是只包含应用程序的二进制文件。

1. 若只更新天空端程序，只需执行1）、2）、3）步骤。
2. 若只更新地面端程序，只需执行1）、2）、4）步骤。
3. 若天空端、地面端程序需要同时更新，第3）步编译完天空端程序之后，需要将所需bin文件取走，再进行第4）步，否则bin文件的内容会被第4）步的编译所覆盖。

# Debug方法

### 串口Console

目前三颗CPU是共用uart0进行output和input来进行debug。可以根据实际的需要在Application里面开发测试用例。“CPU0：”表示是CPU0打印的信息，“CPU1：”表示是CPU1打印的信息，“CPU2：”表示是CPU1打印的信息，如下图所示。



Degug APIs的详细使用信息参考“6.1.2.1 Debug Log APIs”章节。

### GDB（单核CPU）

1. 烧录.bin到flash，使芯片能够正常启动。
2. 打开JLinkGDB server，并敲入以下命令：

|  |
| --- |
| ./JLinkGDBServer -select USB -device Cortex-M7 -if JTAG -speed 4000 -scriptfile CustomM7.JLinkScript |

通过修改 CustomM7.JLinkScript可以实现debug某一颗CPU，具体参见本章节PS部分的内容。

1. 开启GDB工具。目前我们工程主要用~/gcc-arm-none-eabi-5\_2-2015q4/bin/arm-none-eabi-gcc编译，则需要使用arm-none-eabi-gdb作为gdb工具，敲入如下命令：

|  |
| --- |
| ./arm-none-eabi-gdb |

按照相应要求，输入命令：

|  |
| --- |
| GNU gdb (GNU Tools for ARM Embedded Processors) 7.10.1.20151217-cvs  Copyright (C) 2015 Free Software Foundation, Inc.  License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>  This is free software: you are free to change and redistribute it.  There is NO WARRANTY, to the extent permitted by law. Type "show copying"  and "show warranty" for details.  This GDB was configured as "--host=i686-linux-gnu --target=arm-none-eabi".  Type "show configuration" for configuration details.  For bug reporting instructions, please see:  <http://www.gnu.org/software/gdb/bugs/>.  Find the GDB manual and other documentation resources online at:  <http://www.gnu.org/software/gdb/documentation/>.  For help, type "help".  Type "apropos word" to search for commands related to "word".  (gdb) target remote localhost:2331(连接GDB server)  (gdb) file ./ar8020\_cpu0.elf （加载elf,其中包含符号表等） |

此时可以开启debug。

PS：

1. 采用JlinkExe直连AR8020的命令为：

|  |
| --- |
| ./JLinkExe -device Cortex-M7 -if JTAG -speed 4000 -AutoConnect 1 -JLinkScriptFile CustomM7.JLinkScript |

1. CustomM7.JLinkScript脚本为：

|  |
| --- |
| /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* SEGGER MICROCONTROLLER GmbH & Co. K.G. \*  \* Solutions for real time microcontroller applications \*  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* \*  \* (c) 2011-2015 SEGGER Microcontroller GmbH & Co. KG \*  \* \*  \* Internet: www.segger.com Support: support@segger.com \*  \* \*  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  ----------------------------------------------------------------------  Purpose :  ---------------------------END-OF-HEADER------------------------------  \*/  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*  \* ResetTarget  \*/  void ResetTarget(void) {  // In case cores 1-3 are reset, we do nothing,  // as we would lose connection to these cores, when resetting the device  // as a reset disables the clock to them.  }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*  \* ConfigTargetSettings  \*  \* Function description  \* Called before InitTarget(). Maninly used to set some global DLL variables  \* to customize the normal connect procedure. For ARM CoreSight devices this may be specifying  \* the base address of some CoreSight components (ETM, ...) that cannot be auto-detected  \* by J-Link due to erroneous ROM tables etc. May also be used to specify the device name in case debugger does not pass it to the DLL.  \*  \* Notes  \* (1) May not, under absolutely NO circumstances, call any API functions that perform target communication.  \* (2) Should only set some global DLL variables  \*/  void ConfigTargetSettings(void) {  Report("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  Report("J-Link script: Cortex-M7 core J-Link script");  Report("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  JTAG\_AllowTAPReset = 1; // J-Link is allowed to use a TAP reset for JTAG-chain auto-detection  CORESIGHT\_AddAP(0, CORESIGHT\_APB\_AP);  CORESIGHT\_AddAP(1, CORESIGHT\_AHB\_AP); // AHB-AP which controls 1st Cortex-M7  CORESIGHT\_AddAP(2, CORESIGHT\_AHB\_AP); // AHB-AP which controls 2nd Cortex-M7  CORESIGHT\_AddAP(3, CORESIGHT\_AHB\_AP); // AHB-AP which controls 3rd Cortex-M7  CPU = CORTEX\_M7;  CORESIGHT\_IndexAHBAPToUse = 1; // Select 1st M7  } |

需要注意的是：修改 CORESIGHT\_IndexAHBAPToUse参数可以指定debug某一颗CPU，其中 CORESIGHT\_IndexAHBAPToUse=1/2/3分别对应CPU0/1/2。使用时，直接修改该参数就好。

### DS5（单核CPU）

参考ARM官网的《ARM® DS-5 Debugger User Guide》，链接地址为：<http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0446z/index.html>

# Image升级

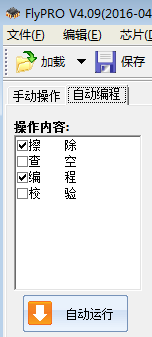
### 烧录器

通过FLASH烧录器，将bin文件ar8020.bin完整的写入到FLASH中。下边以FlyPRO烧录软件为例，进行具体操作说明：

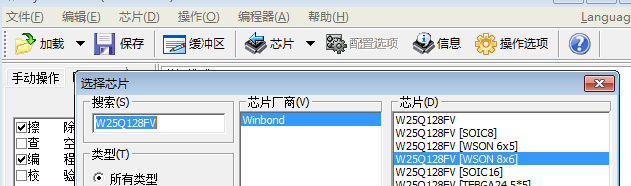
1. 确认烧录软件已经与硬件连接，软件右下角“联机模式”灯为绿色，未连接好时，灯为灰色。



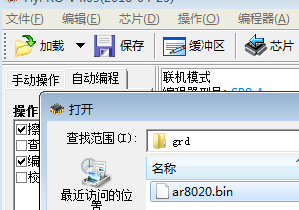
1. 点击“自动编程”。



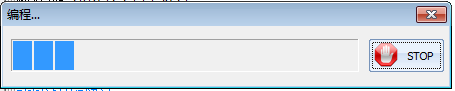
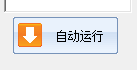
1. 在菜单栏“芯片”中选择具体芯片型号。



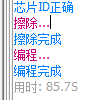
1. 在菜单栏“加载”中选择要烧录的文件，双击要烧录的文件。



1. 点击“自动运行”之后，出现烧录进度条。



1. 出现“编程完成”信息时，表示FLASH烧录已全部完成。



1. 将刚烧录好的FLASH芯片，放入板子，重新上电运行，新升级的程序即开始运行。

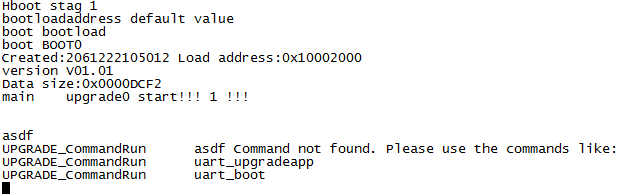
### 进入Bootload bl2阶段升级app或bl2



1. 参照“5.2.1.1 进入app升级app”章节，连接、设置串口。
2. 上电前按住‘t’键，等待板子打印boot bootload，如下图所示，再松开‘t’键。



1. 在串口0上输入任何字符，出现下面界面：



* 命令uart\_upgradeapp,从uart0升级app

1. 输入uart\_upgradeapp
2. 使用串口工具（能发送文件），发送app.bin文件
3. 串口助手出现upgrade finish 说明升级成功。

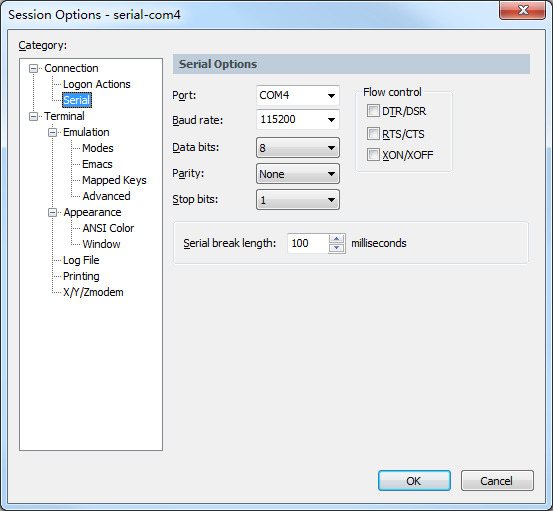
* 命令uart\_boot,从uart0升级bl2(升级模式程序)

1. 输入uart\_boot
2. 使用串口工具（能发送文件），发送boot.bin文件
3. 串口助手出现upgrade finish 说明升级成功。

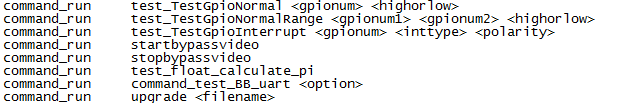
注：编译出来的ar8020\_boot.bin文件大小不能超过60K。

### 进入app升级app

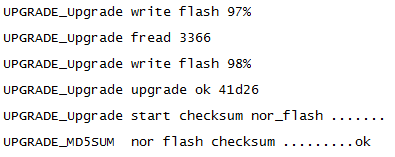
1. 准备一个U盘，将其格式化为FAT32格式，将需要升级的app.bin文件（分为天空端和地面端，参考“3.2.2编译流程”）拷贝到U盘根目录下，并将U盘接到板子的USB1上。
2. 通过串口线将板子上的串口0（TTL电平）与PC机连接，在PC机上使用SecureCRT软件（此处使用的版本为：Version 7.1.1）的Serial功能与板子进行通讯。在SecureCRT的菜单栏中，依次点击“Options -> Session Options -> Serial”进行参数配置，具体配置参数如下（Port需要根据实际情况做修改）：



1. 在SecureCRT输入窗口中，输入任意字符并按回车键，出现如下打印信息，表示SecureCRT与板子通讯正常。



1. 敲入命令：upgrade app.bin，板子开始打印升级过程的进度信息，如下图所示，当出现“UPGRADE\_MD5SUM nor flash checksum .........ok”时，表示升级完成。



1. 将板子断电，再重新上电运行，新升级的程序即开始运行。

# AR8020各个模块的说明与使用

## libcommon.a

libcommon.a是一个类似于libc的基础库，包括string、memory、printf等通用libc的API，同时还包含一些AR8020定制的debug log和system event等基础API。

### 通用API

通过“#include <stdlib.h>”、“#include <string.h>”、“#include <stdio.h>”获得标准API的声明即可。

表格 2 基础库API列表

|  |  |
| --- | --- |
| 编号 | 函数名称 |
|  | int sprintf(char \*buf, const char \*fmt, ...) |
|  | int printf(const char \*fmt, ...) |
|  | void \*memcpy(void \*\_\_dest, \_\_const void \*\_\_src, size\_t \_\_n) |
|  | void \*memmove(void \*\_\_dest, \_\_const void \*\_\_src, size\_t count) |
|  | size\_t strlen(const char \*s) |
|  | size\_t strnlen(const char \*s, size\_t maxlen) |
|  | int memcmp(const void \*cs, const void \*ct, size\_t count) |
|  | int strcmp(const char \*cs, const char \*ct) |
|  | void \*memchr(const void \*s, int c, size\_t count) |
|  | char \*strchr(const char \*s, int c) |
|  | void \*memset(void \*s, int c, size\_t count) |
|  | void \_\_memzero(void \*s, size\_t count) |
|  | unsigned long strtoul(const char \*cp, char \*\*endp, unsigned int base) |
|  | long strtol(const char \*cp, char \*\*endp, unsigned int base) |
|  | int atoi(const char\* nptr) |
|  | \_\_attribute\_\_((weak)) void \*malloc(size\_t size) |
|  | \_\_attribute\_\_((weak)) void free(void \*ap) |

### AR8020定制的基础API

#### Debug Log APIs

使用该部分API函数，需要包含头文件"debuglog.h"。

##### Dlog\_Init

* 完整名称：

|  |
| --- |
| void Dlog\_Init(unsigned int index) |

* 功能说明：

初始化debug log缓存。

* 参数说明：

index debug log 缓存号

CPU0使用0，CPU1使用1，CPU2使用2。

* 数据结构说明：

无。

* 返回值：

1. 无。

* 注意事项：

1. 在系统第一次debug log打印之前必须进行debug log缓存初始化。

* 使用例子：

在CPU0的应用程序中初始化CPU0的debug log缓存。

|  |
| --- |
| Dlog\_Init(0); |

##### Dlog\_Info

* 完整名称：

|  |
| --- |
| Dlog\_Info(fmt, arg...) |

* 功能说明：

输出debug level大于或等于LOG\_LEVEL\_INFO的log内容到debug缓存。

* 参数说明：

fmt log输出格式定义

arg... log输出数据排列

* 数据结构说明：

无。

* 返回值：

1. 无。

* 注意事项：

1. 这里只是输出到debug log的缓冲区，所以可以在中断处理函数中使用，基本不会带来延时。

* 使用例子：

使用方法类似于printf。

输出字符串“test string %d”到debug log缓存。

|  |
| --- |
| int val = 100;  Dlog\_Info(“test string %d”, val); |

##### Dlog\_Warning

* 完整名称：

|  |
| --- |
| Dlog\_Warning(fmt, arg...) |

* 功能说明：

输出debug level大于或等于LOG\_LEVEL\_WARNING的log内容到debug缓存。

* 参数说明：

fmt log输出格式定义

arg... log输出数据排列

* 数据结构说明：

无。

* 返回值：

1. 无。

* 注意事项：

1. 这里只是输出到debug log的缓冲区，所以可以在中断处理函数中使用，基本不会带来延时。

* 使用例子：

输出字符串“test string %d”到debug log缓存。

|  |
| --- |
| int val = 100;  Dlog\_Warning(“test string %d”, val); |

##### Dlog\_Error

* 完整名称：

|  |
| --- |
| Dlog\_Error(fmt, arg...) |

* 功能说明：

输出debug level大于或等于LOG\_LEVEL\_ERROR的log内容到debug缓存。

* 参数说明：

fmt log输出格式定义

arg... log输出数据排列

* 数据结构说明：

无。

* 返回值：

1. 无。

* 注意事项：

1. 这里只是输出到debug log的缓冲区，所以可以在中断处理函数中使用，基本不会带来延时。

* 使用例子：

使用方法类似于printf。

输出字符串“test string %d”到debug log缓存。

|  |
| --- |
| int val = 100;  Dlog\_Error(“test string %d”, val); |

##### Dlog\_Critical

* 完整名称：

|  |
| --- |
| Dlog\_Critical(fmt, arg...) |

* 功能说明：

输出debug level大于或等于LOG\_LEVEL\_CRITICAL的log内容到debug缓存。

* 参数说明：

fmt log输出格式定义

arg... log输出数据排列

* 数据结构说明：

无。

* 返回值：

1. 无。

* 注意事项：

1. 这里只是输出到debug log的缓冲区，所以可以在中断处理函数中使用，基本不会带来延时。

* 使用例子：

使用方法类似于printf。

输出字符串“test string %d”到debug log缓存。

|  |
| --- |
| int val = 100;  Dlog\_Critical(“test string %d”, val); |

##### Dlog\_Output

* 完整名称：

|  |
| --- |
| unsigned int Dlog\_Output(unsigned int byte\_num) |

* 功能说明：

从debug log缓存输出debug log信息到console端口。

* 参数说明：

byte\_num 从debug log缓存输出的最大字符个数。

* 数据结构说明：

无。

* 返回值：

1. 实际往console端口输出的字符个数。

* 注意事项：

1. 这里是输出到console端口，默认是输出到UART端口，这个过程会产生一定的延时。

* 使用例子：

输出100个字符到console。

|  |
| --- |
| Dlog\_Output(100); |

#### System Event APIs

使用该部分API函数，需要包含头文件"sys\_event.h"。

##### SYS\_EVENT\_RegisterHandler

* 完整名称：

|  |
| --- |
| uint8\_t SYS\_EVENT\_RegisterHandler(uint32\_t event\_id,  SYS\_Event\_Handler event\_handler) |

* 功能说明：

注册一个事件处理函数。

* 参数说明：

1. event\_id： 事件的ID。
2. event\_handler：事件的处理函数typedef void (\*SYS\_Event\_Handler)(void \*)。

* 数据结构说明：

无。

* 返回值：

1. 1：注册事件处理函数成功。
2. 0：注册事件处理函数失败。

* 注意事项：

无

* 使用例子：

注册一个事件ID为0x20000005的处理函数。

|  |
| --- |
| void test\_handler(void\*)  {  Dlog\_Info(“test\_handler is working.”);  }  if (SYS\_EVENT\_RegisterHandler(0x20000005, test\_handler) == 1)  {  Dlog\_Info(“test\_handler registeration succeed”);  }  else  {  Dlog\_Error(“test\_handler registeration fail”)  } |

##### SYS\_EVENT\_UnRegisterHandler

* 完整名称：

|  |
| --- |
| uint8\_t SYS\_EVENT\_UnRegisterHandler(uint32\_t event\_id,  SYS\_Event\_Handler event\_handler) |

