ZT0005 专题教程

作者：Eric2013

工程调试利器 RTT，替代串口调试

销售QQ: 1295744630　　销售旺旺：armfly

销售电话：13638617262　　邮箱：armfly@qq.com

公司网址：www.armfly.com　　技术支持论坛：bbs.armfly.com

淘宝直销：armfly.taobao.com

微信公众号：安富莱电子

武汉安富莱电子有限公司　专业开发板、显示模块制造商　承接项目开发（提供生产供货服务）

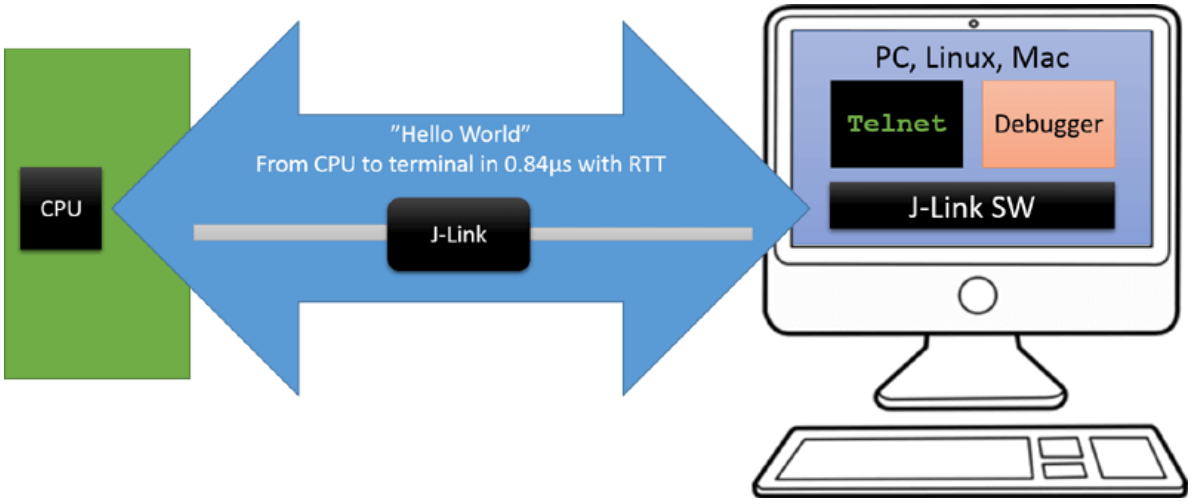
串口作为经典的调试方式已经存在好多年了，缺点是需要一个专门的硬件接口。现在有了 SEGGER 的 RTT（Real Time Transfer），无需占用系统额外的硬件资源，而且速度超快，是替代串口调试的绝佳方式。

1.1 重要提示（必读）

- RTT 的 API 可以在中断和多任务环境中正常使用。并且 JLINK 处于 MDK 或者 IAR 的调试状态，RTT 功能依然可以正常使用。最重要的是速度非常快，普通的 JLINK 也可以飙到几百 KB/S。
- RTT Viewer 小软件支持多个虚拟端口消息展示，比如用户可以一个用于标准输出，一个用于错误输出，另一个用于调试输出。根据需要还可以再增加输出窗口。
- RTT 组件和官方用户手册下载地址：
  http://forum.armfly.com/forum.php?mod=viewthread&tid=86014 。
- 这个软件不需要用到 SWO 引脚，使用标准的下载接口即可。以我们的开发板为例，用到 VCC，GND，SWDIO，SWCLK 和 NRST。大家使用三线 JLINK-OB 也是没问题的，仅需用到 GND，SWDIO 和 SWCLK。
- 大家买的 D 版 JLINK，基本都是来自 JLINK BASE，需要使用 V8，V9， V10 均可。
- 对于本专题配套的例子，使用 MDK4.7X 以及 MDK5 均可。使用 MDK5 的话，推荐使用当前最新的 MDK5.25。
  STM32-V4 和 STM32-V5 板子配套的例子使用 IAR6.3，其它版本未做测试。
  STM32-V6 板子使用 IAR7.5，其它版本未做测试。

1.2 RTT (Real Time Transfer) 简介

通过 RTT，可以输出来自目标芯片的信息，并以非常高的速度发送给应用程序，而不会影响目标的实时行为。当前所有的 JLINK 都支持 RTT 功能。



RTT 的多通道都支持双向通信（发送或者接收，跟串口一样）。默认情况下每个方向一个通道，用于终端的输入和输出。JLINK 的小软件 RTT Viewer 支持多个“虚拟”终端，允许用一个目标缓冲区打印到多个窗口，例如一个用于标准输出，一个用于错误输出，另一个用于调试输出。

这个软件不需要用到 SWO 引脚，使用标准的下载接口即可。以我们的开发板为例，用到 VCC，GND，SWDIO，SWCLK 和 NRST。大家使用三线 JLINK-OB 也是没问题的，仅需用到 GND，SWDIO 和 SWCLK。

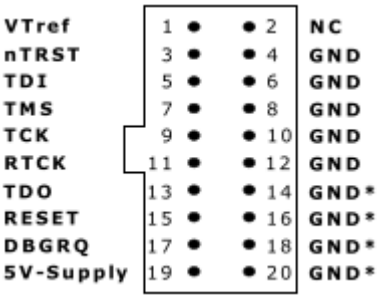
- MDK 逻辑分析仪

关于 MDK 逻辑分析仪的使用，可以看此贴：

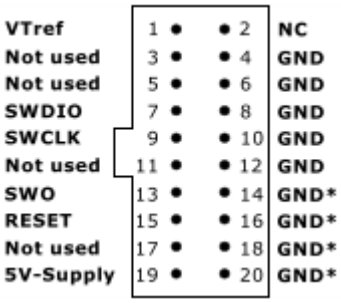
http://forum.armfly.com/forum.php?mod=viewthread&tid=18097 。

