摘 要

本应用笔记，旨在帮助客户在使用不同 IDE（MDK Keil 或 IAR）时，对使用 printf 函数来打印输出 UART 串口信息时遇到的无法打印、打印乱码等移植问题作出梳理，理清不同 IDE 对 printf 函数支持的差异。并在本应用笔记结尾处给出可以一键移植到 Keil、IAR8.x、IAR9.x 版本下的通用重定向代码。

Keil 和 IAR 都是常用的开发工具 IDE，在实际项目开发和调试中，printf 函数是常用的打印函数，通常通过 fputc 接口的重定向，即可以通过 printf 输出串口的信息。

但在 IAR8.x 下，需要一些 IDE 的额外配置，MCU 才能正确通过 UART 口输出打印信息。在 IAR9.x 版本下，IAR 官方已经不再支持 fputc 接口的重定向，此时若移植原 Keil 下代码，则无法在 IAR 下编译通过。

目前，IAR Systems 最新发布的集成开发环境 IAR Embedded Workbench for Arm 9.32 版本已全面支持芯海科技 32 位 MCU 芯片，其 IAR Systems 为芯海科技 CS32 系列 MCU 提供完整的工具链。因此，本应用笔记将重点说明 IAR8.x 和 9.x 版本对 printf 函数的重定向的差异。并在本应用笔记结尾处给出可以一键移植到 Keil、IAR8.x、IAR9.x 版本下的通用重定向代码。

适用范围

类型	适用产品型号或系列	说明
MCU	所有 ARM 内核的 MCU，包括：CS32F0 系列，CS32F1 系列，CS32L0 系列等	

版 本

历史版本	修改内容	日期
V1.0	初版生成	2023-02-16

目 录

1. 概述

Keil 和 IAR 都是常用的开发工具 IDE，在实际项目开发和调试中，printf 函数是常用的打印函数，通常通过 fputc 接口的重定向，即可以通过 printf 输出串口的信息。

但在 IAR8.x 下，需要一些 IDE 的额外配置，MCU 才能正确通过 UART 口输出打印信息。在 IAR9.x 版本下，IAR 官方已经不再支持 fputc 接口的重定向，此时若移植原 Keil 下代码，则无法在 IAR 下编译通过。

目前，IAR Systems 最新发布的集成开发环境 IAR Embedded Workbench for Arm 9.32 版本已全面支持芯海科技 32 位 MCU 芯片，其 IAR Systems 为芯海科技 CS32 系列 MCU 提供完整的工具链。

本应用笔记将会详细介绍 Keil 下和 IAR 下实现 printf 重定向到串口的方法，并给出实现代码。所提到的方法可以通用于所有 ARM 内核的 MCU，包括：CS32F0 系列，CS32F1 系列，CS32L0 系列等，给出的代码中所涉及到的外设驱动接口以 CS32F03x 为例，用户可以根据不同的芯片平台和驱动 SDK 稍加替换即可移植到自己的项目代码中。

表 1　使用环境说明

类别	名称	下载路径
MCU	CS32F036	/
开发板	EVB_CS32F035_036_Start V1.1	/
下载工具	Jlink	/
SDK	ChipSea.CS32F0XX_DFP.1.0.8	/
编译工具	Keil uVision V5.27.1.0 /IAR9.20/IAR8.2x	/

2. Keil 下 printf 重定向到串口的方法

Keil 下有两种实现 printf 重定向到外设串口的方法，一种利用标准 c 库，一种利用轻量型 MicroLIB 库。

两种情况各有利弊，使用标准库跨平台会好些。使用 MicroLIB 库简单、运行效率高。

使用标准库默认会启动半主机模式，同时需要重定向 fputc 函数。在用标准库重定向 fputc 的时候，必须主动声明禁用 ARM 芯片的半主机模式，然后根据自己的芯片平台和 USART 外设特性来调用相关的发送接口。因此，概括来说，在 Keil 下使用标准 c 库来将 printf 重定向到串口时，需要重写 fputc 函数。还需要重写一些因为关闭半主机而被屏蔽的跟半主机密切相关的其他函数。代码如下：

代码 1 Keil 下使用标准 c 库来将 printf 重定向到串口

```
#pragma import(__use_no_semihosting) //确保没有从 C 库链接使用半主机的函数

//标准库需要的支持函数
//因为禁止了半主机模式，需要重写一个半主机模式下的接口，如下
int _ttywrch(int ch)
{
    ch=ch;
    return ch;
}
struct __FILE
{
    int handle;
};

FILE __stdout;

//定义_sys_exit()以避免使用半主机模式
void _sys_exit(int x)
{
      x = x;
}

int fputc(int ch, FILE *f)
{
    usart_data_send(USART1, (uint16_t) ch);  //根据自己使用的芯片平台和 SDK 替换 usart 驱动函数
    while(usart_flag_status_get(USART1, USART_FLAG_TXE) == RESET);
    return ch;
}
```

