目录

1.简述

1.1. 简介

Hyprintf 是专为嵌入式系统设计的轻量级 printf 替代方案。它在追求高效、低内存占用的同时，提供了简单易用的接口，旨在解决传统 printf 在资源受限环境中的问题。

Hyprintf 提供了类似于 printf 的接口，允许开发者以格式字符串的形式输出数据到串口或其他目标。通过嵌入式的设计，该库在保持简洁性的同时，仍然提供了一些基本的格式化输出能力以及特别的功能。

1.2. HYPRINTF 优势

- 低内存占用： Hyprintf 设计追求节省内存资源，特别适用于嵌入式系统等资源受限环境，将解析数据按发送空间拆分发送，用时间换内存空间。
- 高效发送：通过创建 printf 组，Hyprintf 允许将同一份数据轮流发送到不同的接口中，提升发送效率，避免频繁的内存操作。
- 用户 API：提供了一系列用户 API，允许根据具体需求配置不同的发送接口，使 Hyprintf 适应各种硬件和通信协议。
- 格式化输出：实现了部分 printf 的基础功能，支持用户格式化输出，使得 Hyprintf 能够灵活应对各种数据类型。
- 接口安全： 使用互斥锁、临界段保证了接口安全，可以在多线程环境下保护临界接口资源，避免被抢占。

1.3. HYPRINTF 数据类型

考虑到项目的移植要求，本项目仅用以下基本数据类型进行开发。

- char
- int
- uint32_t（已抽象为 hy_uint，可替换）
- double（已抽象为 hy_float，可替换）

1.4. HYPRINTF 功能

描述

发送格式化输出到各个接口中。

printf() 函数的调用格式为:

```
hprintf("<格式化字符串>", <参量表>);
```

声明

下面是 hyprintf() 函数的声明。

```
char hyprintf(const char *format, ...)
```

参数

- format -- 这是字符串，包含了要被写入到接口的文本。它可以包含嵌入的 format 标签，format 标签可被随后的附加参数中指定的值替换，并按需求进行格式化。format 标签属性是 %[flags][number][. number2][type]，具体讲解如下:

type(格式化类型)	意义
d	以十进制形式输出带符号整数(正数不输出符号)
b	以二进制形式输出无符号整数(不输出前缀 0b)
o	以八进制形式输出无符号整数(不输出前缀 0)
x	以十六进制形式输出无符号整数(不输出前缀 0x)
X	以十六进制与大写字母形式输出无符号整数(不输出前缀 0X)
u	以十进制形式输出无符号整数
f	以小数形式输出单、双精度实数
c	输出单个字符
s	输出字符串，遇到'\0'结束符终止
S	输出数组数据，遇到'\0'结束符不会终止
p	输出指针地址

flags(标识)	描述
-	在给定的字段宽度内左对齐，默认是右对齐（参见 number 子说明符）。
+	强制在结果之前显示加号或减号（+ 或 -），即正数前面会显示 + 号。默认情况下，只有负数前面会显示一个 - 号。
#	与 b、o、x 或 X 说明符一起使用时，非零值前面会分别显示 0b、0、0x 或 0X。
0	将填充字符设置为零（0），而不是空格（参见 number 子格式化类型）。

number（参数1）	描述
(number)	要输出的字符的最小数目。如果输出的值短于该数，结果会用空格或字符'0'填充。如果输出的值长于该数，结果不会被截断。 对于 s 格式化类型：表示在字符串的前后要填充的字符个数 对于 S 格式化类型：表示每个元素的字节数，默认情况下此值为 0。