- JTAG 接口和 SWD 接口区别

下图分别是 20pin 的标准 JTAG 引脚和 SWD（Serial Wire Debug）引脚，一般 SWD 接口仅需要 Vref，SWDIO，SWCLK，RESET 和 GND 五个引脚即可，SWO（Serial Wire Output）引脚是可选的。有了 SWO 引脚才可以实现数据从芯片到电脑端的发送。



JTAG 接口



SWD 接口

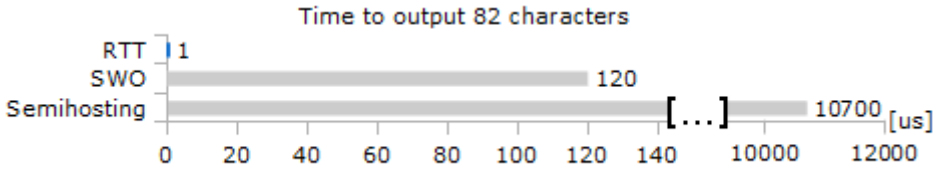
- 词条 SWV（Serial Wire Viewer）

SWV 是由仪器化跟踪宏单元 ITM(Instrumentation Trace Macrocell)和 SWO 构成的。SWV 实现了一种从 MCU 内部获取信息的低成本方案，SWO 接口支持输出两种格式的跟踪数据，但是任意时刻只能使用一种。两种格式的数据编码分别是 UART（串行）和 Manchester（曼彻斯特）。当前 JLINK 仅支持 UART 编码，SWO 引脚可以根据不同的信息发送不同的数据包。当前 M3/M4 可以通过 SWO 引脚输出以下三种信息：

1. ITM 支持 printf 函数的 debug 调用（工程需要做一下接口重定向即可）。ITM 有 32 个通道，如果使用 MDK 的话，通道 0 用于输出调试字符或者实现 printf 函数，通道 31 用于 Event Viewer，这就是为什么实现 Event Viewer 需要配置 SWV 的原因。
2. 数据观察点和跟踪 DWT(Data Watchpoint and Trace)可用于变量的实时监测和 PC 程序计数器采样。
3. ITM 还附带了一个时间戳的功能：当一个新的跟踪数据包进入了 ITM 的 FIFO 时，ITM 就会把一个差分的时间戳数据包插入到跟踪数据流中。跟踪捕获设备在得到了这些时间戳后，就可以找出各跟踪数据之间的时间相关信息。另外，在时间戳计数器溢出时也会发送时间戳数据包。

1.2.1 RTT 方式与 SWO 和半主模式的速度对比

SEGGER 做的速度对比如下：



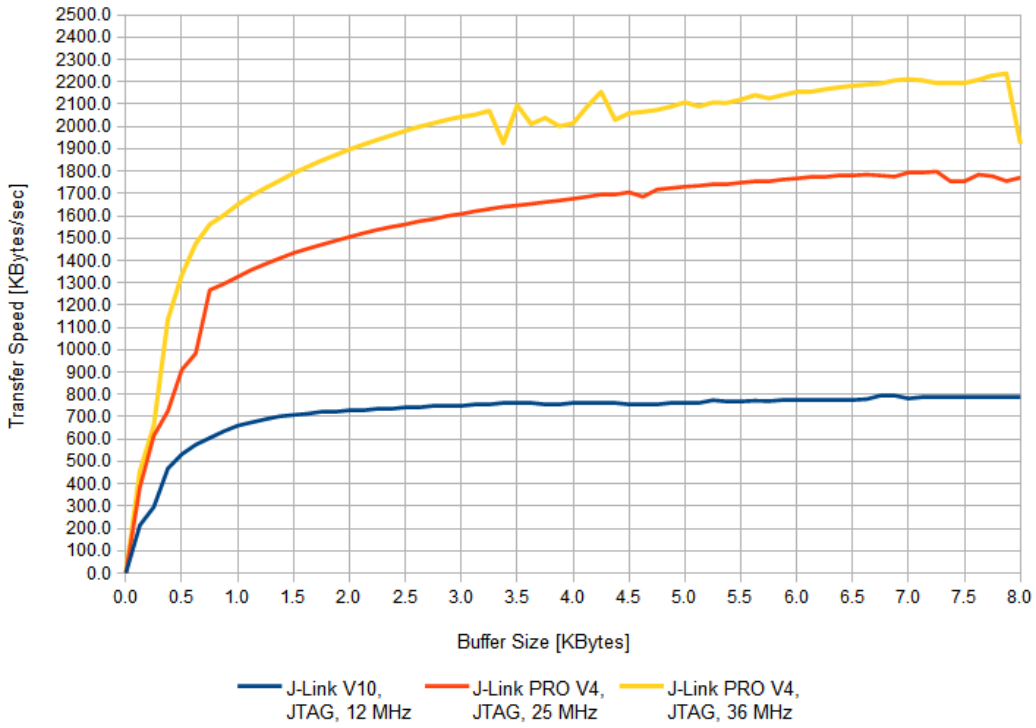
测试条件使用 STM32F407，主频 168MHz，通过重定向 printf 实现。由上面的测试数据，可以看到 RTT 输出 82 个字符需要 1us，SWO 模式需要 120us，而半主模式需要 10700us。