* 功能说明：

注销一个事件处理函数。

* 参数说明：

1. event\_id： 事件的ID。
2. event\_handler：事件的处理函数typedef void (\*SYS\_Event\_Handler)(void \*)。

* 数据结构说明：

无。

* 返回值：

1. 1：注册事件处理函数成功。
2. 0：注册事件处理函数失败。

* 注意事项：

无

* 使用例子：

注销一个事件ID为0x20000005的处理函数。

|  |
| --- |
| if (SYS\_EVENT\_UnRegisterHandler (0x20000005, test\_handler) == 1)  {  Dlog\_Info(“test\_handler unregister succeed”);  }  else  {  Dlog\_Error(“test\_handler unregister fail”)  } |

##### SYS\_EVENT\_Notify

* 完整名称：

|  |
| --- |
| uint8\_t SYS\_EVENT\_Notify(uint32\_t event\_id, void\* parameter) |

* 功能说明：

发布某事件及其事件参数。

* 参数说明：

1. event\_id： 事件的ID。
2. parameter：事件参数，作为处理函数typedef void ( \* SYS\_Event\_Handler ) (void \*)的入参。

* 数据结构说明：

无。

* 返回值：

1. 1：事件发布成功。
2. 0：事件发布失败。

* 注意事项：

无

* 使用例子：

发布一个事件ID为0x20000005的事件，事件参数为字符串“parameter”。

|  |
| --- |
| char \*p = “parameter”;  if (SYS\_EVENT\_Notify(0x20000005, (void\*)p) == 1)  {  Dlog\_Info(“event 0x20000005 notification succeed”);  }  else  {  Dlog\_Error(“event 0x20000005 notification fail”);  } |

##### SYS\_EVENT\_Process

* 完整名称：

|  |
| --- |
| uint8\_t SYS\_EVENT\_Process(void) |

* 功能说明：

System Event机制的engine函数。

* 参数说明：

1. 无。

* 数据结构说明：

无。

* 返回值：

1. 1：事件处理成功。
2. 0：事件处理失败。

* 注意事项：

无

* 使用例子：

在main loop里面调用SYS\_EVENT\_Process。

|  |
| --- |
| main(void)  {  ...  SYS\_EVENT\_Process();  ...  } |

## libhal.a

### I2C

使用该部分API函数，需要包含头文件"hal\_i2c.h"。

#### HAL\_I2C\_MasterInit

* 完整名称：

|  |
| --- |
| HAL\_RET\_T HAL\_I2C\_MasterInit(ENUM\_HAL\_I2C\_COMPONENT e\_i2cComponent,  uint16\_t u16\_i2cAddr,  ENUM\_HAL\_I2C\_SPEED e\_i2cSpeed) |

* 功能说明：

初始化I2C。

* 参数说明：

1. en\_component I2C模块序号，0-5。
2. u16\_i2cAddr 从器件地址。
3. en\_i2cSpeed Normal/Full/High。

* 数据结构说明：

参考“ENUM\_HAL\_I2C\_COMPONENT”、“ENUM\_HAL\_I2C\_SPEED”。

* 返回值：

1. HAL\_OK：初始化成功。
2. HAL\_I2C\_ERR\_INIT 初始失败。

* 注意事项：

1. 只应用于主机模式。
2. High mode要依赖系统并且电路的影响特别重要，I2C子器件要支持high mode。

* 使用例子：

设置I2C0，为fast mode。器件地址为器件地址右移1位。

|  |
| --- |
| HAL\_I2C\_MasterInit(HAL\_I2C\_COMPONENT\_0, 0x51, HAL\_I2C\_FAST\_SPEED); |

#### HAL\_I2C\_MasterWriteData

* 完整名称：

|  |
| --- |
| HAL\_RET\_T HAL\_I2C\_MasterWriteData(ENUM\_HAL\_I2C\_COMPONENT e\_i2cComponent,  uint16\_t u16\_i2cAddr,  uint8\_t \*pu8\_wrData,  uint32\_t u32\_wrSize) |

* 功能说明：

I2C写数据。

* 参数说明：

1. en\_component I2C模块序号，0-5。
2. u16\_i2cAddr 从器件地址。
3. e pu8\_wrData 数据指针。
4. u32\_wrSize 写多少个数据。

* 数据结构说明：

参考“ENUM\_HAL\_I2C\_COMPONENT”。

* 返回值：

1. HAL\_OK：I2C写成功。
2. HAL\_I2C\_ERR\_WRITE\_DATA 写失败。

* 注意事项：

1. 受I2Cfifo影响，size必须小于7。超过6个会有风险
2. High mode要依赖系统并且电路的影响特别重要，I2C子器件要支持high mode。

* 使用例子：

在I2C0接口，器件地址为器件地址右移1位。，写入地址为0x0000，写入数据为61, 62, 63, 64。

|  |
| --- |
| unsigned char data\_src1[6] = {0x00,0x00,61, 62, 63, 64};  HAL\_I2C\_MasterWriteData(HAL\_I2C\_COMPONENT\_0,0x51, data\_src1, 6); |

#### HAL\_I2C\_MasterReadData

* 完整名称：

|  |
| --- |
| HAL\_RET\_T HAL\_I2C\_MasterReadData(ENUM\_HAL\_I2C\_COMPONENT e\_i2cComponent,  uint16\_t u16\_i2cAddr,  uint8\_t \*pu8\_wrData,  uint8\_t u8\_wrSize,  uint8\_t \*pu8\_rdData,  uint32\_t u32\_rdSize) |

* 功能说明：

I2C读数据。

* 参数说明：

1. en\_component I2C模块序号，0-5。
2. u16\_i2cAddr 从器件地址。
3. e pu8\_wrData 读数据的起始地址。
4. u32\_wrSize 地址大小。
5. pu8\_rdData 读出数据放置的指针
6. u32\_rdSize 读多少个数据

* 数据结构说明：

参考“ENUM\_HAL\_I2C\_COMPONENT”。

* 返回值：

1. HAL\_OK：I2C读成功。
2. HAL\_I2C\_ERR\_READ\_DATA 读失败。

* 注意事项：

1. High mode要依赖系统并且电路的影响特别重要，I2C子器件要支持high mode。

* 使用例子：

从I2C0接口，器件地址为器件地址右移1位，地址为0x0处读取6个字节数据。

|  |
| --- |
| unsigned char data\_src1[6];  HAL\_I2C\_MasterReadData(HAL\_I2C\_COMPONENT\_0,0x51, 0, 2, data\_src1, 6); |

### UART

共有9个串口，串口默认配置不可更改（1位起始位 8位数据位 1位停止位 无校验）。

使用该部分API函数，需要包含头文件“hal\_uart.h”。

#### HAL\_UART\_Init

* 完整名称：

|  |
| --- |
| HAL\_RET\_T HAL\_UART\_Init(ENUM\_HAL\_UART\_COMPONENT e\_uartComponent,  ENUM\_HAL\_UART\_BAUDR e\_uartBaudr,  HAL\_UART\_RxHandle pfun\_rxFun) |

* 功能说明：

初始化串口。

* 参数说明：

1. e\_uartComponent：取值范围HAL\_UART\_COMPONENT\_0 ~ HAL\_UART\_COMPONENT\_8。
2. e\_uartBaudr：取值范围HAL\_UART\_BAUDR\_9600 ~ HAL\_UART\_BAUDR\_115200。
3. pfun\_rxFun：串口数据接收处理函数，函数结构如：uint32\_t (\*HAL\_UART\_RxHandle)(uint8\_t \*pu8\_rxBuf, uint8\_t u8\_len)。

* 数据结构说明：

参考“ENUM\_HAL\_UART\_COMPONENT”、“ENUM\_HAL\_UART\_BAUDR”。

* 返回值：

1. HAL\_OK：初始化成功。
2. HAL\_UART\_ERR\_INIT：初始化失败。

* 注意事项：

1. 串口0用于输入输出调试信息，不可用作其他用途。

* 使用例子：

将串口3初始化为9600波特率，开启中断。

|  |
| --- |
| HAL\_UART\_Init(HAL\_UART\_COMPONENT\_3,HAL\_UART\_BAUDR\_9600,myUartRxFunc);  HAL\_NVIC\_EnableIRQ(UART\_INTR3\_VECTOR\_NUM); |

#### HAL\_UART\_TxData

* 完整名称：

|  |
| --- |
| HAL\_RET\_T HAL\_UART\_TxData(ENUM\_HAL\_UART\_COMPONENT e\_uartComponent,  uint8\_t \*pu8\_txBuf,  uint32\_t u32\_len) |

* 功能说明：

从串口发送数据。

* 参数说明：

1. e\_uartComponent：取值范围HAL\_UART\_COMPONENT\_0 ~ HAL\_UART\_COMPONENT\_0。
2. pu8\_txBuf：需要发送数据缓存区首地址。
3. u32\_len：需要发送数据长度。

* 数据结构说明：

参考“ENUM\_HAL\_UART\_COMPONENT”。

* 返回值：

1. HAL\_OK：发送数据成功。
2. HAL\_UART\_ERR\_WRITE\_DATA：发送数据失败。

* 注意事项：

无

* 使用例子：

使用串口3发送数组u8\_data[]前20个字符。

|  |
| --- |
| HAL\_UART\_TxData(HAL\_UART\_COMPONENT\_3, &u8\_data[0], 20); |

#### UartCallBack

* 完整名称：

|  |
| --- |
| typedef uint32\_t (\*HAL\_UART\_RxHandle)(uint8\_t \*pu8\_rxBuf, uint8\_t u8\_len) |

* 功能说明：

在中断服务程序中被调用的函数，用于处理接收到的串口数据。

* 参数说明：

1. pu8\_rxBuf：待接收到数据存放地址。
2. u8\_len：待接收数据长度。

* 数据结构说明：

无。

* 返回值：

无意义。

* 注意事项：

1. 函数为用户自定函数，格式必须符合结构uint32\_t (\*HAL\_UART\_RxHandle)(uint8\_t \*pu8\_rxBuf, uint8\_t u8\_len)。
2. HAL\_UART\_RxHandle是在中断中被调用的，因此HAL\_UART\_RxHandle不可耗时太长，仅仅处理数据即可。

* 使用例子：

将串口3收到的数据存入用户自己的缓存区s\_u8\_uartRxBuf。

|  |
| --- |
| uint32\_t uartRxCallBack(uint8\_t \*pu8\_rxBuf, uint8\_t u8\_len)  {  if (u8\_len > 64)  {  u8\_len = 64;  }  memcpy(s\_u8\_uartRxBuf + s\_u8\_uartRxLen, pu8\_rxBuf, u8\_len);  s\_u8\_uartRxLen += u8\_len;  }  HAL\_UART\_Init(HAL\_UART\_COMPONENT\_3,HAL\_UART\_BAUDR\_9600, uartRxCallBack); |

### GPIO

一共128个GPIO。

使用该部分API函数，需要包含头文件"hal\_gpio.h"。

#### HAL\_GPIO\_OutPut

* 完整名称：

|  |
| --- |
| HAL\_RET\_T HAL\_GPIO\_OutPut(ENUM\_HAL\_GPIO\_NUM e\_gpioPin) |

* 功能说明：

设置gpio为output模式。

* 参数说明：

1. e\_gpioPin: 要设置的gpio，正确的为0—127。

* 数据结构说明：

参考“ENUM\_HAL\_GPIO\_NUM”。

* 返回值：

1. HAL\_OK 函数执行成功。
2. HAL\_GPIO\_ERR\_UNKNOWN 没有找到相应的gpio。

* 注意事项：

1. e\_gpioPin的值必须在在HAL\_GPIO\_NUM0- HAL\_GPIO\_NUM127.

* 使用例子：

设置第64个GPIO为output模式。

|  |
| --- |
| HAL\_GPIO\_OutPut(HAL\_GPIO\_NUM64); |

#### HAL\_GPIO\_InPut

* 完整名称：

|  |
| --- |
| HAL\_RET\_T HAL\_GPIO\_InPut(ENUM\_HAL\_GPIO\_NUM e\_gpioPin) |

* 功能说明：

设置gpio为input模式。

* 参数说明：

1. e\_gpioPin: 要设置的gpio，正确的为0—127。

* 数据结构说明：

参考“ENUM\_HAL\_GPIO\_NUM”。

* 返回值：

1. HAL\_OK 函数执行成功。
2. HAL\_GPIO\_ERR\_UNKNOWN 没有找到相应的gpio。

* 注意事项：

1. e\_gpioPin的值必须在在HAL\_GPIO\_NUM0- HAL\_GPIO\_NUM127.

* 使用例子：

设置第64个GPIO为input模式。

|  |
| --- |
| HAL\_GPIO\_InPut(HAL\_GPIO\_NUM64); |

#### HAL\_GPIO\_SetPin

* 完整名称：

|  |
| --- |
| HAL\_RET\_T HAL\_GPIO\_SetPin(ENUM\_HAL\_GPIO\_NUM e\_gpioPin,  ENUM\_HAL\_GPIO\_PinState e\_pinState) |

* 功能说明：

设置gpio为高电平或低电平。

* 参数说明：

1. e\_gpioPin: 要设置的gpio，正确的为0—127。
2. e\_pinState:

GPIO\_PIN\_RESET set gpio low

GPIO\_PIN\_SET set gpio high

* 数据结构说明：

参考“ENUM\_HAL\_GPIO\_NUM”、“ENUM\_HAL\_GPIO\_PinState”。

* 返回值：

1. HAL\_OK 函数执行成功。
2. HAL\_GPIO\_ERR\_UNKNOWN 没有找到相应的gpio。

* 注意事项：

1. e\_gpioPin的值必须在在HAL\_GPIO\_NUM0- HAL\_GPIO\_NUM127.
2. Gpio必须为output模式

* 使用例子：

设置第64个GPIO输出低电平。

|  |
| --- |
| HAL\_GPIO\_SetPin(HAL\_GPIO\_NUM64, GPIO\_PIN\_RESET); |

#### HAL\_GPIO\_GetPin

* 完整名称：

|  |
| --- |
| HAL\_RET\_T HAL\_GPIO\_GetPin(ENUM\_HAL\_GPIO\_NUM e\_gpioPin,  uint32\_t \*p\_retGpioState) |

* 功能说明：

获得gpio的状态。

* 参数说明：

1. e\_gpioPin: 要设置的gpio，正确的为0—127。
2. \*p\_retGpioState: 返回gpio状态的指针

* 数据结构说明：

参考“ENUM\_HAL\_GPIO\_NUM”。

* 返回值：

1. gpio state

0 means the gpio is low

1 means the gpio is high

HAL\_GPIO\_ERR\_UNKNOWN 没有找到相应的gpio

* 注意事项：

1. e\_gpioPin的值必须在在HAL\_GPIO\_NUM0- HAL\_GPIO\_NUM127.
2. Gpio必须为input模式

* 使用例子：

获取第64个GPIO输入状态。

|  |
| --- |
| uint32\_t u32\_gpioState;  HAL\_GPIO\_SetPin (HAL\_GPIO\_NUM64, & u32\_gpioState); |

#### HAL\_GPIO\_RegisterInterrupt

* 完整名称：

|  |
| --- |
| HAL\_RET\_T HAL\_GPIO\_RegisterInterrupt(ENUM\_HAL\_GPIO\_NUM e\_gpioPin,  ENUM\_HAL\_GPIO\_InterrputLevel e\_inttype,  ENUM\_HAL\_GPIO\_InterrputPolarity e\_polarity,  void \*fun\_callBack) |

* 功能说明：

设置gpio为中断模式。

* 参数说明：

1. e\_gpioPin: 要设置的gpio，正确的为0—127.
2. e\_inttype: GPIO\_LEVEL\_SENUMSITIVE 电平中断
3. GPIO\_EDGE\_SENUMSITIVE 沿中断
4. e\_polarity: GPIO\_ACTIVE\_LOW 低电平或下降沿
5. GPIO\_ACTIVE\_HIGH 高电平或上升沿
6. fun\_callBack：中断回调函数

* 数据结构说明：

参考“ENUM\_HAL\_GPIO\_NUM”、“ENUM\_HAL\_GPIO\_InterrputLevel”、“ENUM\_HAL\_GPIO\_InterrputPolarity”。

* 返回值：

1. HAL\_OK 函数执行成功
2. HAL\_GPIO\_ERR\_UNKNOWN 没有找到相应的gpio.

* 注意事项：

1. e\_gpioPin的值必须在在HAL\_GPIO\_NUM0- HAL\_GPIO\_NUM127.