.number2（参数2）	描述
(.number2)	对于 f 格式化类型：表示为要在小数点后输出的小数位数。 对于 s 格式化类型：表示要输出的最大字符数。默认情况下，所有字符都会被输出，直到遇到末尾的空字符。 对于 S 格式化类型：在 number 为默认值 0 或被设定的值为 1 时，表示要输出的数组字节个数；反之表示元素个数。

other（其它）	描述
%	将直接输出百分号字符本身。
*	退出此次'%'的格式化处理，继续读取后续字符。

2.环境要求

2.1. 基本要求

如果你想要使用 HYPRINTF，请确保环境中满足以下基本要求：

- 系统环境： 确保你的目标系统支持 C 语言编程，并具备足够的资源（内存、处理器性能等）来运行该项目；
- 编译器与标准库： 使用适合目标平台系统的 C 编译器。确保目标系统支持标准库里的 <stdarg.h>；
- 操作系统：项目可以运行在裸机环境或操作系统中。如有运行在操作系统的需求，确保在 hyprintf.h 中正确使能、配置相关宏。

2.2. 可变类型

请在 hyprintf.h 中正确设置以下宏，设置为当前 C 编译器中用于处理可变参数对应宏。

```
#include <stdarg.h>                    // 提供变参宏及函数
...
#define hy_va_list                     va_list
#define hy_va_start(p_args,arg)        va_start(p_args,arg)
#define hy_va_arg(p_args,arg)          va_arg(p_args,arg)
#define hy_va_end(p_args)              va_end(p_args)
```

2.3. 无符号整型

请在 hyprintf.h 中正确修改以下宏，设置为当前 C 编译器与处理器可适用的中无符号的整数类型。

```
#include “stm32f10x.h”                // 提供无符号整型类型
...
#ifndef HY_UNSIGNED_INT
  #define HY_UNSIGNED_INT              // 默认启用无符号整型
#endif
#ifdef HY_UNSIGNED_INT                 // 如果存在无符号整型，请填写对应类型
```

```
  #define hy_uint                    uint32_t
#else
  #define hy_uint                    int
#endif
```

此段定义中，存在一个启用无符号整型的宏——HY_UNSIGNED_INT。如果编译器或目标 CPU 存在——不适用无符号整型的情况，只需将“HY_UNSIGNED_INT”此宏注释或删除，此举会将此项目中的所有无符号整型都将以有符号整型的形式表达。

（注意：请确保无符号整型的位数大于或等于 int 的位数。）

2.4. CPU 字长与 CPU 适用的浮点类型

请在 hyprintf.h 中正确修改以下宏，设置为目标 CPU 的 int 字长，同时设置目标 CPU 适用的浮点类型。（由于开发能力有限，暂且只实现了一种浮点类型的格式化处理）

```
#define HY_INT_SIZE                sizeof(int)          // 或者填为 4
#define hy_float                   double
```

2.5. 总结

以上的配置部分依赖于用户使用的编译器以及运行该项目的目标 CPU，请认真核查用户的编译器以及目标 CPU。

3.基本构成及设置

3.1. 基本组成

该项目使用了一个枚举数组——HYPRINTF_SEND_TYPE，该枚举数组的各个枚举成员将表示为不同的接口（可以是硬件接口或是软件接口）。作为一个轻量化的项目，每个成员的最终表示的本质是一个整型的某个位。如果项目运行在 32 位整型的 CPU 中，至多可以容纳 31 个成员（其中符号位也就是第 31 位，用于区分成员与组）。

以下是为 STM32F10x 编写的枚举数组范例

```
typedef enum{
```

```
    HYPRINTF_UART1,
    HYPRINTF_UART2,

    /* ***
        下列成员具有特殊作用，可按需求使用下列标识符，请谨慎修改
    *** */

#ifdef HYPRINTF_JUMP_FLAG
    HYPRINTF_JUMP,          // 越锁成员，占用一个成员空间
#endif

    HYPRINTF_SEND_TYPE_SIZE,// 此元素不做成员成分，仅用于表示枚举成员的个数
}HYPRINTF_SEND_TYPE;
```