- SWO 模式和半主模式

  详情可以看此贴：http://forum.armfly.com/forum.php?mod=viewthread&tid=526 。

1.2.2 不同版本 JLINK 的速度对比

最大发送速度受配置的发送缓冲区大小和不同版本 JLINK 的接口速度限制。当配置发送缓冲区为 512 字节大小时，接口速度高的 JLINK 可以达到 1MB/S，普通接口速度的 JLINK 也可以达到 0.5MB/S：



横坐标是缓冲区配置大小，纵坐标是速度。

1.3 RTT 是如何工作的

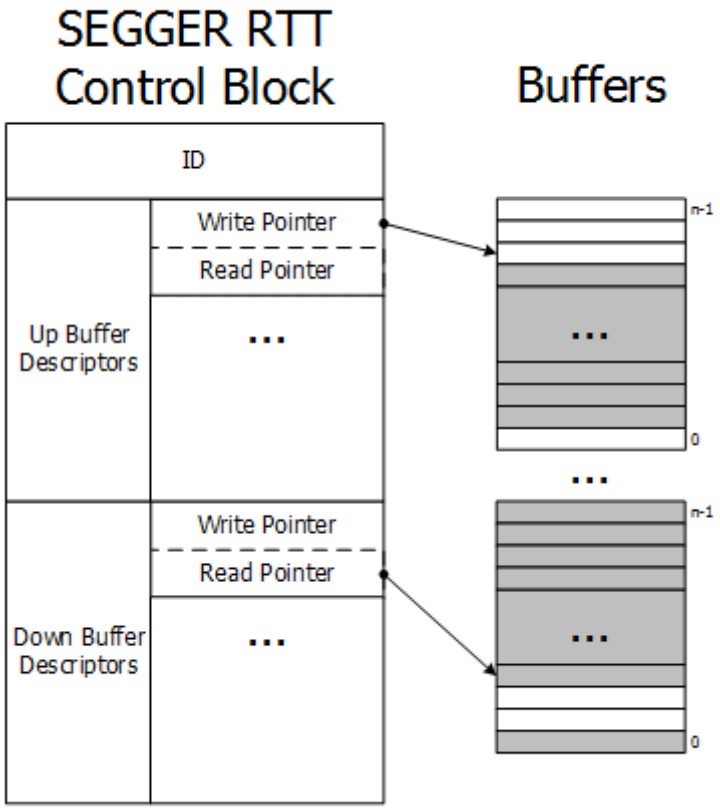
RTT 在芯片内存中使用控制块结构来管理数据的读取和写入。控制块包含一个 ID，以便通过连接的 JLINK 在内存中找到它，并为每个可用通道提供一个结构体，用于描述缓冲区及其状态。

可用通道的最大数量可以在编译时进行配置，并且每个缓冲区都可以在运行时由应用程序进行配置和添加。上行和下行缓冲区可以分开处理。

每个通道可以配置为阻塞或非阻塞。在阻塞模式下，当缓冲区已满时，应用程序将等待，直到有空间可以写入数据，虽然应用程序状态被阻止，但可以防止数据丢失。在非阻塞模式下，只有不超过缓冲区大小的数据被写入，其余的数据将被丢弃。即使没有连接调试器，也可以实时运行。而且开发人员不必创建特殊的调试版本，代码可以保留在以后发布的项目工程中。

==========

RTT 控制块的结构如下：



Up Buffer Descriptors 表示上行缓冲区描述符，即由芯片通过 JLINK 向电脑端上传缓冲区中的数据。Down Buffer Descriptors 表示下行缓冲区描述符，即由电脑端通过 JLINK 向芯片下行缓冲区发送数据。

每个缓冲区大小可以单独配置。缓冲区中的灰色区域是包含有效数据的区域。对于上行缓冲区，写入指针由芯片代码写入，读取指针由 JLINK 写入。当读取和写入指针指向相同的元素时，缓冲区为空。

1.4 RTT 的发送缓冲区配置多大合适

RTT 上行缓冲区可以相对较小。所需的最小缓冲区大小可以近似为一毫秒内写入的数据量或者一次写入操作中写入的最大值。 如果数据发送频率较低 ,那么缓冲区应该有足够的空间存储一次写入的数据。 如果频繁地发送数据，则缓冲区大小应满足在一毫秒内写入最大数据量。下图显示了发送不同数量的数据时所需的最小缓冲区大小，均匀分布，每隔 100 毫秒和每 1 毫秒。

每 100ms 发一次

Bytes per write	Bytes per ms	Buffer size
1	10	6
2	20	11
5	50	31
10	100	61
20	200	101
50	500	401

每秒发送一次

Bytes per write	Bytes per ms	Buffer size
10	10	11
20	20	21
50	50	51
100	100	101
200	200	201
500	500	501

曲线图：

注：采用 NXP K66 运行在 168MHz 测试，调试器是 JLINK Pro V4，JTAG 方式，速度 36MHz。

1.5 RTT 占用空间大小

RTT 组件对 ROM 和 RAM 空间的需求如下：

Memory	Usage
ROM Usage	~500 Bytes
RAM Usage	24 Bytes fixed + (24 + SizeofBuffer) Bytes / channel

1.6 RTT 移植

RTT 的移植比较简单，仅需要用户将所需文件添加到工程里面，并添加路径即可，我们这里分别以 MDK 和 IAR 为例进行说明。

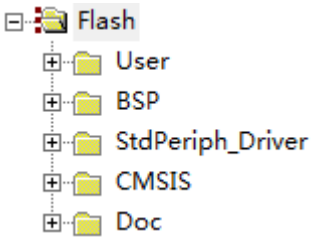
1.6.1 MDK 中移植 RTT 组件

- 第 1 步，下载 RTT 组件。

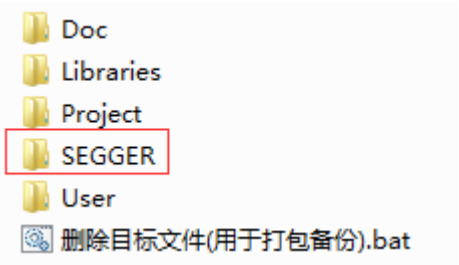
  下载地址：http://forum.armfly.com/forum.php?mod=viewthread&tid=86014 。