* 使用例子：

获取第64个GPIO为中断模式,下降沿中，中断处理函数为GPIOhal\_IRQHandler。

|  |
| --- |
| HAL\_GPIO\_RegisterInterrupt(HAL\_GPIO\_NUM64,  GPIO\_EDGE\_SENUMSITIVE,  GPIO\_ACTIVE\_LOW,  GPIOhal\_IRQHandler); |

### TIMER/PWD

10个timer（0——9），8个为一组，组0包含第0—7timer，组1包含第8—10timer，

Timer和pwm冲突，如果第0个设置为定时器模式，则第0个pwm失效，以此类推。

使用该部分API函数，需要包含头文件"hal\_timer.h"。

#### HAL\_TIMER\_RegisterTimer

* 完整名称：

|  |
| --- |
| HAL\_RET\_T HAL\_TIMER\_RegisterTimer(ENUM\_HAL\_TIMER\_NUM e\_timerNum,  uint32\_t u32\_timeus,  void \*fun\_callBack) |

* 功能说明：

注册timer。

* 参数说明：

1. e\_timerNum: 要设置的tiemr，正确的为0—23。
2. u32\_timeus: 装载的微妙值。
3. \*fun\_callBack:中断回调函数。

* 数据结构说明：

参考“ENUM\_HAL\_TIMER\_NUM”。

* 返回值：

1. HAL\_OK 函数执行成功。
2. HAL\_TIMER\_ERR\_UNKNOWN 没有找到相应的timer。

* 注意事项：

1. e\_timerNum的值必须在在HAL\_TIMER\_NUM0 - HAL\_TIMER\_NUM23。

* 使用例子：

注册第三个timer，装载时间为1ms，中断函数为TIMHAL\_IRQHandler0

|  |
| --- |
| HAL\_TIMER\_RegisterTimer(HAL\_TIMER\_NUM3, 1000, TIMHAL\_IRQHandler0); |

#### HAL\_TIMER\_Stop

* 完整名称：

|  |
| --- |
| HAL\_RET\_T HAL\_TIMER\_ Stop (ENUM\_HAL\_TIMER\_NUM e\_timerNum) |

* 功能说明：

停止 timer。

* 参数说明：

1. e\_timerNum: 要设置的tiemr，正确的为0—23。

* 数据结构说明：

参考“ENUM\_HAL\_TIMER\_NUM”。

* 返回值：

1. HAL\_OK 函数执行成功。
2. HAL\_TIMER\_ERR\_UNKNOWN 没有找到相应的timer。

* 注意事项：

1. e\_timerNum的值必须在在HAL\_TIMER\_NUM0 - HAL\_TIMER\_NUM23。

* 使用例子：

停止第三个timer，timer停止计数

|  |
| --- |
| HAL\_RET\_T HAL\_TIMER\_Stop(HAL\_TIMER\_NUM3); |

#### HAL\_TIMER\_Start

* 完整名称：

|  |
| --- |
| HAL\_RET\_T HAL\_TIMER\_Start(ENUM\_HAL\_TIMER\_NUM e\_timerNum) |

* 功能说明：

开始 timer。

* 参数说明：

1. e\_timerNum: 要设置的tiemr，正确的为0—23。

* 数据结构说明：

参考“ENUM\_HAL\_TIMER\_NUM”。

* 返回值：

1. HAL\_OK 函数执行成功。
2. HAL\_TIMER\_ERR\_UNKNOWN 没有找到相应的timer。

* 注意事项：

1. e\_timerNum的值必须在在HAL\_TIMER\_NUM0 - HAL\_TIMER\_NUM23。

* 使用例子：

开始第三个timer，timer停止计数

|  |
| --- |
| HAL\_RET\_T HAL\_TIMER\_Start (HAL\_TIMER\_NUM3); |

#### HAL\_PWM\_RegisterPwm

* 完整名称：

|  |
| --- |
| HAL\_RET\_T HAL\_PWM\_RegisterPwm(ENUM\_HAL\_PWM\_NUM e\_pwmNum,  uint32\_t u32\_lowus,  uint32\_t u32\_highus) |

* 功能说明：

注册一个pwm。

* 参数说明：

1. e\_pwmNum: 要设置的pwm控制器，正确的为HAL\_PWM\_NUM0 - HAL\_PWM\_NUM9。
2. u32\_lowus: 低电平微妙。
3. u32\_highus: 高电平微妙。

* 数据结构说明：

参考“ENUM\_HAL\_PWM\_NUM”。

* 返回值：

1. HAL\_OK 函数执行成功。
2. HAL\_PWM\_ERR\_UNKNOWN 错误的pwm发生器。

* 注意事项：

1. 前10个timer对应0到9个pwm发生器，所以使用pwm，就不能使用相应的timer，同样适用timer就不能使用pwm。

* 使用例子：

设置pwm3产生高低电平各位1ms的波形，但还没有开始输出波形。

|  |
| --- |
| HAL\_PWM\_RegisterPwm(HAL\_PWM\_NUM3, 1000,1000); |

#### HAL\_PWM\_Start

* 完整名称：

|  |
| --- |
| HAL\_RET\_T HAL\_PWM\_Start(ENUM\_HAL\_PWM\_NUM e\_pwmNum) |

* 功能说明：

注册tiemr，并且timer开始计数。

* 参数说明：

1. e\_pwmNum: 要设置的pwm控制器，正确的为HAL\_PWM\_NUM0 - HAL\_PWM\_NUM9。

* 数据结构说明：

参考“ENUM\_HAL\_PWM\_NUM”。

* 返回值：

1. HAL\_OK 函数执行成功。
2. HAL\_PWM\_ERR\_UNKNOWN 错误的pwm发生。

* 注意事项：

1. 相应的pwm产生波形，相应的pwm控制器必须注册。

* 使用例子：

pwm3产生波形。

|  |
| --- |
| HAL\_PWM\_Start (HAL\_PWM\_NUM3); |

#### HAL\_PWM\_Stop

* 完整名称：

|  |
| --- |
| HAL\_RET\_T HAL\_PWM\_Stop(ENUM\_HAL\_PWM\_NUM e\_pwmNum) |

* 功能说明：

停止产生波形。

* 参数说明：

1. e\_pwmNum: 要设置的pwm控制器，正确的为HAL\_PWM\_NUM0 – HAL\_PWM\_NUM9。

* 数据结构说明：

参考“ENUM\_HAL\_PWM\_NUM”。

* 返回值：

1. HAL\_OK 函数执行成功。
2. HAL\_PWM\_ERR\_UNKNOWN 错误的pwm发生器。

* 注意事项：

1. 停止相应的pwm产生波形。

* 使用例子：

停止pwm3产生波形。

|  |
| --- |
| HAL\_PWM\_ Stop (HAL\_PWM\_NUM3); |

#### HAL\_PWM\_DynamicModifyPwmDutyCycle

* 完整名称：

|  |
| --- |
| HAL\_RET\_T HAL\_PWM\_DynamicModifyPwmDutyCycle(ENUM\_HAL\_PWM\_NUM e\_pwmNum,  uint32\_t u32\_lowus,  uint32\_t u32\_highus) |

* 功能说明：

动态的改变占空比。

* 参数说明：

1. e\_pwmNum: 要设置的pwm控制器，正确的为HAL\_PWM\_NUM0 - HAL\_PWM\_NUM9。
2. u32\_lowus: 低电平微妙。
3. u32\_highus: 高电平微妙。

* 数据结构说明：

参考“ENUM\_HAL\_PWM\_NUM”。

* 返回值：

1. HAL\_OK 函数执行成功。
2. HAL\_PWM\_ERR\_UNKNOWN 错误的pwm发生器。

* 注意事项：

无。

* 使用例子：

改变pwm3的占空比。

|  |
| --- |
| HAL\_PWM\_DynamicModifyPwmDutyCycle (HAL\_PWM\_NUM3，1200，1400); |

#### HAL\_SOFTPWM\_SetTimer

* 完整名称：

|  |
| --- |
| HAL\_RET\_T HAL\_SOFTPWM\_SetTimer(ENUM\_HAL\_TIMER\_NUM e\_timerNum) |

* 功能说明：

注册一个timer。

* 参数说明：

1. e\_timerNum: 要设置的tiemr，正确的为0—23。

* 数据结构说明：

参考“ENUM\_HAL\_PWM\_NUM”。

* 返回值：

1. HAL\_OK 函数执行成功。

* 注意事项：

1. e\_timerNum的值必须在在HAL\_TIMER\_NUM0 - HAL\_TIMER\_NUM23。

* 使用例子：

开始一个timer。

|  |
| --- |
| HAL\_SOFTPWM\_SetTimer (HAL\_TIMER\_NUM4); |

#### HAL\_SOFTPWM\_AddPwm

* 完整名称：

|  |
| --- |
| HAL\_RET\_T HAL\_SOFTPWM\_AddPwm(STRU\_SoftPwmHandle \*tmp) |

* 功能说明：

添加一个pwm。

* 参数说明：

1. e\_timerNum: 要设置的tiemr，正确的为0—23。

* 数据结构说明：

参考“STRU\_SoftPwmHandle”

* 返回值：

1. HAL\_OK 函数执行成功。

* 注意事项：

1. Function程序中自动会设置成HAL\_GPIO\_SetPin。
2. u32\_countArray[2]用户只需要设置， [0]是pwm的高电平时间。[1]是低电平时间。
3. u8\_pin 是要输出的gpio。

* 使用例子：

高电平时间（100×20）um，低电平时间（200×20）um，第64个GPIO输出相应的波形。

|  |
| --- |
| tmp.u32\_countArray[0] =100;  tmp.u32\_countArray[1] =200;  tmp.u8\_pin = 64;  HAL\_SOFTPWM\_AddPwm(&tmp); |

#### HAL\_SOFTPWM\_RunPwm

* 完整名称：

|  |
| --- |
| HAL\_RET\_T HAL\_SOFTPWM\_RunPwm(void) |

* 功能说明：

设置要产生pin和时间后，死循环函数，将产生相应的波形。

* 参数说明：

无。

* 数据结构说明：

无。

* 返回值：

1. HAL\_OK 函数执行成功。

* 注意事项：
* 使用例子：

|  |
| --- |
| while (1)  {  HAL\_SOFTPWM\_RunPwm();  for (i=0;i<MAXSOFTPWM;i++)  {  if ((g\_stPwmQueue[i].u8\_polarity == 0) &&  (g\_ACount[i][0] == 0))  {  tmp2[i]++;  if(tmp2[i]>2)  {  tmp2[i]=1;  }  g\_ACount[i][0] =40\* tmp2[i];  g\_ACount[i][1] =50\* tmp2[i];  }  }  } |

模拟pwm，使用一个timer控制10个gpio，产生pwm波形，高电平时间或低电平时间的设置都是以20um为最小分辨率设置，例如初始化要设置高电平时间为2000um则结构体STRU\_SoftPwmHandle中的变量u32\_countArray[0]=100;//2000=100\*20,低电平时间为1000um则结构体STRU\_SoftPwmHandle中的变量u32\_countArray[1]=50;//1000=50\*20，同时可以变化pwm的周期。例如执行一个周期后，下一个周期高低电平时间可以根据需求设置。在执行一个周期后会有标志位结构体STRU\_SoftPwmHandle中的变量u8\_polarity会变成0，并且全局变量g\_ACount会等于0，设置的周期时间应赋值到全局变量g\_ACount中。Testhal\_softpwm.c中有具体例子。

### SPI Master

SPI是全双工同步串行总线，SPI由SDI（串行数据输入），SDO（串行数据输出），SCK（串行移位时钟输出）和 CS（从使能信号）。

在AR8020中有8组SPI总线SPI0~SPI7。其中SPI7专门用于基带的控制。

使用该部分API函数，需要包含头文件"hal\_spi.h"。

#### HAL\_SPI\_MasterInit

* 完整名称：

|  |
| --- |
| HAL\_RET\_T HAL\_SPI\_MasterInit(ENUM\_HAL\_SPI\_COMPONENT e\_spiComponent,  STRU\_HAL\_SPI\_INIT \*pst\_spiInitInfo) |

* 功能说明：

初始化SPI控制器（master模式）。

* 参数说明：

1. e\_spiComponent：取值范围HAL\_SPI\_COMPONENT\_0 ~ HAL\_SPI\_COMPONENT\_6。
2. pst\_spiInitInfo：初始化SPI需要的控制信息：
3. pst\_spiInitInfo->u16\_halSpiBaudr取值范围为1~165，数值越大，SPI的速率越高。
4. pst\_spiInitInfo->e\_halSpiPolarity取值为HAL\_SPI\_POLARITY\_LOW、HAL\_SPI\_POLARITY\_HIGH。
5. pst\_spiInitInfo->e\_halSpiPhase取值为HAL\_SPI\_PHASE\_1EDGE、HAL\_SPI\_PHASE\_2EDGE。

* 数据结构说明：

参考“ENUM\_HAL\_SPI\_COMPONENT”、“ENUM\_HAL\_SPI\_POLARITY”、“ENUM\_HAL\_SPI\_PHASE”、“STRU\_HAL\_SPI\_INIT”。

* 返回值：

1. HAL\_OK：SPI控制器初始化成功。
2. HAL\_SPI\_ERR\_INIT：SPI控制器初始化失败。

* 注意事项：

无

* 使用例子：

|  |
| --- |
| st\_spiInitInfo.u16\_halSpiBaudr = 9;  st\_spiInitInfo.e\_halSpiPolarity = HAL\_SPI\_POLARITY\_LOW;  st\_spiInitInfo.e\_halSpiPhase = HAL\_SPI\_PHASE\_2EDGE;  HAL\_SPI\_MasterInit(HAL\_SPI\_COMPONENT\_7,&st\_spiInitInfo); |

#### HAL\_SPI\_MasterWriteRead

* 完整名称：

|  |
| --- |
| HAL\_RET\_T HAL\_SPI\_MasterWriteRead(ENUM\_HAL\_SPI\_COMPONENT e\_spiComponent,  uint8\_t \*pu8\_wrData,  uint32\_t u32\_wrSize,  uint8\_t \*pu8\_rdData,  uint32\_t u32\_rdSize) |

* 功能说明：

使用SPI接口进行数据收发。

* 参数说明：

1. e\_spiComponent：取值范围HAL\_SPI\_COMPONENT\_0 ~ HAL\_SPI\_COMPONENT\_6。
2. pu8\_wrData：需要发送数据存储区首地址。
3. u32\_wrSize：需要发送数据长度，按字节计算。
4. pu8\_rdData：读取数据储存区首地址。
5. u32\_rdSize：需要读取数据长度，数值为0或者等于u32\_wrSize。

* 数据结构说明：

参考“ENUM\_HAL\_SPI\_COMPONENT”。

* 返回值：

1. HAL\_OK：SPI收发数据成功。
2. HAL\_SPI\_ERR\_COMPONENT：SPI通道错误。
3. HAL\_SPI\_ERR\_WRITE\_DATA：SPI发送数据失败。
4. HAL\_SPI\_ERR\_READ\_DATA：SPI接收数据失败。

* 注意事项：

1. 只进行数据发送时，pu8\_rdData设置为NULL，u32\_rdSize设置为0。
2. 进行数据接收时，u32\_rdSize数值必须与u32\_wrSize相等，并且u32\_rdSize的数值等于需要发送命令的长度 + 实际需要读取数据的长度。

* 使用例子：

1. 通过SPI的7通道发送3个字节。

|  |
| --- |
| HAL\_SPI\_MasterWriteRead(HAL\_SPI\_COMPONENT\_7, u8\_txData, 3, NULL, 0); |

1. 通过SPI的7通道接收1个字节。

|  |
| --- |
| HAL\_SPI\_MasterWriteRead(HAL\_SPI\_COMPONENT\_7, u8\_txData, 3, u8\_rxData, 3); |

其中u8\_txData[0]为读取数据时需要发送的命令，u8\_txData[1]为读取数据的地址，u8\_txData[2]可以为任意数值，仅用于产生时钟。

u8\_rxData[0]、u8\_rxData[1]为无效数据（发送命令和地址时，产生的垃圾数据），u8\_rxData[2]才是真正读取到的有效数据。

### Baseband

使用者部分API，需要包含头文件“hal\_bb.h”。

#### HAL\_BB\_initGround

* 完整名称：

|  |
| --- |
| HAL\_RET\_T HAL\_BB\_initGround( void ); |

* 功能说明：

初始化Baseband为地面端

* 参数说明：
* 返回值：

1. HAL\_OK：初始化成功。
2. HAL\_BB\_ERR\_INIT：初始化失败。

* 注意事项：

1. 该函数只在cpu2内部被调用。cpu0,cpu1不能够调用。

* 使用例子：

无。

#### HAL\_BB\_initSky

* 完整名称：

|  |
| --- |
| HAL\_RET\_T HAL\_BB\_initSky( void ); |

* 功能说明：

初始化Baseband为天空端

* 参数说明：
* 返回值：

1. HAL\_OK：初始化成功。
2. HAL\_BB\_ERR\_INIT：初始化失败。

* 注意事项：

1. 该函数只在cpu2内部被调用。cpu0,cpu1不能够调用。

* 使用例子：

无。

#### HAL\_BB\_SetFreqBandwidthSelectionProxy

* 完整名称：

|  |
| --- |
| HAL\_RET\_T HAL\_BB\_SetFreqBandwidthSelectionProxy(ENUM\_CH\_BW e\_bandwidth) |

* 功能说明：

cpu0，cpu1发送event到cpu2，设置Baseband的传输带宽

* 参数说明：

1. e\_bandwidth：baseband传输带宽模式：10M bandwidth, 20M bandwidth

* 数据结构说明：

参考“[ENUM\_CH\_BW](#_ENUM_CH_BW)”

* 返回值：

1. HAL\_OK：正常发送
2. HAL\_BB\_ERR\_EVENT\_NOTIFY：发送event失败

* 注意事项：

1. 本函数只在cpu0,cpu1中调用。
2. 只有在5G模式下，支持20M bandwidth

* 使用例子：

1. 设置频点的带宽为10MHz:

|  |
| --- |
| HAL\_BB\_SetFreqBandwidthSelectionProxy(BW\_10M); |

#### HAL\_BB\_SetItChannelSelectionModeProxy

* 完整名称：

|  |
| --- |
| HAL\_RET\_T HAL\_BB\_SetItChannelSelectionModeProxy(ENUM\_RUN\_MODE e\_mode) |

* 功能说明：

cpu0，cpu1发送event到cpu2，设置图传channel选择模式

* 参数说明：

1. e\_mode：图传channel选择模式。在AUTO模式下，系统会根据信道质量自动选择图传channel

* 数据结构说明：

参考“[ENUM\_RUN\_MODE](#_ENUM_RUN_MODE)”