（注意：关于越锁成员，请跳转至 3.2 用户环境-越锁成员）

其中枚举成员——HYPRINTF_SEND_TYPE_SIZE 用于表示枚举成员的个数

在该枚举数组中，使用了 UART1 和 UART2 作为成员，而后需到文件 hyprintf.c 的函数 hyprintf_api()内根据不同的成员配置不同的接口函数，如下所示：

```
void hyprintf_api(int send_type,int send_size,const char *send_str){
    // 只做一次判断，请使用 if(...)else if(...)else(...) 的结构以便省去多余的判断时间

    // 请编写接口函数
    if( send_type==HYPRINTF_UART1 ){
        Usart_SendArray(USART1,send_str,send_size);
    }
    else if( send_type==HYPRINTF_UART2 ){
        Usart_SendArray(USART2,send_str,send_size);
    }
    else{
```

```
    }
}
```

至此，不同的枚举成员都将对应着函数 hyprintf_api()内的不同接口，用户仅需根据这些枚举成员就可以将格式化数据 pirntf 至不同的接口中。

3.2. 功能设置

是的，该项目将配置分成为两部分，一是针对编译器以及目标 CPU 的配置，这是 HYPRINTF 能否编译通过以及可否运行的基石，已在之前的篇章中提及。其次就是本小节的功能设置部分。

基本参数设置

用户可以通过给以下宏定义值来适应项目需求：

```
#define HYPRINTF_MAX_STRLEN                 32
#define HYPRINTF_PARAMETER_SIZE             12
#define HYPRINTF_FLOAT_FIXSIZE              8
#define HYPRINTF_FLOAT_NUMERATION           10
```

- HYPRINTF_MAX_STRLEN：表示存储格式化结果的数据空间大小，其中最后一字节固定用于存放 '\0' 避免字符串溢出、拷贝问题，确保字符串处理函数正确性和准确计算长度。当前值为 32 意味着每当处理结果达到 31 个字节时就会执行一次发送操作。
- HYPRINTF_PARAMETER_SIZE：表示 % 后可携带参数的最大长度。该宏的在项目的功能函数中以 char 类型赋值给某些变量，范围在 0 到 255 之间。
- HYPRINTF_FLOAT_FIXSIZE：默认输出浮点数时小数点后的位数。此值还作为有效位数，即可打印的浮点位数不会超过这个值。该宏的在项目的功能函数中以 char 类型赋值给某些变量，范围在 0 到 255 之间。
- HYPRINTF_FLOAT_NUMERATION：表示浮点数的输出为 10 进制数。

函数 hyprintf() 的保护策略

以下宏关于是否使能——保护 hyprintf()：

```
#ifndef HYPRINTF_PROTECT
  #define HYPRINTF_PROTECT
#endif
```

定义此宏，将使用互斥锁保护 hyprintf() 所配置的接口。一个任务使用 hyprintf() 发送数据前将进行上锁，在解锁前，也就是未完成发送时，（若设置为 OS 模式）其他线程或者中断任务将无法使用此接口。

需特别说明：在裸机环境中，若中断发生在——任务判断未上锁后，准备上锁前，锁将失效，也就是存在竞态条件。不过在裸机环境中，这种竞态条件并不会对主程序产生恶劣影响。如果有在逻辑环境下保护 hyprintf()接口的需求，请跳转至 3.2 功能设置-操作系统

越锁成员

以下宏定义为实现一种越锁的操作：

```
#ifndef HYPRINTF_JUMP_FLAG
    #define HYPRINTF_JUMP_FLAG
#endif
```

定义此宏后，会同时在枚举数组中定义一个枚举成员，如下所示：“

```
#ifdef HYPRINTF_JUMP_FLAG
    HYPRINTF_JUMP,          // 无条件越锁标识符，占用一位
#endif
```