- 第 2 步，准备好一个工程模板。

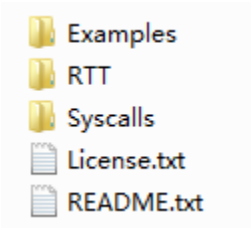
  这里以我们 V6 板子的：V6-101_按键检测和 LED 控制例程进行说明。工程分组如下：



- 第 3 步，在工程目录里面新建一个 SEEGER 文件夹，将 RTT 组件内容全都添加进去。

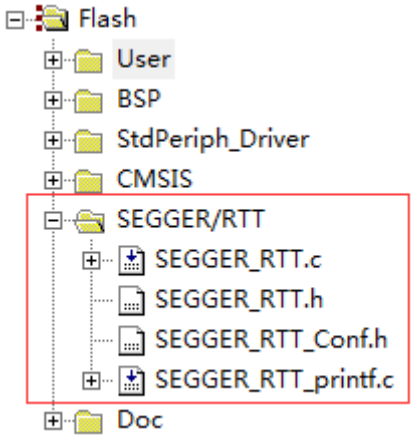


添加的内容如下：

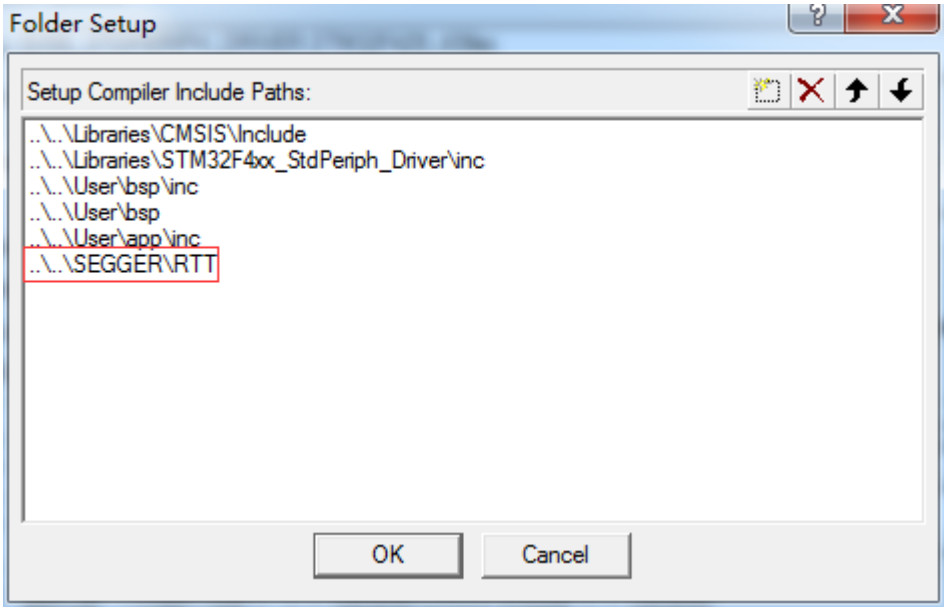


- 第 4 步，添加 RTT 组件到 MDK 工程里面。

将第 3 步中 RTT 文件中的四个文件全部添加到 MDK 工程里面，添加后的效果如下：



- 第 5 步，也是最后一步，添加路径。



至此，RTT 的组件就移植完毕。

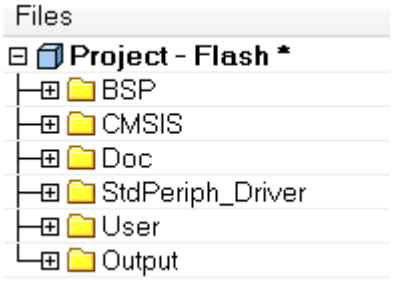
1.6.2 IAR 中移植 RTT 组件

- 第 1 步，下载 RTT 组件。

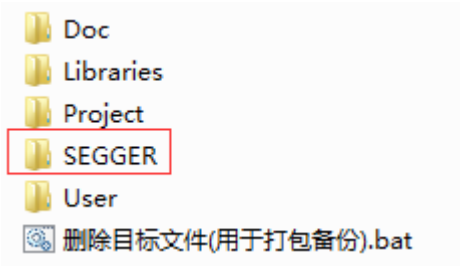
  下载地址：http://forum.armfly.com/forum.php?mod=viewthread&tid=86014 。

- 第 2 步，准备好一个工程模板。

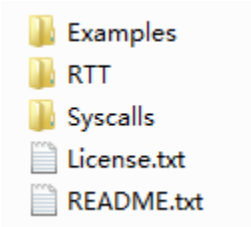
  这里以我们 V6 板子的：V6-101_按键检测和 LED 控制例程进行说明。工程分组如下：