* 返回值：

1. HAL\_OK：正常发送
2. HAL\_BB\_ERR\_EVENT\_NOTIFY：发送event失败

* 注意事项：

1. 本函数只在cpu0,cpu1中调用。

* 使用例子：

设置图传channel的选择模式为AUTO：

|  |
| --- |
| HAL\_BB\_SetItChannelSelectionModeProxy (AUTO); |

#### HAL\_BB\_SetMcsModeProxy

* 完整名称：

|  |
| --- |
| HAL\_RET\_T HAL\_BB\_SetMcsModeProxy(ENUM\_RUN\_MODE e\_mode) |

* 功能说明：

cpu0，cpu1发送event到cpu2，设置MCS(mode channel scheme)选择模式，系统会根据当前信道质量自动选择调制模式(QAM, LDPC)

* 参数说明：

1. e\_mode：MCS (mode channel scheme)选择模式。在ATUO模式下，

* 数据结构说明：

参考“[ENUM\_RUN\_MODE](#_ENUM_RUN_MODE)”

* 返回值：

1. HAL\_OK：正常发送
2. HAL\_BB\_ERR\_EVENT\_NOTIFY：发送event失败

* 注意事项：

1. 本函数只在cpu0,cpu1中调用。

* 使用例子：

设置MCS（）：

|  |
| --- |
| HAL\_BB\_SetMcsModeProxy (AUTO); |

#### HAL\_BB\_SetItQamProxy

* 完整名称：

|  |
| --- |
| HAL\_RET\_T HAL\_BB\_SetItQamProxy(ENUM\_BB\_QAM e\_qam) |

* 功能说明：

cpu0，cpu1发送event到cpu2，设置baseband的QAM模式

* 参数说明：

1. e\_qam：调制QAM模式

* 数据结构说明：

参考“[ENUM\_BB\_QAM](#_ENUM_BB_QAM)”

* 返回值：

1. HAL\_OK：正常发送
2. HAL\_BB\_ERR\_EVENT\_NOTIFY：发送event失败

* 注意事项：

1. 本函数只在cpu0,cpu1中调用。

* 使用例子：

设置图传QAM模式为16QAM,在天空端调用函数：

|  |
| --- |
| HAL\_BB\_SetItQamProxy(MOD\_16QAM); |

#### HAL\_BB\_SetItLdpcProxy

* 完整名称：

|  |
| --- |
| HAL\_RET\_T HAL\_BB\_SetItLdpcProxy(ENUM\_BB\_LDPC e\_ldpc) |

* 功能说明：

cpu0，cpu1发送event到cpu2，设置baseband的LDPC模式

* 参数说明：

1. e\_ldpc：调制ldpc模式

* 数据结构说明：

参考“[ENUM\_BB\_LDPC](#_ENUM_BB_LDPC)”

* 返回值：

1. HAL\_OK：正常发送
2. HAL\_BB\_ERR\_EVENT\_NOTIFY：发送event失败

* 注意事项：

1. 本函数只在cpu0,cpu1中调用。

* 使用例子：

|  |
| --- |
| HAL\_BB\_SetItLdpcProxy (LDPC\_1\_2) |

#### HAL\_BB\_SetEncoderBrcModeProxy

* 完整名称：

|  |
| --- |
| HAL\_RET\_T HAL\_BB\_SetEncoderBrcModeProxy(ENUM\_RUN\_MODE e\_mode) |

* 功能说明：

cpu0，cpu1发送event到cpu2，设置H264 encoder bitrate控制选择模式

* 参数说明：

1. e\_mode：H264 encoder bitrate选择模式，在AUTO模式下，码率会根据QAM, LDPC来自动选择。

* 数据结构说明：

参考“[ENUM\_RUN\_MODE](#_ENUM_RUN_MODE)”

* 返回值：

1. HAL\_OK：正常发送
2. HAL\_BB\_ERR\_EVENT\_NOTIFY：发送event失败

* 注意事项：

1. 本函数只在cpu0,cpu1中调用。

* 使用例子：

设置H264 encoder 码率控制模式：

|  |
| --- |
| HAL\_BB\_SetEncoderBrcModeProxy (AUTO); |

#### HAL\_BB\_SetEncoderBitrateProxy

* 完整名称：

|  |
| --- |
| HAL\_RET\_T HAL\_BB\_SetEncoderBitrateProxy(uint8\_t u8\_bitrateMbps) |

* 功能说明：

cpu0，cpu1发送event到cpu2，设置H264 encoder bitrate,单位：Mbps

* 参数说明：

1. u8\_bitrateMbps：H264 encoder bitrate, unit: Mbps. u8\_bitrateMbps = 0 means 500Kbps

* 数据结构说明：

无。

* 返回值：

1. HAL\_OK：正常发送
2. HAL\_BB\_ERR\_EVENT\_NOTIFY：发送event失败

* 注意事项：

1. 本函数只在cpu0,cpu1中调用。

* 使用例子：

设置H264 encoder bitrate为500Kbps：

|  |
| --- |
| HAL\_BB\_SetEncoderBitrateProxy(0); |

#### HAL\_BB\_UartComRemoteSessionInit

* 完整名称：

|  |
| --- |
| HAL\_RET\_T HAL\_BB\_UartComRemoteSessionInit(void) |

* 功能说明：

初始化uart的会话。

* 参数说明：

无。

* 数据结构说明：

无。

* 返回值：

1. HAL\_OK：成功初始化一个会话
2. HAL\_BB\_ERR\_INIT\_SESSION：初始化会话失败

* 注意事项：

1. 本函数只在cpu0,cpu1中调用。

* 使用例子：

参考“HAL\_BB\_UartComReceiveMsg”的使用例子。

#### HAL\_BB\_UartComRegisterSession

* 完整名称：

|  |
| --- |
| HAL\_RET\_T HAL\_BB\_UartComRegisterSession(ENUM\_BBUARTCOMSESSIONID e\_sessionId) |

* 功能说明：

注册一个会话用于天空地面端的数据传输

* 参数说明：

1. e\_session：参数范围为BB\_UART\_COM\_SESSEION\_0~ BB\_UART\_COM\_SESSEION\_MAX-1， BB\_UART\_COM\_SESSION\_0是cpu2专用。

* 数据结构说明：

参考“[ENUM\_BBUARTCOMSESSIONID](#_ENUM_BBUARTCOMSESSIONID)”。

* 返回值：

1. HAL\_OK：注册会话成功
2. HAL\_BB\_ERR\_SESSION\_OCCUPIED：注册会话失败，没有可以申请的会话

* 注意事项：

1. 必须先调用HAL\_BB\_UartComRemoteSessionInit初始化,然后才能够正常申请会话

* 使用例子：

参考“HAL\_BB\_UartComReceiveMsg”的使用例子。

#### HAL\_BB\_UartComUnRegisterSession

* 完整名称：

|  |
| --- |
| HAL\_RET\_T HAL\_BB\_UartComUnRegisterSession(ENUM\_BBUARTCOMSESSIONID  e\_sessionId) |

* 功能说明：

注册一个会话用于天空地面端的数据传输

* 参数说明：

1. e\_session：参数范围为BB\_UART\_COM\_SESSEION\_0~ BB\_UART\_COM\_SESSEION\_MAX-1，BB\_UART\_COM\_SESSION\_0是cpu2专用。

* 数据结构说明：

参考“[ENUM\_BBUARTCOMSESSIONID](#_ENUM_BBUARTCOMSESSIONID)”。

* 返回值：

1. HAL\_OK：注销会话成功
2. HAL\_BB\_ERR\_UNREGISTER\_SESSION：注册session失败

* 注意事项：

1. 必须先调用HAL\_BB\_UartComRemoteSessionInit初始化

* 使用例子：

参考“HAL\_BB\_UartComReceiveMsg”的使用例子。

#### HAL\_BB\_UartComSendMsg

* 完整名称：

|  |
| --- |
| HAL\_RET\_T HAL\_BB\_UartComSendMsg(ENUM\_BBUARTCOMSESSIONID e\_sessionId,  uint8\_t \*pu8\_dataBuf,  uint32\_t u32\_length) |

* 功能说明：

通过串口向已经注册的会话发送数据

* 参数说明：

1. e\_session：参数范围为BB\_UART\_COM\_SESSEION\_0~ BB\_UART\_COM\_SESSEION\_MAX-1，BB\_UART\_COM\_SESSION\_0是cpu2专用。
2. pu8\_dataBuf： 指向需要通过会话发送的数据指针
3. u32\_length： 需要发送数据的长度

* 数据结构说明：

参考“[ENUM\_BBUARTCOMSESSIONID](#_ENUM_BBUARTCOMSESSIONID)”。

* 返回值：

1. HAL\_OK：发送数据成功
2. HAL\_BB\_ERR\_SESSION\_SEND：发送数据失败

* 注意事项：

1. 必须先调用HAL\_BB\_UartComRemoteSessionInit初始化，并且调用HAL\_BB\_UartComRegisterSession注册一个会话。

* 使用例子：

参考“HAL\_BB\_UartComReceiveMsg”的使用例子。

#### HAL\_BB\_UartComReceiveMsg

* 完整名称：

|  |
| --- |
| HAL\_RET\_T HAL\_BB\_UartComReceiveMsg(ENUM\_BBUARTCOMSESSIONID e\_sessionId,  uint8\_t \*pu8\_dataBuf,  uint32\_t u32\_lengthMax,  uint32\_t \*pu32\_dataLen) |

* 功能说明：

接收从串口收到的会话数据

* 参数说明：

1. e\_session：参数范围为BB\_UART\_COM\_SESSEION\_0~ BB\_UART\_COM\_SESSEION\_MAX-1，BB\_UART\_COM\_SESSION\_0是cpu2专用。
2. pu8\_dataBuf：指针，指向接收到的数据的存放地址
3. u32\_lengthMax：能够存放的最多数据的数量
4. pu32\_dataLen： 指针，指向实际接收到的会话数据

* 数据结构说明：

参考“ENUM\_BBUARTCOMSESSIONID”。

* 返回值：

1. HAL\_OK：发送数据成功
2. HAL\_BB\_ERR\_SESSION\_RCV：接收数据失败

* 注意事项：

1. 必须先调用HAL\_BB\_UartComRemoteSessionInit初始化，并且调用HAL\_BB\_UartComRegisterSession注册一个会话。

* 使用例子：

下面的例子可以用来实现天空到地面的数据传输：

天空端注册BB\_UART\_COM\_SESSION\_1，然后发送数据：

|  |
| --- |
| HAL\_BB\_UartComRegisterSession(BB\_UART\_COM\_SESSION\_1);  uint8\_t data\_buf\_tmp[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,  15, 16, 17, 18, 19, 20, 21, 22};  HAL\_BB\_UartComSendMsg(BB\_UART\_COM\_SESSION\_0,  data\_buf\_tmp,  sizeof(data\_buf\_tmp)); |

地面端注册BB\_UART\_COM\_SESSION\_1，然后接收数据数据：

|  |
| --- |
| uint32\_t len = 0;  HAL\_BB\_UartComRegisterSession(BB\_UART\_COM\_SESSION\_1);  HAL\_BB\_UartComReceiveMsg(BB\_UART\_COM\_SESSION\_0,  data\_buf\_proc,  sizeof(data\_buf\_proc),  &len);  uint32\_t i = 0;  for(i = 0; i < len; i++)  {  dlog\_info("%d,", data\_buf\_proc[i]);  } |

#### HAL\_BB\_SetRcFreqProxy

* 完整名称：

|  |
| --- |
| HAL\_RET\_T HAL\_BB\_SetRcFreqProxy(uint32\_t u32\_freqSetting); |

* 功能说明：

设置RC的工作频率

* 参数说明：

1. u32\_freqSetting：RC frequency registers settings

* 数据结构说明：
* 返回值：

1. HAL\_OK：发送数据成功
2. HAL\_BB\_ERR\_EVENT\_NOTIFY：发送event失败

* 注意事项：

本函数只在cpu0,cpu1中调用。

* 使用例子：

无

#### HAL\_BB\_SetItFreqProxy

* 完整名称：

|  |
| --- |
| HAL\_RET\_T HAL\_BB\_SetItFreqProxy (uint32\_t u32\_freqSetting); |

* 功能说明：

cpu0，cpu1发送event到cpu2，设置设置图传的工作频率

* 参数说明：

1. u32\_freqSetting：RC frequency registers settings

* 数据结构说明：
* 返回值：

1. HAL\_OK：发送数据成功
2. HAL\_BB\_ERR\_EVENT\_NOTIFY：发送event失败

* 注意事项：

本函数只在cpu0,cpu1中调用。

* 使用例子：

无

#### HAL\_RF8003s\_writeReg

* 完整名称：

|  |
| --- |
| HAL\_RET\_T HAL\_RF8003s\_writeReg(uint8\_t u8\_addr, uint8\_t u8\_data); |

* 功能说明：

通过SPI写RF 8003寄存器。

* 参数说明：

1. u8\_addr： RF8003 register address
2. u8\_data： data to write to the RF8003 address

* 数据结构说明：
* 返回值：

1. HAL\_OK：写RF8003寄存器成功
2. HAL\_BB\_ERR\_SPI\_WRITE：写RF8003寄存器失败

* 注意事项：

只有在测试模式下，才能够调用该函数

* 使用例子：

无

#### HAL\_RF8003s\_readByte

* 完整名称：

|  |
| --- |
| HAL\_RET\_T HAL\_RF8003s\_readByte(uint8\_t u8\_addr, uint8\_t \*pu8\_regValue); |

* 功能说明：

通过SPI读取RF 8003寄存器。

* 参数说明：

1. u8\_addr： RF8003 register address
2. pu8\_regValue：pointer to the data where to store the RF8003 register value

* 数据结构说明：
* 返回值：

1. HAL\_OK：写RF8003寄存器成功
2. HAL\_BB\_ERR\_SPI\_READ：读取RF8003寄存器失败

* 注意事项：

只有在测试模式下，才能够调用该函数

* 使用例子：

无

#### HAL\_BB\_writeByte

* 完整名称：

|  |
| --- |
| HAL\_RET\_T HAL\_BB\_writeByte(ENUM\_REG\_PAGES e\_page,  uint8\_t u8\_addr,  uint8\_t u8\_data); |

* 功能说明：

通过SPI写baseband寄存器。

* 参数说明：

1. e\_page： page number of the baseband register
2. u8\_addr： baseband register address
3. u8\_data： baseband register value

* 数据结构说明：

参考“ENUM\_REG\_PAGES”。

* 返回值：

1. HAL\_OK：写baseband寄存器成功
2. HAL\_BB\_ERR\_SPI\_WRITE：写baseband寄存器失败

* 注意事项：

只有在测试模式下，才能够调用该函数

* 使用例子：

无

#### HAL\_BB\_curPageWriteByte

* 完整名称：

|  |
| --- |
| HAL\_RET\_T HAL\_BB\_curPageWriteByte(uint8\_t u8\_addr, uint8\_t u8\_data); |

* 功能说明：

通过SPI写baseband当前页寄存器。

* 参数说明：

1. u8\_addr： baseband register address
2. u8\_data： baseband register value

* 数据结构说明：
* 返回值：

1. HAL\_OK：写baseband寄存器成功
2. HAL\_BB\_ERR\_SPI\_WRITE：写baseband寄存器失败

* 注意事项：

只有在测试模式下，才能够调用该函数

* 使用例子：

无

#### HAL\_BB\_curPageReadByte

* 完整名称：

|  |
| --- |
| HAL\_RET\_T HAL\_BB\_curPageReadByte(uint8\_t u8\_addr, uint8\_t \*pu8\_regValue); |

* 功能说明：

通过SPI读取baseband当前页寄存器。

* 参数说明：

1. u8\_addr： baseband register address
2. pu8\_regValue：pointer to the data where to store the RF8003 register value

* 数据结构说明：
* 返回值：

1. HAL\_OK：写baseband寄存器成功
2. HAL\_BB\_ERR\_SPI\_WRITE：写baseband寄存器失败

* 注意事项：

只有在测试模式下，才能够调用该函数

* 使用例子：

无

### System Controller

使用该部分API函数，需要包含头文件"hal\_sys\_ctl.h"。

#### HAL\_SYS\_CTL\_Init

* 完整名称：

|  |
| --- |
| HAL\_RET\_T HAL\_SYS\_CTL\_Init(STRU\_HAL\_SYS\_CTL\_CONFIG \*pst\_halSysCtlCfg) |

* 功能说明：

初始化CPU clock、FPU使能状态、system tick周期、天空端地面端工作模式等等。

* 参数说明：

1. pst\_halSysCtlCfg： 指向STRU\_HAL\_SYS\_CTL\_CONFIG结构体的指针，参考STRU\_HAL\_SYS\_CTL\_CONFIG定义。当pst\_halSysCtlCfg == NULL的时候，启动内部缺省system controller参数进行初始化。

* 数据结构说明：

参考“STRU\_HAL\_SYS\_CTL\_CONFIG”

* 返回值：

1. HAL\_OK：CPU clock、FPU使能状态、system tick周期、天空端地面端工作模式等等都初始化成功。
2. HAL\_SYS\_CTL\_ERR\_INIT：初始化过程出错。

* 注意事项：

可以通过HAL\_SYS\_CTL\_GetConfig获取内部缺省system controller参数。

* 使用例子：

使能FPU访问权限。

|  |
| --- |
| STRU\_HAL\_SYS\_CTL\_CONFIG \*pcfg;  HAL\_SYS\_CTL\_GetConfig(&pcfg);  Pcfg->u8\_fpuEnable = 1;  HAL\_SYS\_CTL\_Init(pcfg); |

#### HAL\_SYS\_CTL\_GetConfig

* 完整名称：

|  |
| --- |
| HAL\_RET\_T HAL\_SYS\_CTL\_GetConfig(STRU\_HAL\_SYS\_CTL\_CONFIG \*\*ppst\_halSysCtlCfg) |