以上代码可以复制并移植到 Keil 项目工程中的任一 c 文件中。

当我们在 keil 中要使用 MicroLib 库的时候，则更为简单，只需要进行下面 2 步操作：

1. 在 Keil 中勾选 Use MicroLIB 选项，如下图所示：

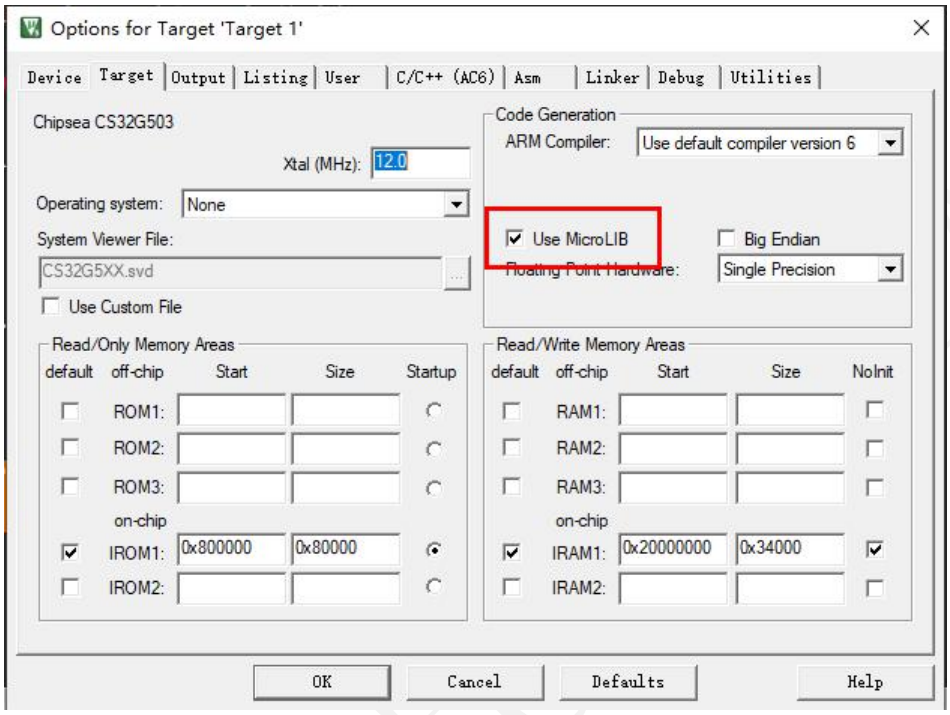


图 1 keil 下 MicroLib 配置

2. 实现 fputc()函数。fputc 函数与上面代码 1 Keil 下使用标准 c 库来将 printf 重定向到串口中 fputc 函数完全一致。

3. IAR8.x 版本下 printf 重定向到串口的方法

对于 IAR 8.x 版本，包括 IAR8.x 之前的版本，printf 重定向到串口与 Keil 下并无太大差异，可以完全复用 KEIL 下的重定向代码。仅需额外对 IAR IDE 作简单的配置即可。