◆ 第 3 步，在工程目录里面新建一个 SEEGER 文件夹，将 RTT 组件内容全都添加进去。

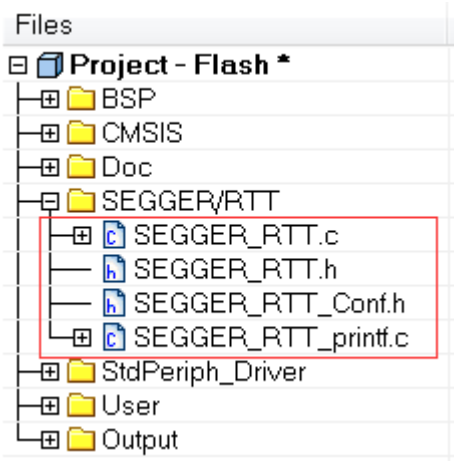


添加的内容如下：

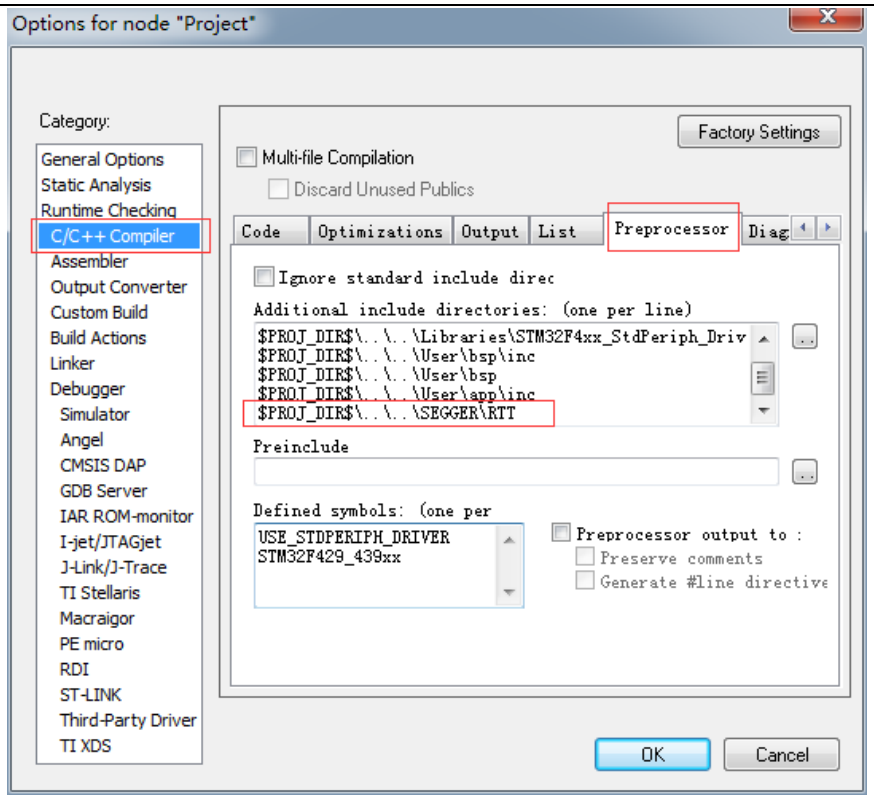


◆ 第 4 步，添加 RTT 组件到 IAR 工程里面。

将第 3 步中 RTT 文件中的四个文件全部添加到 IAR 工程里面，添加后的效果如下：



◆ 第 5 步，也是最后一步，添加路径。

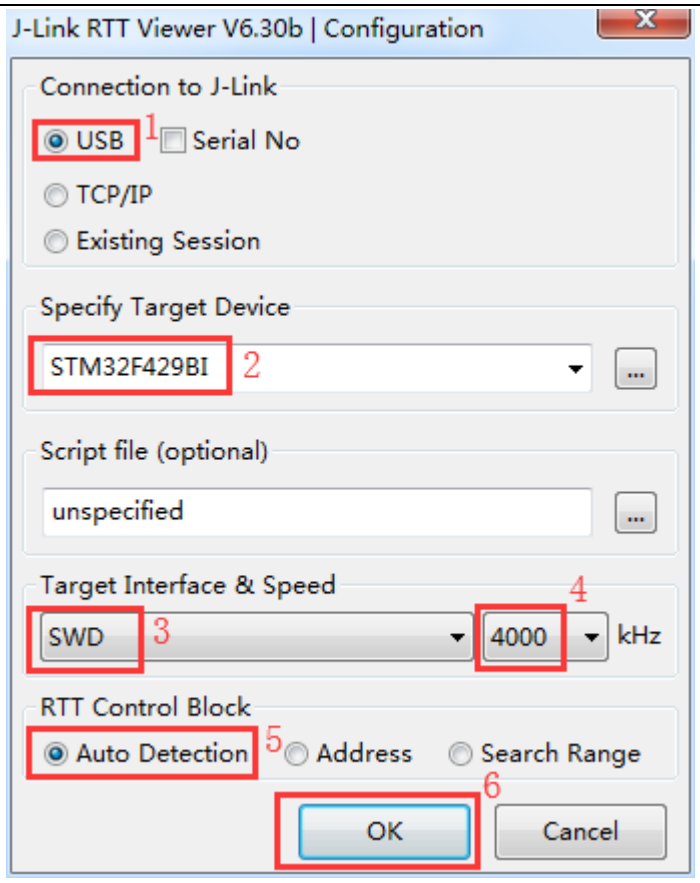


至此，RTT 的组件就移植完毕。