* 功能说明：

获取默认CPU clock、FPU使能状态、system tick周期、天空端地面端工作模式等等的配置。

* 参数说明：

1. ppst\_halSysCtlCfg： 指向STRU\_HAL\_SYS\_CTL\_CONFIG结构体指针的指针，参考STRU\_HAL\_SYS\_CTL\_CONFIG定义。

* 数据结构说明：

参考“STRU\_HAL\_SYS\_CTL\_CONFIG”。

* 返回值：

1. HAL\_OK：成功获取CPU clock、FPU使能状态、system tick周期、天空端地面端工作模式等等参数。

* 注意事项：

无

* 使用例子：

使能FPU访问权限。

|  |
| --- |
| STRU\_HAL\_SYS\_CTL\_CONFIG \*pcfg;  HAL\_SYS\_CTL\_GetConfig(&pcfg);  Pcfg->u8\_fpuEnable = 1;  HAL\_SYS\_CTL\_Init(pcfg); |

#### HAL\_SYS\_CTL\_FpuEnable

* 完整名称：

|  |
| --- |
| HAL\_RET\_T HAL\_SYS\_CTL\_FpuEnable(uint8\_t u8\_fpuEnable) |

* 功能说明：

使能当前CPU上的FPU访问。

* 参数说明：

1. u8\_fpuEnable： 1 - 使能； 0 - 去使能。

* 数据结构说明：

无。

* 返回值：

1. HAL\_OK：成功修改FPU访问使能状态。

* 注意事项：

无

* 使用例子：

使能FPU访问权限。

|  |
| --- |
| HAL\_SYS\_CTL\_FpuEnable(1); |

#### HAL\_SYS\_CTL\_SetCpuClk

* 完整名称：

|  |
| --- |
| HAL\_RET\_T HAL\_SYS\_CTL\_SetCpuClk(uint16\_t u16\_cpu0cpu1Clk,  uint16\_t u16\_cpu2Clk) |

* 功能说明：

设置CPU0、CPU1、CPU2的clock。

* 参数说明：

1. u16\_cpu0cpu1Clk： CPU0与CPU1的CPU clock，单位是MHz。
2. u16\_cpu2Clk： CPU2的CPU clock，单位是MHz。

* 数据结构说明：

无。

* 返回值：

1. HAL\_OK：设置CPU clock成功。
2. HAL\_SYS\_CTL\_ERR\_SYS\_TICK\_INIT：设置CPU clock出错。

* 注意事项：

只有在CPU0上面设置CPU clock才是合法操作。

* 使用例子：

设置CPU0和CPU1的CPU clock为200M，CPU2的CPU clock为166M。

|  |
| --- |
| HAL\_SYS\_CTL\_SetCpuClk（200， 166）; |

### Cortex interrupt HAL

使用该部分API函数，需要包含头文件"hal\_nvic.h"。

#### HAL\_NVIC\_SetPriorityGrouping

* 完整名称：

|  |
| --- |
| HAL\_RET\_T HAL\_NVIC\_SetPriorityGrouping(uint32\_t priorityGroup) |

* 功能说明：

设置中断域。

* 参数说明：

priorityGroup: The priority grouping bits length.

This parameter can be one of the following values:

@arg NVIC\_PRIORITYGROUP\_0: 0 bits for preemption priority

5 bits for subpriority

@arg NVIC\_PRIORITYGROUP\_1: 1 bits for preemption priority

4 bits for subpriority

@arg NVIC\_PRIORITYGROUP\_2: 2 bits for preemption priority

3 bits for subpriority

@arg NVIC\_PRIORITYGROUP\_3: 3 bits for preemption priority

2 bits for subpriority

@arg NVIC\_PRIORITYGROUP\_4: 4 bits for preemption priority

1 bits for subpriority

@arg NVIC\_PRIORITYGROUP\_5: 5 bits for preemption priority

0 bits for subpriority

* 数据结构说明：

#define NVIC\_PRIORITYGROUP\_0 ((uint32\_t)0x00000007)

/\*!< 0 bits for pre-emption priority 5 bits for subpriority \*/

#define NVIC\_PRIORITYGROUP\_1 ((uint32\_t)0x00000006)

/\*!< 1 bits for pre-emption priority 4 bits for subpriority \*/

#define NVIC\_PRIORITYGROUP\_2 ((uint32\_t)0x00000005)

/\*!< 2 bits for pre-emption priority 3 bits for subpriority \*/

#define NVIC\_PRIORITYGROUP\_3 ((uint32\_t)0x00000004)

/\*!< 3 bits for pre-emption priority 2 bits for subpriority \*/

#define NVIC\_PRIORITYGROUP\_4 ((uint32\_t)0x00000003)

/\*!< 4 bits for pre-emption priority 1 bits for subpriority \*/

#define NVIC\_PRIORITYGROUP\_5 ((uint32\_t)0x00000002)

/\*!< 5 bits for pre-emption priority 0 bits for subpriority \*/

* 返回值：

1. HAL\_OK 函数执行成功。

* 注意事项：

无。

* 使用例子：

设置4位抢占优先级，1位子优先级。

|  |
| --- |
| HAL\_NVIC\_SetPriorityGrouping(NVIC\_PRIORITYGROUP\_4); |

#### HAL\_NVIC\_SetPriority

* 完整名称：

|  |
| --- |
| HAL\_RET\_T HAL\_NVIC\_SetPriority(ENUM\_HAL\_NVIC\_IRQ\_NUM e\_Irqn,  uint32\_t PreemptPriority,  uint32\_t SubPriority) |

* 功能说明：

设置相应中断的抢占优先级和子优先级。

* 参数说明：

1. e\_Irqn: 中断号。
2. PreemptPriority: 抢占优先级。
3. SubPriority:子优先级。

* 数据结构说明：

参考“ENUM\_HAL\_NVIC\_IRQ\_NUM”。

* 返回值：

1. HAL\_OK 函数执行成功。

* 注意事项：

无。

* 使用例子：

设置串口0中断，抢占优先级为1，子优先级为0。

|  |
| --- |
| HAL\_NVIC\_SetPriority (UART\_INTR0\_VECTOR\_NUM，1，0); |

#### HAL\_NVIC\_GetPriorityGrouping

* 完整名称：

|  |
| --- |
| HAL\_RET\_T HAL\_NVIC\_GetPriorityGrouping(uint32\_t \*p\_retPriorityGroup) |

* 功能说明：

获取中断域。

* 参数说明：

p\_retPriorityGroup: 返回的中断域会赋值到这个指针。

#define NVIC\_PRIORITYGROUP\_0 ((uint32\_t)0x00000007)

/\*!< 0 bits for pre-emption priority 5 bits for subpriority \*/

#define NVIC\_PRIORITYGROUP\_1 ((uint32\_t)0x00000006)

/\*!< 1 bits for pre-emption priority 4 bits for subpriority \*/

#define NVIC\_PRIORITYGROUP\_2 ((uint32\_t)0x00000005)

/\*!< 2 bits for pre-emption priority 3 bits for subpriority \*/

#define NVIC\_PRIORITYGROUP\_3 ((uint32\_t)0x00000004)

/\*!< 3 bits for pre-emption priority 2 bits for subpriority \*/

#define NVIC\_PRIORITYGROUP\_4 ((uint32\_t)0x00000003)

/\*!< 4 bits for pre-emption priority 1 bits for subpriority \*/

#define NVIC\_PRIORITYGROUP\_5 ((uint32\_t)0x00000002)

/\*!< 5 bits for pre-emption priority 0 bits for subpriority \*/

* 数据结构说明：

参考“ENUM\_HAL\_NVIC\_IRQ\_NUM”。

* 返回值：

1. HAL\_OK 函数执行成功。

* 注意事项：

无。

* 使用例子：

获取中断域。

|  |
| --- |
| HAL\_NVIC\_GetPriorityGrouping(p\_retPriorityGroup); |

#### HAL\_NVIC\_GetPriority

* 完整名称：

|  |
| --- |
| HAL\_RET\_T HAL\_NVIC\_GetPriority(ENUM\_HAL\_NVIC\_IRQ\_NUM e\_Irqn,  uint32\_t priorityGroup,  uint32\_t \*p\_preemptPriority,  uint32\_t \*p\_subPriority) |

* 功能说明：

获取中断域。

* 参数说明：

1. e\_Irqn: 中断号。
2. priorityGroup：The priority grouping bits length.
3. \*p\_preemptPriority: 返回抢占优先级的指针。
4. \*p\_subPriority:返回子优先级的指针。

* 数据结构说明：

参考“ENUM\_HAL\_NVIC\_IRQ\_NUM”。

* 返回值：

1. HAL\_OK 函数执行成功。

* 注意事项：

无。

* 使用例子：

获取串口0的优先级。

|  |
| --- |
| uint32\_t u32\_preemptPriority;  uint32\_t u32\_subPriority;  HAL\_NVIC\_GetPriority(UART\_INTR0\_VECTOR\_NUM,  NVIC\_PRIORITYGROUP\_3,  &u32\_preemptPriority,  &u32\_subPriority); |

#### HAL\_NVIC\_SetPendingIrq

* 完整名称：

|  |
| --- |
| HAL\_RET\_T HAL\_NVIC\_SetPendingIrq(ENUM\_HAL\_NVIC\_IRQ\_NUM e\_Irqn) |

* 功能说明：

设置IRQ中断挂起。

* 参数说明：

1. e\_Irqn: 中断号。

* 数据结构说明：

参考“ENUM\_HAL\_NVIC\_IRQ\_NUM”。

* 返回值：

1. HAL\_OK 函数执行成功。

* 注意事项：

无。

* 使用例子：

|  |
| --- |
| HAL\_NVIC\_SetPendingIrq (UART\_INTR0\_VECTOR\_NUM); |

#### HAL\_NVIC\_GetPendingIrq

* 完整名称：

|  |
| --- |
| HAL\_RET\_T HAL\_NVIC\_GetPendingIrq(ENUM\_HAL\_NVIC\_IRQ\_NUM e\_Irqn,  uint32\_t \*p\_retPending) |

* 功能说明：

设置IRQ中断挂起。

* 参数说明：

1. e\_Irqn: 中断号。

* 数据结构说明：

参考“ENUM\_HAL\_NVIC\_IRQ\_NUM”。

* 返回值：

1. HAL\_OK 函数执行成功。

* 注意事项：

无。

* 使用例子：

|  |
| --- |
| uint32\_t u32 \_retPending;  HAL\_NVIC\_GetPendingIrq (UART\_INTR0\_VECTOR\_NUM, & u32 \_retPending); |

#### HAL\_NVIC\_ClearPendingIrq

* 完整名称：

|  |
| --- |
| HAL\_RET\_T HAL\_NVIC\_ClearPendingIRQ(ENUM\_HAL\_NVIC\_IRQ\_NUM e\_Irqn) |

* 功能说明：

清除IRQ中断挂起状态。

* 参数说明：

1. e\_Irqn: 中断号。

* 数据结构说明：

参考“ENUM\_HAL\_NVIC\_IRQ\_NUM”。

* 返回值：

1. HAL\_OK 函数执行成功。

* 注意事项：

无。

* 使用例子：

|  |
| --- |
| HAL\_NVIC\_ClearPendingIrq(UART\_INTR0\_VECTOR\_NUM); |

#### HAL\_NVIC\_GetActive

* 完整名称：

|  |
| --- |
| HAL\_RET\_T HAL\_NVIC\_GetActive(ENUM\_HAL\_NVIC\_IRQ\_NUM e\_Irqn,  uint32\_t \*p\_retActive) |

* 功能说明：

返回有效的IRQ中断号。

* 参数说明：

1. e\_Irqn: 中断号。
2. p\_retActive: IRQ中断是否有效的指针。

* 数据结构说明：

参考“ENUM\_HAL\_NVIC\_IRQ\_NUM”。

* 返回值：

1. HAL\_OK 函数执行成功。

* 注意事项：

无。

* 使用例子：

|  |
| --- |
| uint32\_t u32\_ retActive;  HAL\_NVIC\_ClearPendingIrq (UART\_INTR0\_VECTOR\_NUM, & u32\_ retActive); |

#### HAL\_NVIC\_RegisterHandler

* 完整名称：

|  |
| --- |
| HAL\_NVIC\_RegisterHandler(ENUM\_HAL\_NVIC\_IRQ\_NUM e\_Irqn,  HAL\_NVIC\_Irqhandler NVIC\_Handler，  HAL\_NVIC\_Irqhandler NVIC\_ClearHandler) |
|  |

* 功能说明：

注册IRQ中断对应的中断函数。

* 参数说明：

1. e\_Irqn: 中断号。
2. NVIC\_Handler: e\_Irqn的中断处理函数。
3. NVIC\_ClearHandler：e\_Irqn的中断清除函数。

* 数据结构说明：

参考“ENUM\_HAL\_NVIC\_IRQ\_NUM”。

HAL\_NVIC\_Irqhandler 定义函数原型 void function（uint32\_t vectorNum）。

* 返回值：

1. HAL\_OK 函数执行成功。

* 注意事项：

无。

* 使用例子：

|  |
| --- |
| void ClearHandler（uint32\_t vectorNum）  void Handler（uint32\_t vectorNum）  HAL\_NVIC\_RegisterHandler(UART\_INTR0\_VECTOR\_NUM, Handler，ClearHandler); |

#### HAL\_NVIC\_UNRegisterHandler

* 完整名称：

|  |
| --- |
| HAL\_NVIC\_RegisterHandler(ENUM\_HAL\_NVIC\_IRQ\_NUM e\_Irqn) |

* 功能说明：

撤销IRQ中断对应的中断函数。

* 参数说明：

1. e\_Irqn: 中断号。

* 数据结构说明：

参考“ENUM\_HAL\_NVIC\_IRQ\_NUM”。

* 返回值：

1. HAL\_OK 函数执行成功。

* 注意事项：

无。

* 使用例子：

|  |
| --- |
| HAL\_NVIC\_RegisterHandler (UART\_INTR0\_VECTOR\_NUM); |

#### HAL\_NVIC\_EnableIrq

* 完整名称：

|  |
| --- |
| HAL\_NVIC\_EnableIrq(ENUM\_HAL\_NVIC\_IRQ\_NUM e\_Irqn) |

* 功能说明：

使能IRQ中断

* 参数说明：

1. e\_Irqn: 中断号。

* 数据结构说明：

参考“ENUM\_HAL\_NVIC\_IRQ\_NUM”。

* 返回值：

1. HAL\_OK 函数执行成功。

* 注意事项：

无。

* 使用例子：

|  |
| --- |
| HAL\_NVIC\_ EnableIrq(UART\_INTR0\_VECTOR\_NUM); |

#### HAL\_NVIC\_DisableIrq

* 完整名称：

|  |
| --- |
| HAL\_NVIC\_DisableIrq(ENUM\_HAL\_NVIC\_IRQ\_NUM e\_Irqn) |

* 功能说明：

关闭IRQ中断

* 参数说明：

1. e\_Irqn: 中断号。

* 数据结构说明：

参考“ENUM\_HAL\_NVIC\_IRQ\_NUM”。

* 返回值：

1. HAL\_OK 函数执行成功。

* 注意事项：

无。

* 使用例子：

|  |
| --- |
| HAL\_NVIC\_ DisableIrq(UART\_INTR0\_VECTOR\_NUM); |

### USB\_HOST

共有2个USB OTG口，USB Host0用于升级、读写U盘等功能，USB Host1专门用于UVC摄像头。

使用该部分API函数，需要包含头文件“hal\_usb\_host.h”。

#### HAL\_USB\_InitHost

* 完整名称：

|  |
| --- |
| void HAL\_USB\_InitHost(ENUM\_HAL\_USB\_PORT e\_usbPort,  ENUM\_HAL\_USB\_HOST\_CLASS e\_usbHostClass) |

* 功能说明：

初始化USB Host功能。

* 参数说明：

1. e\_usbPort：取值范围HAL\_USB\_PORT\_0 or HAL\_USB\_PORT\_1。
2. e\_usbHostClass：取值范围HAL\_USB\_HOST\_CLASS\_MSC or HAL\_USB\_HOST\_CLASS\_UVC。

* 数据结构说明：

参考“ENUM\_HAL\_USB \_PORT”、“ENUM\_HAL\_USB\_HOST\_CLASS”。

* 返回值：

1. 无。

* 注意事项：

1. 使用USB Host功能前必须调用此函数进行初始化。

* 使用例子：

将USB1初始化为UVC功能。

|  |
| --- |
| HAL\_USB\_InitHost(HAL\_USB\_PORT\_1, HAL\_USB\_HOST\_CLASS\_UVC); |

#### HAL\_USB\_HostProcess

* 完整名称：

|  |
| --- |
| void HAL\_USB\_HostProcess(void) |

* 功能说明：

轮询USB Host的设备接入、枚举等状态。

* 参数说明：

1. 无。

* 数据结构说明：

无。

* 返回值：

无。

* 注意事项：

1. 该函数用于USB Host状态机的轮询，和插入的USB设备进行枚举等功能。

* 使用例子：

|  |
| --- |
| 无 |

#### HAL\_USB\_GetHostAppState

* 完整名称：

|  |
| --- |
| ENUM\_HAL\_USB\_HOST\_STATE HAL\_USB\_GetHostAppState(void) |

* 功能说明：

获取当前USB Host的状态。

* 参数说明：

1. 无。

* 数据结构说明：

参考“ENUM\_HAL\_USB\_HOST\_STATE”。

* 返回值：

1. HAL\_USB\_HOST\_STATE\_IDLE
2. HAL\_USB\_HOST\_STATE\_READY
3. HAL\_USB\_HOST\_STATE\_DISCONNECT

* 注意事项：

当获取到的状态为HAL\_USB\_STATE\_READY时，表明此时已经完成了和插入的USB设备的枚举，可以开始进行USB操作。

* 使用例子：

插叙当前是否可以开始进行UVC

|  |
| --- |
| if (HAL\_USB\_STATE\_READE == HAL\_USB\_GetHostAppState())  {  HAL\_USB\_StartUVC( );  } |