如下图所示：

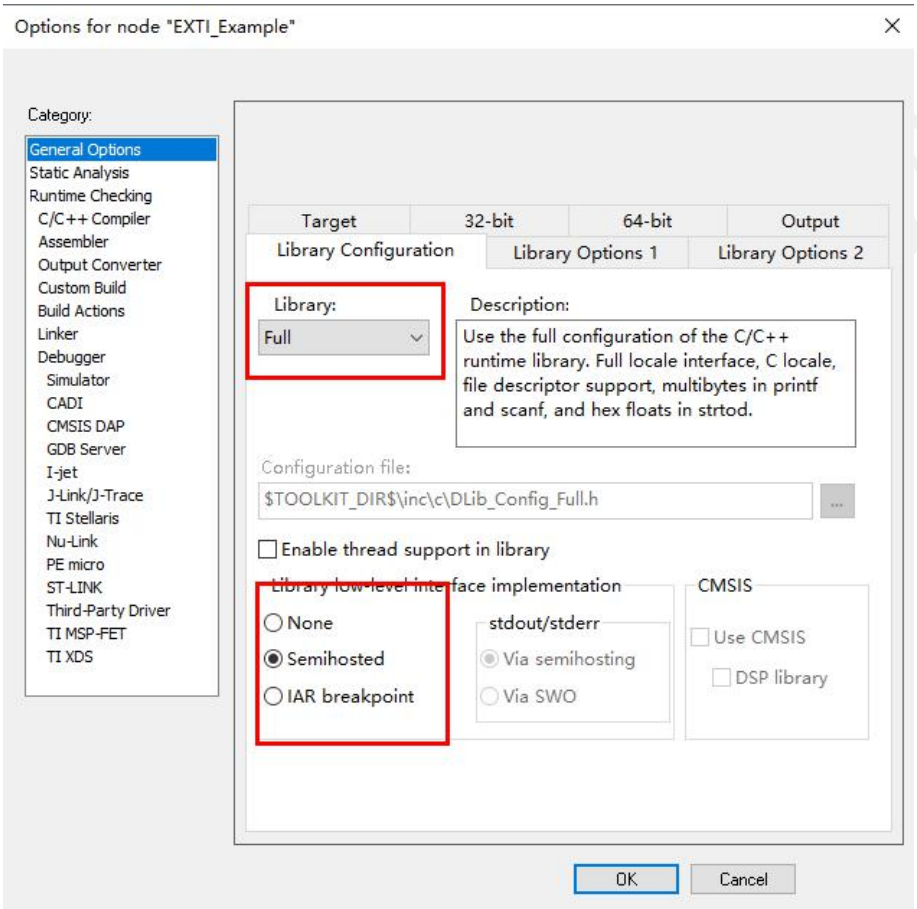


图 2 IAR8.x 及之前版本下 printf 重定向到串口配置

至此程序中便可以正常使用 printf 函数。

4. IAR9.x 版本下 printf 重定向到串口的方法

目前，IAR Systems 最新发布的集成开发环境 IAR Embedded Workbench for Arm 9.32 版本已全面支持芯海科技 32 位 MCU 芯片，其 IAR Systems 为芯海科技 CS32 系列 MCU 提供完整的工具链。

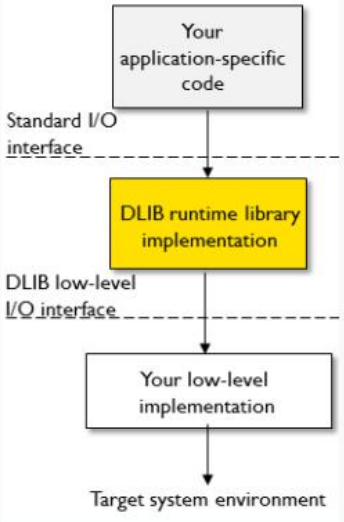
IAR9.x 版本下，对 printf 重定向到串口的要求发生了较大改变。在 IAR9.x 版本下，IAR 官方已经不再支持 fputc 接口的重定向，此时若移植原 Keil 下代码，则无法在 IAR 下编译通过。IAR 官方的说明如下：

BRIEFLY ABOUT RETARGETING

Retargeting is the process where you adapt the runtime environment so that your application can execute I/O operations on your target system.

The standard I/O interface is large and complex. To make retargeting easier, the DLIB runtime environment is designed so that it performs all I/O operations through a small set of simple functions, which is referred to as the *DLIB low-level I/O interface*. By default, the functions in the low-level interface lack usable implementations. Some are unimplemented, others have stub implementations that do not perform anything except returning error codes.

To retarget the standard I/O interface, all you have to do is to provide implementations for the functions in the DLIB low-level I/O interface.



For example, if your application calls the functions `printf` and `fputc` in the standard I/O interface, the implementations of those functions both call the low-level function `__write` to output individual characters. To make them work, you just need to provide an implementation of the `__write` function—either by implementing it yourself, or by using a third-party implementation.

图 3 IAR9.x 下 printf 重定向要求

根据官方文档，可以看出，在 IAR9.x 版本下，用户需要负责__write 函数的实现，以下为示例代码：

代码 2 IAR9.x 下 printf 重定向到串口

```
#include <LowLevelIOInterface.h>
#pragma module_name = "?__write"
uint8_t usart_transmit(usart_reg_t* usart, uint8_t *pdata, uint16_t size)
{
  uint8_t ch = 0;
  uint16_t i = 0;
  uint16_t timeout = 0x1000;
  for(i=0;i<size;i++)
  {
      ch = pdata[i];

      usart_data_send(usart,(uint8_t)ch);
      while ((usart_flag_status_get(usart, USART_FLAG_TXE) == RESET) && (timeout --
));
      if(timeout == 0)
      {
          return 0;
      }
      timeout = 0x1000;
  }
          return 1;
}

size_t __write(int handle, const unsigned char * buffer, size_t size)
{
     if (buffer == 0)
     {
         return 0;
     }
     if (handle != _LLIO_STDOUT && handle != _LLIO_STDERR)
     {
         return _LLIO_ERROR;
     }
     if(usart_transmit(USART1,(uint8_t *)buffer,size) == 1)
     {
         return size;
     }
     else
     {
         return _LLIO_ERROR;
     }
}
```