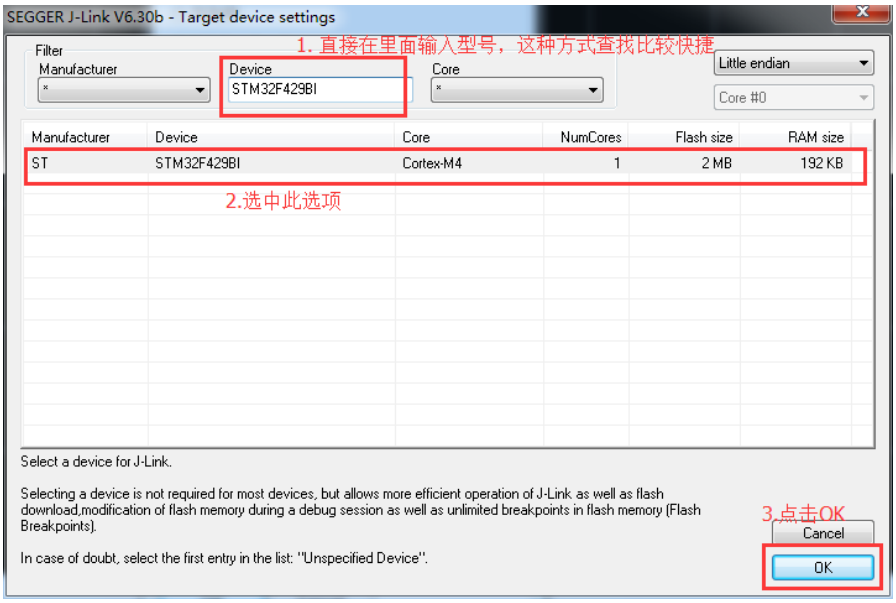
1.7 RTT 使用方法（RTT Viewer）

RTT 的电脑端软件 JLINK RTT Viewer 就在大家的 JLINK 电脑端驱动安装目录里面，下面把这个软件做个简单说明，此软件的使用也比较简单。

- 第 1 步：将板子下载程序后，重新上电。
- 第 2 步：打开软件 RTT Viewer，效果如下：

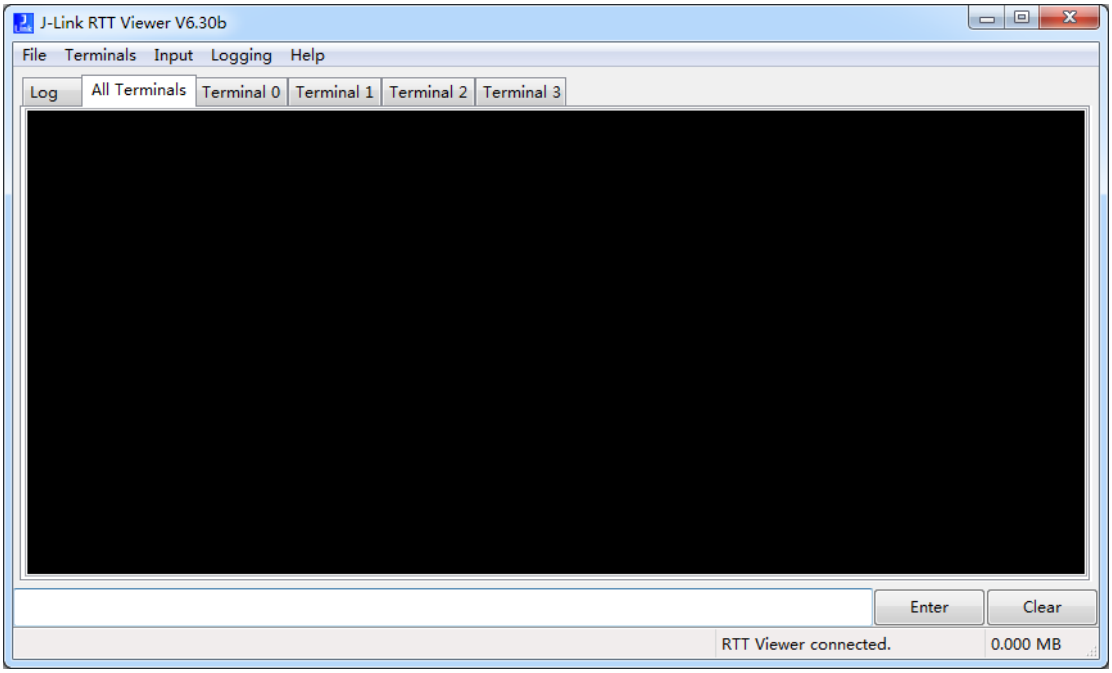


1. 一般大家用的 JLINK 都是 USB 接口，所以这里选择 USB。
2. 根据板子使用的芯片具体型号进行选择。比如我们要使用 V6 板子的 STM32F429BIT6：