#### HAL\_USB\_StartUVC

* 完整名称：

|  |
| --- |
| void HAL\_USB\_StartUVC(void) |

* 功能说明：

启动UVC，开始捕获视频数据。

* 参数说明：

1. 无。

* 数据结构说明：

无。

* 返回值：

1. 无

* 注意事项：

只有当UVC设备已经枚举完成后，UVC才能获取到视频流。

* 使用例子：

先查询当前是否可以开始进行UVC，状态为READY后

|  |
| --- |
| if (HAL\_USB\_STATE\_READY == HAL\_USB\_GetHostAppState())  {  HAL\_USB\_StartUVC( );  } |

#### HAL\_USB\_GetVideoFrame

* 完整名称：

|  |
| --- |
| HAL\_RET\_T HAL\_USB\_GetVideoFrame(uint8\_t \*u8\_buff) |

* 功能说明：

获取最新一帧UVC出来的图像。

* 参数说明：

1. u8\_buff:将获取到的图像存放到u8\_buff指向的buffer中。

* 数据结构说明：

无。

* 返回值：

1. HAL\_OK:图像获取成功
2. HAL\_USB\_ERROR\_BUFF\_IS\_EMPTY:视频缓冲区没有最新数据

* 注意事项：

调用该函数需要判断返回值是否获取成功，以识别数据有效性。

* 使用例子：

获取当前UVC缓冲区中最新Frame到g\_u8buff中。

|  |
| --- |
| if (HAL\_OK == HAL\_USB\_GetVideoFrame(g\_u8buff))  {  ProcessUVCData( g\_u8Buff);  } |

### USB\_DEVICE

共有2个USB OTG口，USB Device 0目前用于和PC、PAD等应用设备对接，传输图像、OSD信息等。

使用该部分API函数，需要包含头文件“hal\_usb\_device.h”。

#### HAL\_USB\_InitDevice

* 完整名称：

|  |
| --- |
| void HAL\_USB\_InitDevice(ENUM\_HAL\_USB\_PORT e\_usbPort) |

* 功能说明：

初始化USB Device功能。

* 参数说明：

1. e\_usbPort：取值范围HAL\_USB\_PORT\_0 ~ HAL\_USB\_PORT\_1。

* 数据结构说明：

参考“ENUM\_HAL\_USB \_PORT”。

* 返回值：

1. 无

* 注意事项：

无

* 使用例子：

打开USB0 USB Device功能

|  |
| --- |
| HAL\_USB\_InitDevice(HAL\_USB\_PORT\_0); |

#### HAL\_USB\_RegisterUserProcess

* 完整名称：

|  |
| --- |
| void HAL\_USB\_RegisterUserProcess(void (\*pUserFunc)(void \*)) |

* 功能说明：

注册用户接收主机数据的回调函数。

* 参数说明：

1. e\_ pUserFunc：用户的接收数据回调函数。

* 数据结构说明：

无

* 返回值：

无

* 注意事项：

无

* 使用例子：

|  |
| --- |
| void UserReceiveCallBack(void \*)  HAL\_USB\_Register(UserReceiveCallBack); |

### USB\_OTG

共有2个USB OTG口，如6.2.9和6.2.10所述，AR8020的USB控制器，通过firmware既可以将USB设置为HOST，作为主机使用，也可以将USB设置为DEVICE，作为从机使用。

使用该部分API函数，需要包含头文件“hal\_usb\_otg.h”。

#### HAL\_USB\_InitOTG

* 完整名称：

|  |
| --- |
| void HAL\_USB\_InitOTG(ENUM\_HAL\_USB\_PORT e\_usbPort) |

* 功能说明：

初始化USB OTG功能，firmware会根据当前USB ID线的高低电平去设置当前为HOST或是DEVICE。

* 参数说明：

1. e\_usbPort：取值范围HAL\_USB\_PORT\_0 ~ HAL\_USB\_PORT\_1。

* 数据结构说明：

参考“ENUM\_HAL\_USB \_PORT”。

* 返回值：

1. 无

* 注意事项：

无

* 使用例子：

打开USB0 USB OTG功能

|  |
| --- |
| HAL\_USB\_InitOTG(HAL\_USB\_PORT\_0); |

### SD

使用该部分API函数，需要包含头文件“hal\_sd.h”。

#### HAL\_SD\_Init

* 完整名称：

|  |
| --- |
| HAL\_RET\_T HAL\_SD\_Init(void) |

* 功能说明：

用以初始化AR8020 SD控制器，调用成功之后，可以正确操作SD卡。卡默认工作处于4线模式，速度为SDR25（数据传输率为25MB/s）

* 参数说明：

无

* 数据结构说明：

无

* 返回值：

1. HAL\_OK 初始化SD卡成功
2. HAL\_SD\_ERR\_ERROR 初始化SD卡失败

* 注意事项：

1. 在反初始化SD卡（即调用HAL\_SD\_Deinit()函数）之前，该函数有且只能调用一次，受SD控制器的限制，未做反初始化SD时，调用该函数则会陷入死循环

* 使用例子：

|  |
| --- |
| HAL\_RET\_T res;  if ((res = HAL\_SD\_Init()) != HAL\_OK)  {  Dlog\_Error("init SD failed\n");  return;  } |

#### HAL\_SD\_InitIRQ

* 完整名称：

|  |
| --- |
| void HAL\_SD\_InitIRQ(void) |

* 功能说明：

用以初始化SD卡控制器中断处理函数，调用该函数之后，可以实现SD卡热插拔等功能

* 参数说明：

无

* 数据结构说明：

无

* 返回值：

无

* 注意事项：

1. 由于该函数在实现时，已经包含识别卡插入检测并做初始化功能。因此该函数必须先于HAL\_SD\_Init()调用，调用成功后，即可实现卡热插拔等功能
2. 由于部分原因，SD驱动程序对于64G的SDXC卡热插拔支持较不稳定，该问题后续会修复

* 使用例子：

|  |
| --- |
| HAL\_RET\_T res;  HAL\_SD\_InitIRQ();  if ((res = HAL\_SD\_Init()) != HAL\_OK)  {  Dlog\_Error("init SD failed\n");  return;  } |

#### HAL\_SD\_Write

* 完整名称：

|  |
| --- |
| HAL\_RET\_T HAL\_SD\_Write(uint32\_t u32\_dstStartAddr,  uint32\_t u32\_srcStartAddr,  uint32\_t u32\_sectorNum) |

* 功能说明：

用以向SD卡写入数据，采用DMA方式写

* 参数说明：

1. u32\_dstStartAddr: 写入SD卡的LBA地址，单位为block
2. u32\_srcStartAddr: 数据的源地址，单位为byte
3. u32\_sectorNum: 写入SD卡的block数量，单位为block

* 数据结构说明：

无

* 返回值：

1. HAL\_OK 写SD卡成功
2. HAL\_SD\_ERR\_ERROR 写SD卡失败

* 注意事项：

1. 参数u32\_dstStartAddr为SD卡block地址，如向SD卡第0个block写数据，则该参数值为0，向第1个block写数据，则值为1; u32\_srcStartAddr为源数据地址，如从DTCM0起始地址（0x20000000）向SD卡写数据，则此值为0x20000000; u32\_sectorNum为写入数据的block数，如需写3个block,则此值为3

* 使用例子：

|  |
| --- |
| HAL\_SD\_Write(1, 0x20000000, 10); //从0x20000000地址处写10 block数据到SD卡1 block起始位置,SD卡被写的区域为1~10 block |

#### HAL\_SD\_Read

* 完整名称：

|  |
| --- |
| HAL\_RET\_T HAL\_SD\_Read(uint32\_t u32\_dstStartAddr,  uint32\_t u32\_srcStartAddr,  uint32\_t u32\_sectorNum) |

* 功能说明：

用以从SD卡读取数据到u32\_dstStartAddr地址处，采用DMA方式写

* 参数说明：

1. u32\_dstStartAddr: 目的地址，单位为byte
2. u32\_srcStartAddr: 从SD读数据的源地址，单位为block
3. u32\_sectorNum: 读取SD卡的block数量，单位为block

* 数据结构说明：

无

* 返回值：

1. HAL\_OK 读SD卡成功
2. HAL\_SD\_ERR\_ERROR 读SD卡失败

* 注意事项：

1. 参数u32\_srcStartAddr为SD卡block地址，如从SD卡第0个block读数据，则该参数值为0，从第1个block读数据，则值为1; u32\_dstStartAddr为目的地址，如向DTCM0起始地址（0x20000000）写入SD数据，则此值为0x20000000; u32\_sectorNum为读数据的block数，如需读3个block,则此值为3

* 使用例子：

|  |
| --- |
| HAL\_SD\_Read(0x20000000, 0, 10); //从 SD卡0 block起始位置读10个block到0x20000000地址处 |

#### HAL\_SD\_Erase

* 完整名称：

|  |
| --- |
| HAL\_RET\_T HAL\_SD\_Erase(uint32\_t u32\_startBlock, uint32\_t u32\_sectorNum) |

* 功能说明：

擦除SD卡相关区域数据

* 参数说明：

1. u32\_startBlock: 擦除SD的起始LBA地址，单位为block
2. u32\_sectorNum: 擦除的block数，单位为block

* 数据结构说明：

无

* 返回值：

1. HAL\_OK 擦除SD卡成功
2. HAL\_SD\_ERR\_ERROR 擦除SD卡失败

* 注意事项：

1. 参数u32\_startBlock和u32\_sectorNum单位都为block，如需擦除SD卡第0个block数据到第6个block数据,则u32\_startBlock为0，u32\_sectorNum为7

* 使用例子：

|  |
| --- |
| HAL\_SD\_Erase(0,7); |

#### HAL\_SD\_Ioctl

* 完整名称：

|  |
| --- |
| HAL\_RET\_T HAL\_SD\_Ioctl(ENUM\_HAL\_SD\_CTRL e\_sdCtrl, uint32\_t \*pu32\_info) |

* 功能说明：

获取卡相关信息和状态

* 参数说明：

1. e\_sdCtrl: 输入的宏控制变量，相关参数见“ENUM\_HAL\_SD\_CTRL”
2. pu32\_info: 获取的值

* 数据结构说明：

HAL\_SD\_GET\_CSD\_VERSION: 获取CSD 版本信息

HAL\_SD\_GET\_SECTOR\_COUNT: 获取SD卡容量，单位为MB

HAL\_SD\_GET\_SECTOR\_SIZE: 获取卡block长度，单位为byte

HAL\_SD\_GET\_TRAN\_SPEED: 获取卡速度信息

HAL\_SD\_GET\_CARD\_STATUS: 获取当前卡状态

HAL\_SD\_GET\_MANUID: 获取卡生产商ID号

HAL\_SD\_GET\_OEMID: 获取卡OEM ID

* 返回值：

1. HAL\_OK ioctl SD卡成功
2. HAL\_SD\_ERR\_ERROR ioctl SD卡失败

* 注意事项：

无

* 使用例子：

|  |
| --- |
| uint32\_t info = 0;  ENUM\_HAL\_SD\_CTRL cmd = HAL\_SD\_GET\_SECTOR\_COUNT;  if (HAL\_SD\_Ioctl(cmd, &info) != HAL\_OK)  {  Dlog\_Info("ioctl failed\n");  }  cmd = HAL\_SD\_GET\_SECTOR\_SIZE;  if (HAL\_SD\_Ioctl(cmd, &info) != HAL\_OK)  {  Dlog\_Info("ioctl failed\n");  }  cmd = HAL\_SD\_GET\_CSD\_VERSION;  if (HAL\_SD\_Ioctl(cmd, &info) != HAL\_OK)  {  Dlog\_Info("ioctl failed\n");  }  cmd = HAL\_SD\_GET\_MANUID;  if (HAL\_SD\_Ioctl(cmd, &info) != HAL\_OK)  {  Dlog\_Info("ioctl failed\n");  }  cmd = HAL\_SD\_GET\_TRAN\_SPEED;  if (HAL\_SD\_Ioctl(cmd, &info) != HAL\_OK)  {  Dlog\_Info("ioctl failed\n");  } |

#### HAL\_SD\_Deinit

* 完整名称：

|  |
| --- |
| HAL\_RET\_T HAL\_SD\_Deinit(void) |

* 功能说明：

反初始化SD卡，即卸载SD卡

* 参数说明：

无

* 数据结构说明：

无

* 返回值：

1. HAL\_OK 反初始化SD卡成功
2. HAL\_SD\_ERR\_ERROR 反初始化SD卡失败

* 注意事项：

无

* 使用例子：

|  |
| --- |
| HAL\_RET\_T res;  if ((res = HAL\_SD\_Deinit()) != HAL\_OK)  {  Dlog\_Error("Deinit SD failed\n");  return;  } |

### CAN

共有4路CAN总线，可使用标准模式（11位ID）或扩展模式（29位ID）。

使用该部分API函数，需要包含头文件“hal\_can.h”。

#### HAL\_CAN\_Init

* 完整名称：

|  |
| --- |
| HAL\_RET\_T HAL\_CAN\_Init(STRU\_HAL\_CAN\_CONFIG \*st\_halCanConfig) |

* 功能说明：

完成CAN总线控制器的初始化。

* 参数说明：

1. st\_halCanConfig：初始化信息结构体指针。

* 数据结构说明：

参考“ENUM\_HAL\_CAN\_COMPONENT”、“ENUM\_HAL\_CAN\_BAUDR”、“ENUM\_HAL\_CAN\_FORMAT”、“STRU\_HAL\_CAN\_CONFIG”。

* 返回值：

1. HAL\_OK：初始化成功。
2. HAL\_CAN\_ERR\_INIT：初始化失败。
3. HAL\_CAN\_ERR\_COMPONENT：通道参数错误。

* 注意事项：

无

* 使用例子：

将串口0初始化为标准模式，波特率500K，滤波码0x00123，对最低为ID不进行滤波。

|  |
| --- |
| STRU\_HAL\_CAN\_CONFIG st\_canConfig;  st\_canConfig.e\_halCanComponent = HAL\_CAN\_COMPONENT\_0;  st\_canConfig.e\_halCanBaudr = HAL\_CAN\_BAUDR\_500K;  st\_canConfig.u32\_halCanAcode = 0x123;  st\_canConfig.u32\_halCanAmask = 0x3;  st\_canConfig.e\_halCanFormat = HAL\_CAN\_FORMAT\_STD;  st\_canConfig.pfun\_halCanRcvMsg = test\_CAN\_RcvMsgHandler;    HAL\_CAN\_Init(&st\_canConfig); |

#### HAL\_CAN\_Send

* 完整名称：

|  |
| --- |
| HAL\_RET\_T HAL\_CAN\_Send(STRU\_HAL\_CAN\_MSG \*st\_halCanMsg) |

* 功能说明：

发送CAN总线报文帧。

* 参数说明：

1. st\_halCanMsg：待发送帧结构指针。

* 数据结构说明：

参考“ENUM\_HAL\_CAN\_COMPONENT”、“ENUM\_HAL\_CAN\_FORMAT”、“ENUM\_HAL\_CAN\_TYPE”、“STRU\_HAL\_CAN\_MSG”。

* 返回值：

1. HAL\_OK：发送成功。
2. HAL\_CAN\_ERR\_SEND\_MSG：发送失败。

* 注意事项：

无

* 使用例子：

从通道1发送扩展数据帧，数据长度为7，ID为0x123456。

|  |
| --- |
| uint8\_t u8\_ii;  STRU\_HAL\_CAN\_MSG st\_canSendMsg;    st\_canSendMsg.e\_halCanComponent = HAL\_CAN\_COMPONENT\_1;  st\_canSendMsg.u32\_halCanId = 0x123456;  for(u8\_ii=0;u8\_ii<7;u8\_ii++)  {  st\_canSendMsg.u8\_halCanDataArray[u8\_ii] = u8\_ii+1;  }  st\_canSendMsg.u8\_halCanDataLen = 7;  st\_canSendMsg.e\_halCanFormat = HAL\_CAN\_FORMAT\_EXT;  st\_canSendMsg.e\_halCanType = HAL\_CAN\_TYPE\_DATA;  HAL\_CAN\_Send(&st\_canSendMsg); |

#### CanCallBack

* 完整名称：

|  |
| --- |
| typedef uint32\_t (\*HAL\_CAN\_RcvMsgHandler)(struct STRU\_HAL\_CAN\_MSG \*pst\_canRxBuf, uint8\_t u8\_canRxCnt) |

* 功能说明：

在中断服务程序中被调用的函数，用于处理接收到的CAN数据。

* 参数说明：

1. pst\_canRxBuf：待接收数据存放地址。
2. u8\_canRxCnt：待接收数据长度。

* 数据结构说明：

参考“ENUM\_HAL\_CAN\_COMPONENT”、“ENUM\_HAL\_CAN\_FORMAT”、“ENUM\_HAL\_CAN\_TYPE”、“STRU\_HAL\_CAN\_MSG”。

* 返回值：

1. 无意义。

* 注意事项：

1. 函数为用户自定函数，格式必须符合结构uint32\_t (\*HAL\_CAN\_RcvMsgHandler)(struct STRU\_HAL\_CAN\_MSG \*pst\_canRxBuf, uint8\_t u8\_canRxCnt)。
2. HAL\_CAN\_RcvMsgHandler是在中断中被调用的，因此HAL\_CAN\_RcvMsgHandler不可耗时太长，仅仅处理数据即可。