__write 函数为 IAR 要求我们进行重定向的函数，这里使用的发送串口是串口 1，若需要使用其他串口或者使用其他芯片 SDK 的 USART 外设驱动，可以相应修改或替换。

usart_transmit 函数是为了便捷使用发送功能进行定义的函数，其采用的是轮训发送的操作，有超时设置，返回 1 为发送成功，返回 0 为发送失败。

至此程序中便可以正常使用 printf 函数。

5. IAR8.x 与 IAR9.x 及 Keil 下的通用重定向代码

此节，根据本应用笔记之前对不同 IDE 及 IAR 不同版本下重定向区别的说明，为了方便用户代码在不同 IDE 和不同 IDE 版本下移植，避免耗费精力开发不同 IDE 下重定向代码。在此节，整理之前代码，并给出可以一键移植到 Keil、IAR8.x、IAR9.x 版本下的一套通用重定向代码。代码如下：

代码 3 Keil、IAR8.x、IAR9.x 版本下 printf 通用重定向代码

```
#if defined(__ICCARM__) && (__VER__ >= 9000000)
#include <LowLevelIOInterface.h>
#pragma module_name = "?__write"
#endif

#if defined (__GNUC__) && !defined (__ARMCC_VERSION)
/* With GCC, small printf (option LD Linker->Libraries->Small printf
   set to 'Yes') calls __io_putchar() */
#define PUTCHAR_PROTOTYPE int __io_putchar(int ch)
#else
#define PUTCHAR_PROTOTYPE int fputc(int ch, FILE *f)
#endif /* __GNUC__ */

PUTCHAR_PROTOTYPE
{
  /* Place your implementation of fputc here */
  /* e.g. write a character to the EVAL_COM1 and Loop until the end of transmission */
    (void)(f);
    usart_data_send(USART1, (uint16_t) ch);
    while(usart_flag_status_get(USART1, USART_FLAG_TXE) == RESET);
    return ch;
}

#if defined(__ICCARM__) && (__VER__ >= 9000000)
uint8_t usart_transmit(usart_reg_t* usart, uint8_t *pdata, uint16_t size)
{
    uint8_t ch = 0;
    uint16_t i = 0;
    uint16_t timeout = 0x1000;
    for(i=0;i<size;i++)
    {
        ch = pdata[i];
        usart_data_send(usart,(uint8_t)ch);
        while ((usart_flag_status_get(usart, USART_FLAG_TXE) == RESET) && (timeout -- ));
        if(timeout == 0)
        {
           return 0;
        }
        timeout = 0x1000;
    }
    return 1;
```

```
}

size_t __write(int handle, const unsigned char * buffer, size_t size)
{
  if (buffer == 0)
  {
      return 0;
  }
  if (handle != _LLIO_STDOUT && handle != _LLIO_STDERR)
  {
      return _LLIO_ERROR;
  }
  if(usart_transmit(USART1,(uint8_t *)buffer,size) == 1)
  {
      return size;
  }
  else
  {
      return _LLIO_ERROR;
  }
}
#endif
```

使用方法：用户只需将以上[代码 3 Keil、IAR8.x、IAR9.x 版本下 printf 通用重定向代码]，整体替换掉原来 Keil 项目中的 fputc 函数，即可无需作任何其他改动，就可实现 printf 函数在 Keil、IAR8.x、IAR9.x 下的编译与打印输出。

免责声明和版权公告

本文档中的信息，包括供参考的 URL 地址，如有变更，恕不另行通知。

本文档可能引用了第三方的信息，所有引用的信息均为“按现状”提供，芯海科技不对信息的准确性、真实性做任何保证。

芯海科技不对本文档的内容做任何保证，包括内容的适销性、是否适用于特定用途，也不提供任何其他芯海科技提案、规格书或样品在他处提到的任何保证。

芯海科技不对本文档是否侵犯第三方权利做任何保证，也不对使用本文档内信息导致的任何侵犯知识产权的行为负责。本文档在此未以禁止反言或其他方式授予任何知识产权许可，不管是明示许可还是暗示许可。

Wi-Fi 联盟成员标志归 Wi-Fi 联盟所有。蓝牙标志是 Bluetooth SIG 的注册商标。

文档中提到的所有商标名称、商标和注册商标均属其各自所有者的财产，特此声明。

芯海科技
CHIPSEA
股票代码:688595