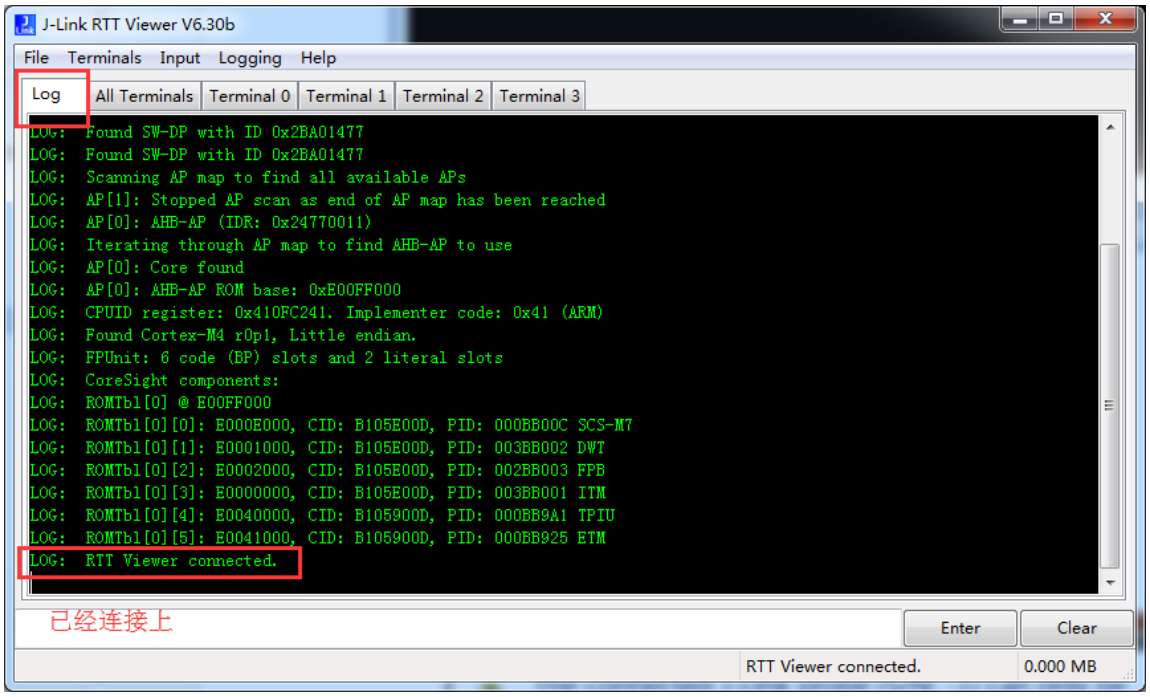


3. 根据使用的 JTAG 接口还是 SWD 接口进行选择。
4. 首次使用，JLINK 的速度不要太高，我这里设置的是 4MHz，防止不必要的麻烦。
5. 选择 Auto Detection。
6. 最后，点击 OK。