* 使用例子：

配合前边HAL\_CAN\_Init的使用例子，完成CAN总线数据的接收。

|  |
| --- |
| static uint32\_t test\_CAN\_RcvMsgHandler(STRU\_HAL\_CAN\_MSG \*pst\_canRxBuf,  uint8\_t u8\_canRxCnt)  {  memcpy(&s\_st\_canRxBuf[s\_u8\_canRxCnt],  pst\_canRxBuf,  u8\_canRxCnt\*sizeof(STRU\_HAL\_CAN\_MSG));  s\_u8\_canRxCnt += u8\_canRxCnt;  return 0;  } |

### ADC

使用该部分API函数，需要包含头文件“hal\_adc.h”。

#### HAL\_ADC\_Read

* 完整名称：

|  |
| --- |
| uint32\_t HAL\_ADC\_Read(uint8\_t channel) |

* 功能说明：

用以读取ADC后模拟量转数字量的值，AR8020芯片ADC模块使用采样率为25Mhz/12≈2.0833Mhz，支持16个模拟输入，参数channel指定读取某一模拟输入的采样值。

* 参数说明：

channel指定获取相应通道的采样值，channel值为0~15

* 数据结构说明：

无

* 返回值：

返回值为采样值的十进制数。

* 注意事项：

无

* 使用例子：

无

### DMA

使用该部分API函数，需要包含头文件“hal\_dma.h”。

#### HAL\_DMA\_Start

* 完整名称：

|  |
| --- |
| void HAL\_DMA\_Start(uint32\_t u32\_srcAddress,  uint32\_t u32\_dstAddress,  uint32\_t u32\_dataLength,  ENUM\_Chan e\_channel,  ENUM\_TransferType e\_transType) |

* 功能说明：

用于实现DMA传输。

* 参数说明：

uint32\_t u32\_srcAddress：DMA传输数据的源地址

uint32\_t u32\_dstAddress：DMA传输数据的目的地址

uint32\_t u32\_dataLength：传输的数据长度（单位：byte）

ENUM\_Chan e\_channel：DMA通道选择，DMA同时支持8个道的数据传输，同时该参数支持自动选择空闲的通道进行传输。若指定0~7中某通道的数据传输，则填写参数为CHAN0~CHAN7；若使用自动通道选择，则该参数使用AUTO

ENUM\_TransferType e\_transType：DMA传输类型，支持自动链式存储读写,参数值为LINK\_LIST\_ITEM;支持自动重载读写，参数值为：AUTO\_RELOAD

* 数据结构说明：

无

* 返回值：

无

* 注意事项：

无

* 使用例子：

|  |
| --- |
| HAL\_DMA\_Start(0x10000000,0x20000000,20,AUTO,LINK\_LIST\_ITEM); |

## libfreertos.a

1. 操作系统（FreeRTOS）版本：V8.2.1。
2. 文件系统（FatFS）版本：R0.11。

# 附录1：函数索引

|  |  |  |
| --- | --- | --- |
| 序号 | 函数名 | 页码 |
|  | Dlog\_Init | 14 |
|  | Dlog\_Info | 14 |
|  | Dlog\_Warning | 15 |
|  | Dlog\_Error | 16 |
|  | Dlog\_Critical | 17 |
|  | Dlog\_Output | 17 |
|  | SYS\_EVENT\_RegisterHandler | 18 |
|  | SYS\_EVENT\_UnRegisterHandler | 19 |
|  | SYS\_EVENT\_Notify | 20 |
|  | SYS\_EVENT\_Process | 21 |
|  | HAL\_I2C\_MasterInit | 22 |
|  | HAL\_I2C\_MasterWriteData | 23 |
|  | HAL\_I2C\_MasterReadData | 24 |
|  | HAL\_UART\_Init | 25 |
|  | HAL\_UART\_TxData | 26 |
|  | UartCallBack | 26 |
|  | HAL\_GPIO\_OutPut | 27 |
|  | HAL\_GPIO\_InPut | 28 |
|  | HAL\_GPIO\_SetPin | 29 |
|  | HAL\_GPIO\_GetPin | 29 |
|  | HAL\_GPIO\_RegisterInterrupt | 30 |
|  | HAL\_TIMER\_RegisterTimer | 31 |
|  | HAL\_TIMER\_Stop | 32 |
|  | HAL\_TIMER\_Start | 33 |
|  | HAL\_PWM\_RegisterPwm | 33 |
|  | HAL\_PWM\_Start | 34 |
|  | HAL\_PWM\_Stop | 35 |
|  | HAL\_PWM\_DynamicModifyPwmDutyCycle | 35 |
|  | HAL\_SOFTPWM\_SetTimer | 36 |
|  | HAL\_SOFTPWM\_AddPwm | 36 |
|  | HAL\_SOFTPWM\_RunPwm | 37 |
|  | HAL\_SPI\_MasterInit | 39 |
|  | HAL\_SPI\_MasterWriteRead | 40 |
|  | HAL\_NVIC\_SetPriorityGrouping | 57 |
|  | HAL\_NVIC\_SetPriority | 58 |
|  | HAL\_NVIC\_GetPriorityGrouping | 59 |
|  | HAL\_NVIC\_GetPriority | 60 |
|  | HAL\_NVIC\_SetPendingIrq | 61 |
|  | HAL\_NVIC\_GetPendingIrq | 62 |
|  | HAL\_NVIC\_ClearPendingIrq | 62 |
|  | HAL\_NVIC\_GetActive | 63 |
|  | HAL\_NVIC\_RegisterHandler | 63 |
|  | HAL\_NVIC\_UNRegisterHandler | 64 |
|  | HAL\_NVIC\_EnableIrq | 65 |
|  | HAL\_NVIC\_DisableIrq | 65 |
|  | HAL\_USB\_InitHost | 66 |
|  | HAL\_USB\_HostProcess | 66 |
|  | HAL\_USB\_GetHostAppState | 67 |
|  | HAL\_USB\_StartUVC | 68 |
|  | HAL\_USB\_GetVideoFrame | 68 |
|  | HAL\_USB\_InitDevice | 69 |
|  | HAL\_USB\_RegisterUserProcess | 70 |
|  | HAL\_USB\_InitOTG | 71 |
|  | HAL\_SD\_Init | 71 |
|  | HAL\_SD\_InitIRQ | 72 |
|  | HAL\_SD\_Write | 73 |
|  | HAL\_SD\_Read | 74 |
|  | HAL\_SD\_Erase | 75 |
|  | HAL\_SD\_Ioctl | 75 |
|  | HAL\_SD\_Deinit | 77 |
|  | HAL\_SYS\_CTL\_Init | 54 |
|  | HAL\_SYS\_CTL\_GetConfig | 55 |
|  | HAL\_SYS\_CTL\_FpuEnable | 56 |
|  | HAL\_SYS\_CTL\_SetCpuClk | 56 |
|  | HAL\_BB\_initGround | 41 |
|  | HAL\_BB\_initSky | 42 |
|  | HAL\_BB\_SetFreqBandwidthSelectionProxy | 42 |
|  | HAL\_BB\_SetItChannelSelectionModeProxy | 43 |
|  | HAL\_BB\_SetMcsModeProxy | 43 |
|  | HAL\_BB\_SetItQamProxy | 44 |
|  | HAL\_BB\_SetItLdpcProxy | 44 |
|  | HAL\_BB\_SetEncoderBrcModeProxy | 45 |
|  | HAL\_BB\_SetEncoderBitrateProxy | 46 |
|  | HAL\_BB\_UartComRemoteSessionInit | 46 |
|  | HAL\_BB\_UartComRegisterSession | 47 |
|  | HAL\_BB\_UartComUnRegisterSession | 47 |
|  | HAL\_BB\_UartComSendMsg | 48 |
|  | HAL\_BB\_UartComReceiveMsg | 49 |
|  | HAL\_CAN\_Init | 78 |
|  | HAL\_CAN\_Send | 79 |
|  | HAL\_ADC\_Read | 81 |
|  | HAL\_DMA\_Start | 82 |

# 附录2：数据结构说明

#### ENUM\_HAL\_UART\_COMPONENT

typedef enum

{

HAL\_UART\_COMPONENT\_0 = 0,

HAL\_UART\_COMPONENT\_1,

HAL\_UART\_COMPONENT\_2,

HAL\_UART\_COMPONENT\_3,

HAL\_UART\_COMPONENT\_4,

HAL\_UART\_COMPONENT\_5,

HAL\_UART\_COMPONENT\_6,

HAL\_UART\_COMPONENT\_7,

HAL\_UART\_COMPONENT\_8,

} ENUM\_HAL\_UART\_COMPONENT;

#### ENUM\_HAL\_UART\_BAUDR

typedef enum

{

HAL\_UART\_BAUDR\_9600 = 0,

HAL\_UART\_BAUDR\_19200,

HAL\_UART\_BAUDR\_38400,

HAL\_UART\_BAUDR\_57600,

HAL\_UART\_BAUDR\_115200,

} ENUM\_HAL\_UART\_BAUDR;

#### ENUM\_HAL\_SPI\_COMPONENT

typedef enum

{

HAL\_SPI\_COMPONENT\_0 = 0,

HAL\_SPI\_COMPONENT\_1,

HAL\_SPI\_COMPONENT\_2,

HAL\_SPI\_COMPONENT\_3,

HAL\_SPI\_COMPONENT\_4,

HAL\_SPI\_COMPONENT\_5,

HAL\_SPI\_COMPONENT\_6,

HAL\_SPI\_COMPONENT\_7,

} ENUM\_HAL\_SPI\_COMPONENT;

#### ENUM\_HAL\_SPI\_POLARITY

typedef enum

{

HAL\_SPI\_POLARITY\_LOW = 0, // Inactive state of serial clock is low

HAL\_SPI\_POLARITY\_HIGH // Inactive state of serial clock is high

} ENUM\_HAL\_SPI\_POLARITY;

#### ENUM\_HAL\_SPI\_PHASE

typedef enum

{

// data are captured on the first edge of the serial clock

HAL\_SPI\_PHASE\_1EDGE = 0,

// data are captured on the second edge of the serial clock

HAL\_SPI\_PHASE\_2EDGE

} ENUM\_HAL\_SPI\_PHASE;

#### STRU\_HAL\_SPI\_INIT

typedef struct

{

uint16\_t u16\_halSpiBaudr; // range 1~165

ENUM\_HAL\_SPI\_POLARITY e\_halSpiPolarity;

ENUM\_HAL\_SPI\_PHASE e\_halSpiPhase;

} STRU\_HAL\_SPI\_INIT;

#### ENUM\_HAL\_I2C\_COMPONENT

typedef enum

{

HAL\_I2C\_COMPONENT\_0 = 0,

HAL\_I2C\_COMPONENT\_1,

HAL\_I2C\_COMPONENT\_2,

HAL\_I2C\_COMPONENT\_3,

HAL\_I2C\_COMPONENT\_4,

HAL\_I2C\_COMPONENT\_MAX

} ENUM\_HAL\_I2C\_COMPONENT;

#### ENUM\_HAL\_I2C\_SPEED

typedef enum

{

HAL\_I2C\_STANDARD\_SPEED = 0,

HAL\_I2C\_FAST\_SPEED,

HAL\_I2C\_HIGH\_SPEED

} ENUM\_HAL\_I2C\_SPEED;

#### ENUM\_HAL\_PWM\_NUM

typedef enum

{

HAL\_PWM\_NUM0=0,

HAL\_PWM\_NUM1,

HAL\_PWM\_NUM2,

HAL\_PWM\_NUM3,

HAL\_PWM\_NUM4,

HAL\_PWM\_NUM5,

HAL\_PWM\_NUM6,

HAL\_PWM\_NUM7,

HAL\_PWM\_NUM8,

HAL\_PWM\_NUM9

} ENUM\_HAL\_PWM\_NUM;

#### ENUM\_HAL\_TIMER\_NUM

typedef enum

{

HAL\_TIMER\_NUM0=0,

HAL\_TIMER\_NUM1,

HAL\_TIMER\_NUM2,

HAL\_TIMER\_NUM3,

HAL\_TIMER\_NUM4,

HAL\_TIMER\_NUM5,

HAL\_TIMER\_NUM6,

HAL\_TIMER\_NUM7,

HAL\_TIMER\_NUM8,

HAL\_TIMER\_NUM9,

HAL\_TIMER\_NUM10,

HAL\_TIMER\_NUM11,

HAL\_TIMER\_NUM12,

HAL\_TIMER\_NUM13,

HAL\_TIMER\_NUM14,

HAL\_TIMER\_NUM15,

HAL\_TIMER\_NUM16,

HAL\_TIMER\_NUM17,

HAL\_TIMER\_NUM18,

HAL\_TIMER\_NUM19,

HAL\_TIMER\_NUM20,

HAL\_TIMER\_NUM21,

HAL\_TIMER\_NUM22,

HAL\_TIMER\_NUM23

} ENUM\_HAL\_TIMER\_NUM;

#### ENUM\_HAL\_GPIO\_PinState

typedef enum

{

HAL\_GPIO\_PIN\_RESET = 0, //gpio output 0

HAL\_GPIO\_PIN\_SET //gpio output 1

} ENUM\_HAL\_GPIO\_PinState;

#### ENUM\_HAL\_GPIO\_InterrputLevel

typedef enum

{

HAL\_GPIO\_ACTIVE\_LOW = 0, //falling-edge or active-low sensitive

HAL\_GPIO\_ACTIVE\_HIGH //rising-edge or active-high sensitive

} ENUM\_HAL\_GPIO\_InterrputLevel;

#### ENUM\_HAL\_GPIO\_InterrputPolarity

typedef enum

{

HAL\_GPIO\_LEVEL\_SENUMSITIVE = 0,//level-interrupt

HAL\_GPIO\_EDGE\_SENUMSITIVE //edge-interrupt

} ENUM\_HAL\_GPIO\_InterrputPolarity;

#### ENUM\_HAL\_GPIO\_NUM

typedef enum

{

HAL\_GPIO\_NUM0 = 0,

HAL\_GPIO\_NUM1,

HAL\_GPIO\_NUM2,

HAL\_GPIO\_NUM3,

HAL\_GPIO\_NUM4,

HAL\_GPIO\_NUM5,

HAL\_GPIO\_NUM6,

HAL\_GPIO\_NUM7,

HAL\_GPIO\_NUM8,

HAL\_GPIO\_NUM9,

HAL\_GPIO\_NUM10,

HAL\_GPIO\_NUM11,

HAL\_GPIO\_NUM12,

HAL\_GPIO\_NUM13,

HAL\_GPIO\_NUM14,

HAL\_GPIO\_NUM15,

HAL\_GPIO\_NUM16,

HAL\_GPIO\_NUM17,

HAL\_GPIO\_NUM18,

HAL\_GPIO\_NUM19,

HAL\_GPIO\_NUM20,

HAL\_GPIO\_NUM21,

HAL\_GPIO\_NUM22,

HAL\_GPIO\_NUM23,

HAL\_GPIO\_NUM24,

HAL\_GPIO\_NUM25,

HAL\_GPIO\_NUM26,

HAL\_GPIO\_NUM27,

HAL\_GPIO\_NUM28,

HAL\_GPIO\_NUM29,

HAL\_GPIO\_NUM30,

HAL\_GPIO\_NUM31,

HAL\_GPIO\_NUM32,

HAL\_GPIO\_NUM33,

HAL\_GPIO\_NUM34,

HAL\_GPIO\_NUM35,

HAL\_GPIO\_NUM36,

HAL\_GPIO\_NUM37,

HAL\_GPIO\_NUM38,

HAL\_GPIO\_NUM39,

HAL\_GPIO\_NUM40,

HAL\_GPIO\_NUM41,

HAL\_GPIO\_NUM42,

HAL\_GPIO\_NUM43,

HAL\_GPIO\_NUM44,

HAL\_GPIO\_NUM45,

HAL\_GPIO\_NUM46,

HAL\_GPIO\_NUM47,

HAL\_GPIO\_NUM48,

HAL\_GPIO\_NUM49,

HAL\_GPIO\_NUM50,

HAL\_GPIO\_NUM51,

HAL\_GPIO\_NUM52,

HAL\_GPIO\_NUM53,

HAL\_GPIO\_NUM54,

HAL\_GPIO\_NUM55,

HAL\_GPIO\_NUM56,

HAL\_GPIO\_NUM57,

HAL\_GPIO\_NUM58,

HAL\_GPIO\_NUM59,

HAL\_GPIO\_NUM60,

HAL\_GPIO\_NUM61,

HAL\_GPIO\_NUM62,

HAL\_GPIO\_NUM63,

HAL\_GPIO\_NUM64,

HAL\_GPIO\_NUM65,

HAL\_GPIO\_NUM66,

HAL\_GPIO\_NUM67,

HAL\_GPIO\_NUM68,

HAL\_GPIO\_NUM69,

HAL\_GPIO\_NUM70,

HAL\_GPIO\_NUM71,

HAL\_GPIO\_NUM72,

HAL\_GPIO\_NUM73,

HAL\_GPIO\_NUM74,

HAL\_GPIO\_NUM75,

HAL\_GPIO\_NUM76,

HAL\_GPIO\_NUM77,

HAL\_GPIO\_NUM78,

HAL\_GPIO\_NUM79,

HAL\_GPIO\_NUM80,

HAL\_GPIO\_NUM81,

HAL\_GPIO\_NUM82,

HAL\_GPIO\_NUM83,

HAL\_GPIO\_NUM84,

HAL\_GPIO\_NUM85,

HAL\_GPIO\_NUM86,

HAL\_GPIO\_NUM87,

HAL\_GPIO\_NUM88,

HAL\_GPIO\_NUM89,

HAL\_GPIO\_NUM90,

HAL\_GPIO\_NUM91,

HAL\_GPIO\_NUM92,

HAL\_GPIO\_NUM93,

HAL\_GPIO\_NUM94,

HAL\_GPIO\_NUM95,

HAL\_GPIO\_NUM96,

HAL\_GPIO\_NUM97,

HAL\_GPIO\_NUM98,

HAL\_GPIO\_NUM99,

HAL\_GPIO\_NUM100,

HAL\_GPIO\_NUM101,

HAL\_GPIO\_NUM102,

HAL\_GPIO\_NUM103,

HAL\_GPIO\_NUM104,

HAL\_GPIO\_NUM105,

HAL\_GPIO\_NUM106,

HAL\_GPIO\_NUM107,

HAL\_GPIO\_NUM108,

HAL\_GPIO\_NUM109,

HAL\_GPIO\_NUM110,

HAL\_GPIO\_NUM111,

HAL\_GPIO\_NUM112,

HAL\_GPIO\_NUM113,

HAL\_GPIO\_NUM114,

HAL\_GPIO\_NUM115,

HAL\_GPIO\_NUM116,

HAL\_GPIO\_NUM117,

HAL\_GPIO\_NUM118,

HAL\_GPIO\_NUM119,

HAL\_GPIO\_NUM120,

HAL\_GPIO\_NUM121,

HAL\_GPIO\_NUM122,

HAL\_GPIO\_NUM123,

HAL\_GPIO\_NUM124,

HAL\_GPIO\_NUM125,

HAL\_GPIO\_NUM126,

HAL\_GPIO\_NUM127

} ENUM\_HAL\_GPIO\_NUM;