这个枚举成员会占用一个接口空间，若该项目运行在 32 位整型的 CPU 上，除去符号位与该越锁成员，仅剩下 30 个成员可供用户定义。

HYPRINTF_JUMP 成员一般不作为接口成员在 hyprintf_api 中配置对应的接口函数（当然也可以配置）。它一般用于使用函数 hyprintf_set_group 与其他成员绑定成一个 printf 组，此时该组可以使用函数 hyprintf_by 执行发送任务，无论上锁与否。详情请见后续补充。

操作系统

以下宏涉及该项目是否运行在操作系统上：

```
#ifndef HYPRINTF_IN_OS
 #define HYPRINTF_IN_OS
```

```
#endif


#ifdef HYPRINTF_IN_OS
// 对接 进入临界区以及退出临界区的函数
    #define hy_enter_critical()          rt_enter_critical()
    #define  hy_exti_critical()          rt_exit_critical()
#else
    #define hy_enter_critical()          __set_PRIMASK(1)  // 如果需要避免竞态条件
    #define  hy_exti_critical()          __set_PRIMASK(0)
#endif
```

如上所示，如有将该项目运行在操作系统上的需求，可以定义 HYPRINTF_IN_OS 宏，将用户操作系统（如 RT-Thread）临界段操作的函数填写在宏 hy_enter_critical() 以及 hy_exti_critical()。此操作将可以避免竞态条件的发送。

如果有在裸机系统中避免竞态条件的存在的需求，请在 #else 下填写目标处理器（如 STM32F10x）关闭中断以及开启中断的函数。

4.接口函数介绍

4.1. 简介

本章将介绍接口及功能函数，不涉及内部函数。用户仅需了解这些函数就可以将一个低内存、安全、多功能的类 printf 项目嵌入至用户的软件中。

4.2. 函数返回类型

```
#define HY_RET_T                    char
```

本项目用一个 char 类型作为函数的返回结果，并将返回结果以枚举数组 HYPRINTF_RET_TYPE 做区分，用户根据返回类型判断程序是否成功执行，以及错误返回类型。下列是枚举数组 HYPRINTF_RET_TYPE：

```
typedef enum{

    HY_RET_FALSE=0,
    HY_RET_TRUE=1,

    HY_RET_INT_ERR,                 // 整型定义错误
    HY_RET_RAM_ERR,                 // 数据空间大小错误
    HY_RET_ENUM_ERR,                // 枚举成员数错误
    HY_RET_PARAMETER_SIZE_ERR,      // 可处理参数大小错误
    HY_RET_FLOAT_SIZE_ERR,          // 默认打印浮点数小数点后的个数错误
    HY_RET_FLOAT_NUM_ERR,           // 浮点数的数进制错误
    HY_RET_ARG_ERR,                 // 参数错误
    HY_RET_LOCKED,                  // 已上锁，无法执行发送任务

}HYPRINTF_RET_TYPE;
```

4.3. 函数 hyprintf_init()

函数说明

此函数用于检查用户的宏定义是否存在错误，若存在错误将返回对应的错误类型。用户可根据返回的错误类型在 HYPRINTF_RET_TYPE 枚举数组或源码中找到错误点。

函数原型

HY_RET_T hyprintf_init(void)

参数

无

返回值

成功时，返回 HY_RET_TRUE；失败时，返回错误类型码。

注意事项

可在函数内更改 send_type 变量的值从而更改 hyprintf 的接口。

函数调用示例

程序	描述
HY_RET_T err = hyprintf_init();	// 检查用户的宏定义是否存在错误

4.4. 函数 hyprintf()

函数说明

将格式化后的字符串输出至各接口。

函数原型

HY_RET_T hyprintf(const char *format, ...)

参数

format（输入）：指定输出格式的字符串。

...（输入）：可变参数列表，包含要格式化并输出的数据。

返回值

成功时，返回 HY_RET_TRUE；失败时，返回错误类型。

函数调用示例

已在 1.4-HYPRINTF 功能中介绍了 hyprintf 可使用的格式化类型，以下是调用范例：

程序	注释或输出结果
HY_RET_T err = hyprintf_init(); if(err!=HY_RET_TRUE){ hyprintf("\nerr is %d",err);while(1); }	// 用于检查宏定义是否正确
hyprintf("\n:%d:",30);	:30:

hyprintf("\n:%d:",-30);
hyprintf("\n:%+d:",30);
hyprintf("\n:%10d:",30);
hyprintf("\n:%3d:",3000);
hyprintf("\n:%-10d:",30);
hyprintf("\n:%010d:",30);
hyprintf("\n:%+-010d:",30);
	:-30:
	:+30:
	: 30:
	:3000:
	:30 :
	:0000000030:
	:+300000000:

hyprintf("\n:%b:",30);
hyprintf("\n:%o:",30);
hyprintf("\n:%x:",30);
hyprintf("\n:%X:",30);
	:11110:
	:36:
	:1e:
	:1E:

hyprintf("\n:%#b:",30);
hyprintf("\n:%#o:",30);
hyprintf("\n:%#x:",30);
hyprintf("\n:%#X:",30);
	:0b11110:
	:036:
	:0x1e:
	:0X1E:

hyprintf("\n:%u:",0xFFFFFFFF);
hyprintf("\n:%u:",-30);
hyprintf("\n:%+u:",-30);
	:4294967295:
	:4294967266:
	:+4294967266:

hyprintf("\n:%f:",123456.123456789); hyprintf("\n:%.5f:",123456.123456789);	:123456.12345678: :123456.12345:

int temp_p=0;
hyprintf("\n:%p:",&temp_p);
hyprintf("\n:%08p:",&temp_p);
	// 定义变量
	:2000040C:
	:2000040C:

hyprintf("\n:%c:",'a'); hyprintf("\n:%-10c:",'a');	:a: :a :

hyprintf("\n:%s:","09AZaz");
hyprintf("\n:%10s:","09AZaz");
hyprintf("\n:%.3s:","09AZaz");
hyprintf("\n:%-010.3s:","09AZaz");
	:09AZaz:
	: 09AZaz:
	:09A:
	:09A0000000000:

int32_t i32[2] = {0x31323334,0x35363738};	// 定义一个 32 位的整型数组
hyprintf("\n:%.8S:",i32);	:43218765:
hyprintf("\n:%4.2S:",i32);	:12345678:
hyprintf("\n:%8.1S:",i32);	:56781234:
int16_t i16[3] = {0x3132,0x3334,0x3536};	// 定义一个 16 位的整型数组
hyprintf("\n:%2.2S:",&i16[1]);	:3456:

4.5. 函数 hyprintf_echo()

函数说明

将格式化后的字符串返回至指定的字符串缓冲区。、

函数原型

HY_RET_T hyprintf_echo(int ret_str_size,char *ret_str,const char *format,...)

参数

ret_str_size（输入）：存放格式化结果的数据空间大小。

ret_str（输出）：指向存放格式化结果的字符串的指针。

format（输入）：指向需进行格式化的字符串的指针。

...（输入）：变长参数列表，为格式化字符串的可变参数。

返回值

函数执行结果，返回错误码（HY_RET_T 类型），成功返回 HY_RET_TRUE。

注意事项

函数调用示例

下面范例中，将格式化结果直接返回至数组内。

```
// 已通过 hyprintf_init() 检查
```

```
char ret_str[16];
HY_RET_T err = hyprintf_echo(16,ret_str,"%s","Hello World!");
hyprintf("\n:%s:",ret_str);
```

```
:Hello World!:
// 用于返回的数组大小为 16
// 可用变量存储返回值，函数执行成功将会返回 HY_RET_TRUE
```

```
char ret_str2[4];
hyprintf_echo(4,ret_str2,"%s","Hello World!");
hyprintf("\n:%s:",ret_str2);
```

```
:ld!:
// 用于返回的数组大小为 16
// 若格式化结果大于可用于返回的数
// 组大小，则会覆盖结果
```

4.6. 函数 hyprintf_set_group()