#### ENUM\_HAL\_USB\_HOST\_STATE

typedef enum

{

HAL\_USB\_HOST\_STATE\_IDLE = 0,

HAL\_USB\_HOST\_STATE\_READY,

HAL\_USB\_HOST\_STATE\_DISCONNECT,

} ENUM\_HAL\_USB\_HOST\_STATE;

#### ENUM\_HAL\_USB\_HOST\_CLASS

typedef enum

{

HAL\_USB\_HOST\_CLASS\_MSC = 0,

HAL\_USB\_HOST\_CLASS\_UVC,

} ENUM\_HAL\_USB\_HOST\_CLASS;

#### ENUM\_HAL\_USB \_PORT

typedef enum

{

HAL\_USB\_PORT\_0 = 0,

HAL\_USB\_PORT\_1,

} ENUM\_HAL\_USB\_PORT;

#### ENUM\_HAL\_SD\_CTRL

typedef enum

{

HAL\_SD\_GET\_CSD\_VERSION = 0,

HAL\_SD\_GET\_SECTOR\_COUNT,

HAL\_SD\_GET\_SECTOR\_SIZE,

HAL\_SD\_GET\_TRAN\_SPEED,

HAL\_SD\_GET\_CARD\_STATUS,

HAL\_SD\_GET\_MANUID,

HAL\_SD\_GET\_OEMID

} ENUM\_HAL\_SD\_CTRL;

#### ENUM\_RUN\_MODE

typedef enum

{

AUTO,

MANUAL

}ENUM\_RUN\_MODE;

#### ENUM\_BB\_MODE

typedef enum

{

BB\_SKY\_MODE = 0x00,

BB\_GRD\_MODE = 0x01,

} ENUM\_BB\_MODE;

#### ENUM\_RF\_BAND

typedef enum

{

RF\_2G = 0x00,

RF\_5G = 0x01,

}ENUM\_RF\_BAND;

#### ENUM\_BB\_QAM

typedef enum

{

MOD\_BPSK = 0x00,

MOD\_4QAM = 0x01,

MOD\_16QAM = 0x02,

MOD\_64QAM = 0x03,

}ENUM\_BB\_QAM;

#### ENUM\_BB\_LDPC

typedef enum

{

LDPC\_1\_2 = 0x00,

LDPC\_2\_3 = 0x01,

}ENUM\_BB\_LDPC;

#### ENUM\_CH\_BW

typedef enum

{

BW\_10M = 0x02,

BW\_20M = 0x00,

}ENUM\_CH\_BW;

#### ENUM\_BBUARTCOMSESSIONID

typedef enum

{

BB\_UART\_COM\_SESSION\_0 = 0,

BB\_UART\_COM\_SESSION\_1,

BB\_UART\_COM\_SESSION\_2,

BB\_UART\_COM\_SESSION\_3,

BB\_UART\_COM\_SESSION\_4,

BB\_UART\_COM\_SESSION\_MAX

} ENUM\_BBUARTCOMSESSIONID;

#### ENUM\_HAL\_NVIC\_IRQ\_NUM

typedef enum

{

HAL\_NVIC\_UART\_INTR0\_VECTOR\_NUM = 16, //16

HAL\_NVIC\_UART\_INTR1\_VECTOR\_NUM, //17

HAL\_NVIC\_UART\_INTR2\_VECTOR\_NUM, //18

HAL\_NVIC\_UART\_INTR3\_VECTOR\_NUM, //19

HAL\_NVIC\_UART\_INTR4\_VECTOR\_NUM, //20

HAL\_NVIC\_UART\_INTR5\_VECTOR\_NUM, //21

HAL\_NVIC\_UART\_INTR6\_VECTOR\_NUM, //22

HAL\_NVIC\_UART\_INTR7\_VECTOR\_NUM, //23

HAL\_NVIC\_UART\_INTR8\_VECTOR\_NUM, //24

HAL\_NVIC\_TIMER\_INTR00\_VECTOR\_NUM, //25

HAL\_NVIC\_TIMER\_INTR01\_VECTOR\_NUM, //26

HAL\_NVIC\_TIMER\_INTR02\_VECTOR\_NUM, //27

HAL\_NVIC\_TIMER\_INTR03\_VECTOR\_NUM, //28

HAL\_NVIC\_TIMER\_INTR04\_VECTOR\_NUM, //29

HAL\_NVIC\_TIMER\_INTR05\_VECTOR\_NUM, //30

HAL\_NVIC\_TIMER\_INTR06\_VECTOR\_NUM, //31

HAL\_NVIC\_TIMER\_INTR07\_VECTOR\_NUM, //32

HAL\_NVIC\_TIMER\_INTR10\_VECTOR\_NUM, //33

HAL\_NVIC\_TIMER\_INTR11\_VECTOR\_NUM, //34

HAL\_NVIC\_TIMER\_INTR12\_VECTOR\_NUM, //35

HAL\_NVIC\_TIMER\_INTR13\_VECTOR\_NUM, //36

HAL\_NVIC\_TIMER\_INTR14\_VECTOR\_NUM, //37

HAL\_NVIC\_TIMER\_INTR15\_VECTOR\_NUM, //38

HAL\_NVIC\_TIMER\_INTR16\_VECTOR\_NUM, //39

HAL\_NVIC\_TIMER\_INTR17\_VECTOR\_NUM, //40

HAL\_NVIC\_TIMER\_INTR20\_VECTOR\_NUM, //41

HAL\_NVIC\_TIMER\_INTR21\_VECTOR\_NUM, //42

HAL\_NVIC\_TIMER\_INTR22\_VECTOR\_NUM, //43

HAL\_NVIC\_TIMER\_INTR23\_VECTOR\_NUM, //44

HAL\_NVIC\_TIMER\_INTR24\_VECTOR\_NUM, //45

HAL\_NVIC\_TIMER\_INTR25\_VECTOR\_NUM, //46

HAL\_NVIC\_TIMER\_INTR26\_VECTOR\_NUM, //47

HAL\_NVIC\_TIMER\_INTR27\_VECTOR\_NUM, //48

HAL\_NVIC\_SSI\_INTR\_N0\_VECTOR\_NUM, //49

HAL\_NVIC\_SSI\_INTR\_N1\_VECTOR\_NUM, //50

HAL\_NVIC\_SSI\_INTR\_N2\_VECTOR\_NUM, //51

HAL\_NVIC\_SSI\_INTR\_N3\_VECTOR\_NUM, //52

HAL\_NVIC\_SSI\_INTR\_N4\_VECTOR\_NUM, //53

HAL\_NVIC\_SSI\_INTR\_N5\_VECTOR\_NUM, //54

HAL\_NVIC\_SSI\_INTR\_N6\_VECTOR\_NUM, //55

HAL\_NVIC\_I2C\_INTR0\_VECTOR\_NUM, //56

HAL\_NVIC\_I2C\_INTR1\_VECTOR\_NUM, //57

HAL\_NVIC\_I2C\_INTR2\_VECTOR\_NUM, //58

HAL\_NVIC\_I2C\_INTR3\_VECTOR\_NUM, //59

HAL\_NVIC\_CAN\_IRQ0\_VECTOR\_NUM, //60

HAL\_NVIC\_CAN\_IRQ1\_VECTOR\_NUM, //61

HAL\_NVIC\_CAN\_IRQ2\_VECTOR\_NUM, //62

HAL\_NVIC\_CAN\_IRQ3\_VECTOR\_NUM, //63

HAL\_NVIC\_WDT\_INTR0\_VECTOR\_NUM, //64

HAL\_NVIC\_WDT\_INTR1\_VECTOR\_NUM, //65

HAL\_NVIC\_GPIO\_INTR\_N0\_VECTOR\_NUM, //66

HAL\_NVIC\_GPIO\_INTR\_N1\_VECTOR\_NUM, //67

HAL\_NVIC\_GPIO\_INTR\_N2\_VECTOR\_NUM, //68

HAL\_NVIC\_GPIO\_INTR\_N3\_VECTOR\_NUM, //69

HAL\_NVIC\_I2C\_SLV\_INTR\_VECTOR\_NUM, //70

HAL\_NVIC\_RTC\_INTR\_VECTOR\_NUM, //71

HAL\_NVIC\_OTG\_INTR0\_VECTOR\_NUM, //72

HAL\_NVIC\_OTG\_INTR1\_VECTOR\_NUM, //73

HAL\_NVIC\_SD\_INTR\_VECTOR\_NUM, //74

HAL\_NVIC\_DMA\_INTR\_N\_VECTOR\_NUM, //75

HAL\_NVIC\_VIDEO\_UART9\_INTR\_VECTOR\_NUM, //76

HAL\_NVIC\_VIDEO\_ARMCM7\_IRQ\_VECTOR\_NUM, //77

HAL\_NVIC\_VIDEO\_UART10\_INTR\_VECTOR\_NUM, //78

HAL\_NVIC\_VIDEO\_I2C\_INTR\_VIDEO\_VECTOR\_NUM, //79

HAL\_NVIC\_VIDEO\_SSI\_INTR\_VIDEO\_VECTOR\_NUM, //80

HAL\_NVIC\_VIDEO\_WDT\_INTR2\_VECTOR\_NUM, //81

HAL\_NVIC\_VIDEO\_SPI\_INTR\_BB\_VECTOR\_NUM, //82

HAL\_NVIC\_VIDEO\_IMGWR\_FD\_CH1\_INTR\_VECTOR\_NUM, //83

HAL\_NVIC\_VIDEO\_IMGWR\_FD\_CH0\_INTR\_VECTOR\_NUM, //84

HAL\_NVIC\_VIDEO\_GLOBAL2\_INTR\_RES\_VSOC0\_VECTOR\_NUM, //85

HAL\_NVIC\_VIDEO\_GLOBAL2\_INTR\_RES\_VSOC1\_VECTOR\_NUM, //86

HAL\_NVIC\_BB\_SRAM\_READY\_IRQ\_0\_VECTOR\_NUM, //87

HAL\_NVIC\_BB\_SRAM\_READY\_IRQ\_1\_VECTOR\_NUM, //88

HAL\_NVIC\_BB\_TX\_ENABLE\_VECTOR\_NUM, //89

HAL\_NVIC\_BB\_RX\_ENABLE\_VECTOR\_NUM, //90

HAL\_NVIC\_CM7\_1\_FPU\_IRQ\_NUM0 = 91, //91

HAL\_NVIC\_CM7\_1\_FPU\_IRQ\_NUM1, //92

HAL\_NVIC\_CM7\_1\_FPU\_IRQ\_NUM2, //93

HAL\_NVIC\_CM7\_1\_FPU\_IRQ\_NUM3, //94

HAL\_NVIC\_CM7\_1\_FPU\_IRQ\_NUM4, //95

HAL\_NVIC\_CM7\_1\_FPU\_IRQ\_NUM5, //96

HAL\_NVIC\_CM7\_1\_CTIIRQ\_NUM0, //97

HAL\_NVIC\_CM7\_1\_CTIIRQ\_NUM1, //98

HAL\_NVIC\_CM7\_2\_FPU\_IRQ\_NUM0 = 91, //91

HAL\_NVIC\_CM7\_2\_FPU\_IRQ\_NUM1, //92

HAL\_NVIC\_CM7\_2\_FPU\_IRQ\_NUM2, //93

HAL\_NVIC\_CM7\_2\_FPU\_IRQ\_NUM3, //94

HAL\_NVIC\_CM7\_2\_FPU\_IRQ\_NUM4, //95

HAL\_NVIC\_CM7\_2\_FPU\_IRQ\_NUM5, //96

HAL\_NVIC\_CM7\_2\_CTIIRQ\_NUM0, //97

HAL\_NVIC\_CM7\_2\_CTIIRQ\_NUM1, //98

HAL\_NVIC\_CM7\_3\_FPU\_IRQ\_NUM0 = 91, //91

HAL\_NVIC\_CM7\_3\_FPU\_IRQ\_NUM1, //92

HAL\_NVIC\_CM7\_3\_FPU\_IRQ\_NUM2, //93

HAL\_NVIC\_CM7\_3\_FPU\_IRQ\_NUM3, //94

HAL\_NVIC\_CM7\_3\_FPU\_IRQ\_NUM4, //95

HAL\_NVIC\_CM7\_3\_FPU\_IRQ\_NUM5, //96

HAL\_NVIC\_CM7\_3\_CTIIRQ\_NUM0, //97

HAL\_NVIC\_CM7\_3\_CTIIRQ\_NUM1, //98

} ENUM\_HAL\_NVIC\_IRQ\_NUM;

#### ENUM\_HAL\_CAN\_COMPONENT

typedef enum

{

HAL\_CAN\_COMPONENT\_0 = 0,

HAL\_CAN\_COMPONENT\_1,

HAL\_CAN\_COMPONENT\_2,

HAL\_CAN\_COMPONENT\_3,

} ENUM\_HAL\_CAN\_COMPONENT;

#### ENUM\_HAL\_CAN\_BAUDR

typedef enum

{

HAL\_CAN\_BAUDR\_125K = 0,

HAL\_CAN\_BAUDR\_250K,

HAL\_CAN\_BAUDR\_500K,

HAL\_CAN\_BAUDR\_1M,

} ENUM\_HAL\_CAN\_BAUDR;

#### ENUM\_HAL\_CAN\_FORMAT

typedef enum

{

HAL\_CAN\_FORMAT\_STD= 0,

HAL\_CAN\_FORMAT\_EXT,

} ENUM\_HAL\_CAN\_FORMAT;

#### ENUM\_HAL\_CAN\_TYPE

typedef enum

{

HAL\_CAN\_TYPE\_DATA= 0,

HAL\_CAN\_TYPE\_RMT,

} ENUM\_HAL\_CAN\_TYPE;

#### STRU\_SoftPwmHandle

typedef struct

{

uint32\_t u32\_countArray[2]; //[0] pwm high [1] pwm low

uint32\_t u32\_overFlow;

uint32\_t u32\_baseTime;

uint8\_t u8\_polarity;

uint8\_t u8\_pin; // pwm gpio

HAL\_RET\_T(\*function)(ENUM\_HAL\_GPIO\_NUM,ENUM\_HAL\_GPIO\_PinState);//callbak

} STRU\_SoftPwmHandle;

#### STRU\_HAL\_SYS\_CTL\_CONFIG

typedef struct

{

uint16\_t u16\_cpu0cpu1Clk;

uint16\_t u16\_cpu2Clk;

uint8\_t u8\_fpuEnable;

uint16\_t u16\_sysTickIntervalUs;

uint8\_t u8\_workMode;

} STRU\_HAL\_SYS\_CTL\_CONFIG;

#### STRU\_HAL\_CAN\_CONFIG

typedef struct STRU\_HAL\_CAN\_CONFIG

{

ENUM\_HAL\_CAN\_COMPONENT e\_halCanComponent;

ENUM\_HAL\_CAN\_BAUDR e\_halCanBaudr;

uint32\_t u32\_halCanAcode; /\*std bit10~0 <-> ID10~0

ext bit28~0 <-> ID28~0\*/

uint32\_t u32\_halCanAmask; /\*std bit10~0 <-> ID10~0

ext bit28~0 <-> ID28~0\*/

ENUM\_HAL\_CAN\_FORMAT e\_halCanFormat;

HAL\_CAN\_RcvMsgHandler pfun\_halCanRcvMsg;

} STRU\_HAL\_CAN\_CONFIG;

#### STRU\_HAL\_CAN\_MSG

typedef struct STRU\_HAL\_CAN\_MSG

{

ENUM\_HAL\_CAN\_COMPONENT e\_halCanComponent;

uint32\_t u32\_halCanId;

uint8\_t u8\_halCanDataArray[8];

uint8\_t u8\_halCanDataLen;

ENUM\_HAL\_CAN\_FORMAT e\_halCanFormat;

ENUM\_HAL\_CAN\_TYPE e\_halCanType;

} STRU\_HAL\_CAN\_MSG;