函数说明

该函数用于创建一个 hyprintf 组，将格式化结果轮流发送至组内的不同成员。需要注意的是，组内任一成员未通过锁都将无法发送。

若使能越锁功能，可用越锁成员在创建 hyprintf 组时作为组成员，此时该组都可无条件越锁打印格式化结果至接口。

本质：之前提到，本项目用 32 位整型的位来表示不同的枚举成员。如果成员值为 1，那么意味着它所表示的位为 1 左移 1 位；如果成员值为 2，那么意味着它所表示的位为 1 左移 2 位。

函数原型

HY_RET_T hyprintf_set_group(int *group,int group_num,...)

参数

group（输入输出）：指向表示 hyprintf 组的整数指针，用于存储组成员的发送类型标志位。

group_num（输入）：传递的参数个数，表示待创建的 hyprintf 组成员个数，待加入的成员可以是组。

...（输入）：变长参数列表，为待创建的 hyprintf 组成员的发送类型。每个参数应为整数，表示发送类型的索引。

返回值

函数执行结果，返回错误码（HY_RET_T 类型），成功返回 HY_RET_TRUE。

注意事项

- 如果 group_num 不为枚举数组 HYPRINTF_SEND_TYPE 的枚举成员，则直接跳过。

函数调用示例

```
// 已通过 hyprintf_init() 检查
```

```
hyprintf_by(HYPRINTF_UART1,"\nU1:%s:","Hello World!");
hyprintf_by(HYPRINTF_UART2,"\nU2:%s:","Hello World!");
```

```
（UART1 发送）U1:Hello World!:
（UART2 发送）U2:Hello World!:
```

```
// 初值不能为负数
int HYPRINTF_UART12=0;
err = hyprintf_set_group(&HYPRINTF_UART12, 2, HYPRINTF_UART1, HYPRINTF_UART2);
if(err!=HY_RET_TRUE){ hyprintf("\nerr is %d",err);while(1); }
else{ hyprintf("\ngroup is %d=-%#b",HYPRINTF_UART12,-HYPRINTF_UART12); }
```

```
group is -3=-0b11
// 表示该组成员的有枚举成员 0 和枚举成员 1.（成员下标左移 n 位）
```

```
hyprintf_by(HYPRINTF_UART12,"\nset U1+U2:%s:","Hello World!");
```

```
（UART1 发送）set U1+U2:Hello World!:
（UART2 发送）set U1+U2:Hello World!:
```

4.7. 函数 hyprintf_leave_group()

函数说明

该函数用于创建一个 hyprintf 组，将格式化结果轮流发送至组内的不同成员。需要注意的是，组内任一成员未通过锁都将无法发送。

若使能越锁功能，可用越锁成员在创建 hyprintf 组时作为组成员，此时该组都可无条件越锁打印格式化结果至接口。

函数说明

HY_RET_T hyprintf_set_group(int* group, int group_num, ...);

参数

group（输入输出）：指向表示 hyprintf 组的整数指针，用于存储组成员的发送类型标志位。

group_num（输入）：传递的参数个数，表示待创建的 hyprintf 组成员个数，待加入的成员可以是组。

...（输入）：变长参数列表，为待创建的 hyprintf 组成员的发送类型。每个参数应为整数，表示发送类型的索引。

返回值

函数执行结果，返回错误码（HY_RET_T 类型），成功返回 HY_RET_TRUE。

注意事项

如果 group_num 不为枚举数组 HYPRINTF_SEND_TYPE 的枚举成员，则直接跳过。

函数调用示例

```
// 已通过 hyprintf_init() 检查
// 此时 HYPRINTF_UART12 已有成员 HYPRINTF_UART1 和 HYPRINTF_UART2
err = hyprintf_leave_group(&HYPRINTF_UART12, 1, HYPRINTF_UART1);
if(err!=HY_RET_TRUE){ hyprintf("\nerr is %d",err);while(1); }
```

else{ hyprintf("\ngroup is %d=-%#b",HYPRINTF_UART12,-HYPRINTF_UART12); }
group is -1=-0b1 // 表示目前的组成员有枚举成员 0（成员下标左移 n 位）
hyprintf_by(HYPRINTF_UART12,"\nleave U1:%s:","Hello World!");
（仅 UART1 发送）leave U1:Hello World!:

4.8. 越锁成员使用示例

// 将解锁代码注释，表示在上锁后将一直处于锁定状态
hyprintf_by(HYPRINTF_UART1,"\nU1-1:%s:","Hello World!"); hyprintf_by(HYPRINTF_UART1,"\nU1-2:%s:","Hello World!");

U1-1:Hello World!:
The current channel is locked, and the current lock value is 0b1
// 上述语句会在不开启 hyprintf()保护，也就是不定义 HYPRINTF_PROTECT 宏时输出
// 表示当前通道已上锁，上锁的为枚举成员 0（成员下标左移 n 位）

int HYPRINTF_UART1J=0;
err = hyprintf_set_group(&HYPRINTF_UART1J, 2, HYPRINTF_UART1, HYPRINTF_JUMP);
if(err!=HY_RET_TRUE){ hyprintf("\nerr is %d",err);while(1); }
else{ hyprintf("\ngroup is %d=-%#b",HYPRINTF_UART1J,-HYPRINTF_UART1J); }

group is -5=-0b101 // 表示将枚举成员 HYPRINTF_UART1 与 HYPRINTF_JUMP 组队
hyprintf_by(HYPRINTF_UART1J,"\nU1-3:%s:","Hello World!");
（UART1 发送）U1-3:Hello World!: // 此时可以越锁完成发送任务
hyprintf("\nU1-4:%s:","Hello World!");
U1-4:Hello World!: // 默认不保护 hyprintf 的接口，因此可以完成发送

int HYPRINTF_USART12=0;
hyprintf_set_group(&HYPRINTF_USART12, 2, HYPRINTF_UART1, HYPRINTF_UART2);
hyprintf_by(HYPRINTF_USART12,"\nU1+U2-5:%s:","Hello World!");

The current channel is locked, and the current lock value is 0b1 // 由于成员 HYPRINTF_UART1 已被上锁，因此无法完成该组的发送任务
hyprintf_set_group(&HYPRINTF_USART12, 1, HYPRINTF_JUMP); hyprintf_by(HYPRINTF_USART12,"\nU1+U2-6:%s:","Hello World!");
U1+U2-6:Hello World!: // 由于组加入了越缩成员，因此该组可以完成发送任务
hyprintf_leave_group(&HYPRINTF_USART12,1,HYPRINTF_JUMP); hyprintf_by(HYPRINTF_USART12,"\nU1+U2-7:%s:","Hello World!");
The current channel is locked, and the current lock value is 0b1 // 当越锁成员离开后，该组就无法完成发送任务

5.结语

受限于开发者的技术水平，目前此项目可能仅兼容部分 C 编译器，如需扩展、完善等需求或有不错的 idea，可联系作者，联系邮箱：h3314829906@163.com.

开发此项目的目的是，在嵌入式开发过程中，存在着许多的通信需求，不论是任务对硬件的，还是任务对任务的（线程对线程的）。这些对话不一定依赖 hex 格式来进行交流，如在使用 UART 对外部设备进行 AT 配置或交流，或者对上位机发送可视信息。而 printf 往往只能用于单个接口的格式化输出，给多接口需求的开发带来了些许不便。此项目其一目的是为了解决这些问题，那还有的就是根据过往项目经历中的一些常见需求，做的一些简单设计。希望这些设计能成为我或是你在日后的漫长开发路程的一项有力工具。

如果将来我有更多的空闲时间，可能会提供一些实实在在的简单项目，供开发者可以更好的理解。

至此，感谢你的阅读。