◆ 第 3 步，弹出的界面效果如下：

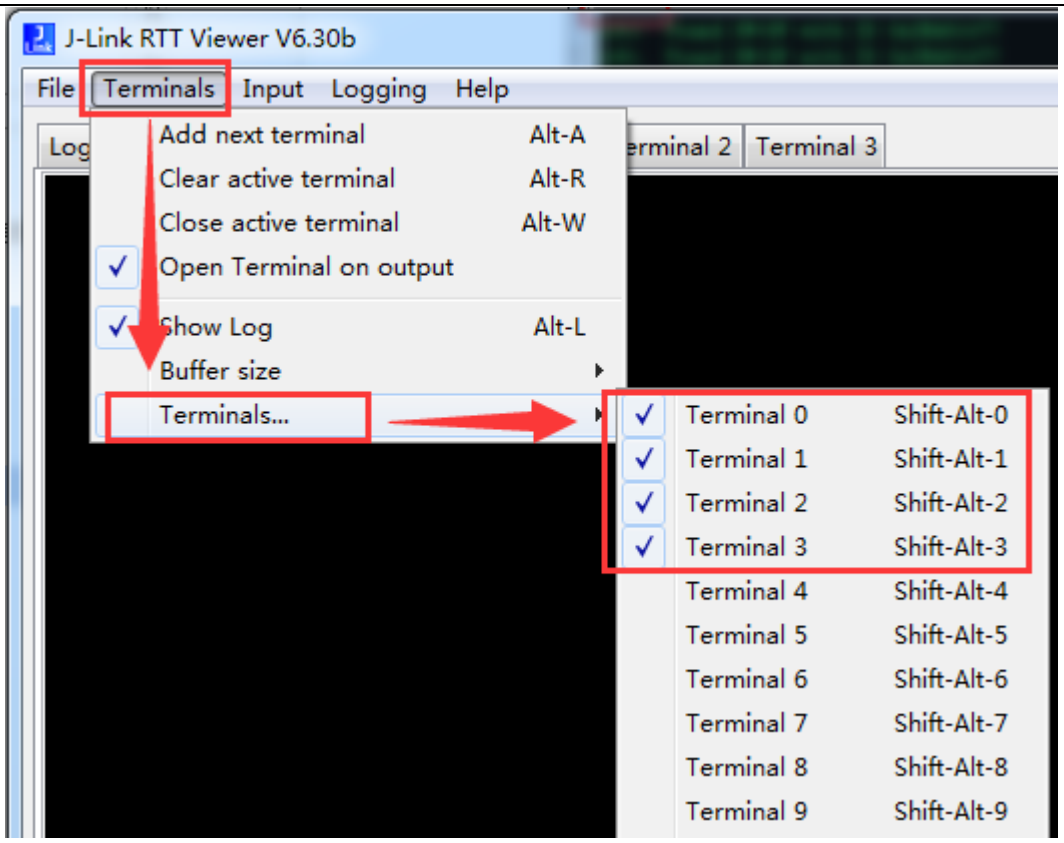


然后选择 Log 选项查看：



可以看到已经连接上，基本上大家打开这个软件就连接上了。如果没有连接上，点击菜单 File->Connect 即可。

也许有读者会问，为什么你这里有 Terminal0，1，2，3 四个窗口，这个是通过下面的选项进行选择的。



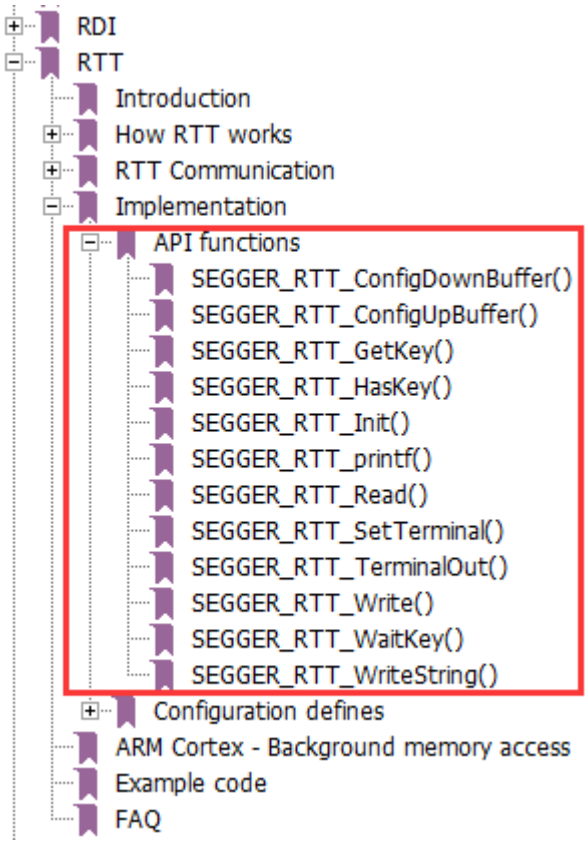
最多支持 15 个。这个功能非常有用，可以在 1.11 小节中看到效果。菜单里面其它选项也比较好掌握，操作几次就熟练了。

1.8 RTT 的 API 函数说明

RTT 支持如下 API：

API functions
SEGGER_RTT_ConfigDownBuffer()
SEGGER_RTT_ConfigUpBuffer()
SEGGER_RTT_GetKey()
SEGGER_RTT_HasKey()
SEGGER_RTT_Init()
SEGGER_RTT_printf()
SEGGER_RTT_Read()
SEGGER_RTT_SetTerminal()
SEGGER_RTT_TerminalOut()
SEGGER_RTT_WaitKey()
SEGGER_RTT_Write()
SEGGER_RTT_WriteString()

这几个 API 函数都比较容易掌握，在 JLINK 的用户文档里面有专门的讲解：



我们这里也对例程中用到的几个函数做个说明。

◆ 函数 int SEGGER_RTT_ConfigUpBuffer (unsigned BufferIndex, const char* sName, void* pBuffer, unsigned BufferSize, unsigned Flags)

对于第 1 个参数 BufferIndex = 0 的时候，RTT 组件已经为其配置了缓冲和默认的大小，其大小配置是在 SEGGER_RTT_Conf.h 文件里面通过如下宏定义进行：

```
#define BUFFER_SIZE_UP       (1024)
```

所以使用缓冲区 0 的时候，配置比较简单（第 3 个参数缓冲区地址设置为 NULL，第 4 个参数缓冲区大小设置为 0 即可）：

```
SEGGER_RTT_ConfigUpBuffer(0, "RTTUP", NULL, 0, SEGGER_RTT_MODE_NO_BLOCK_SKIP);
```

还有两个与此函数相关的宏定义：

- #define SEGGER_RTT_MAX_NUM_UP_BUFFERS (3)

  用于配置最大的上行缓冲数，也就是参数 BufferIndex 的范围。

- #define SEGGER_RTT_MODE_DEFAULT SEGGER_RTT_MODE_NO_BLOCK_SKIP

  用于配置 RTT 的默认工作模式，也就是此函数的最后一个参数，这里是表示如果上行缓冲区不够存储要发送的数据，将放弃写入缓冲区。除了这种配置还有如下两种：

- SEGGER_RTT_MODE_NO_BLOCK_TRIM

  表示如果上行缓冲区不够存储这些数据，能写多少写多少，无法写入的将丢弃。

- SEGGER_RTT_MODE_BLOCK_IF_FIFO_FULL

  表示如果上行缓冲区满，将被阻塞，等待有空间可用。

- int SEGGER_RTT_ConfigDownBuffer (unsigned BufferIndex, const char* sName, char* pBuffer, int BufferSize, int Flags);

  同上面的函数一样，仅仅是方向不同，这里是从电脑端向开发板发送数据。

- int SEGGER_RTT_HasKey (void)

  此函数用于判断接收缓冲区中是否有数据。返回 1 表示至少 1 个数据，返回 0 表示没有。

- int SEGGER_RTT_GetKey (void)

  从接收缓冲区 buffer 0 中接收一个字符。

- void SEGGER_RTT_SetTerminal(char TerminalId)

  用于设置在 RTT Viewer 的那个终端窗口显示。

- unsigned SEGGER_RTT_WriteSting (unsigned BufferIndex, const char* s)

  用于显示字符串，此函数比较简单。

- int SEGGER_RTT_printf (unsigned BufferIndex, const char * sFormat, ...)

  这个函数跟 C 库中 printf 一样，区别是不支持浮点数。

1.9 RTT 的 API 函数多任务调用

RTT 的 API 函数多任务调用是通过类似 FreeRTOS 的开关中断方式做的保护，也是基于 CM 内核的 BASEPRI 寄存器进行设置。关于这方面的知识可以看我们 FreeRTOS 教程的第 12 章，有详细说明。http://forum.armfly.com/forum.php?mod=viewthread&tid=17658。

而 RTT 的中断设置是在 SEGGER_RTT_Conf.h 文件里面进行。

```
//
// Target is not allowed to perform other RTT operations while string still has not been stored completely.
// Otherwise we would probably end up with a mixed string in the buffer.
// If using  RTT from within interrupts, multiple tasks or multi processors, define the SEGGER_RTT_LOCK() and
//SEGGER_RTT_UNLOCK() function here.

//
// SEGGER_RTT_MAX_INTERRUPT_PRIORITY can be used in the sample lock routines on Cortex-M3/4.
// Make sure to mask all interrupts which can send RTT data, i.e. generate SystemView events, or cause task
//switches.

// When high-priority interrupts must not be masked while sending RTT data, SEGGER_RTT_MAX_INTERRUPT_PRIORITY
//needs to be adjusted accordingly.
// (Higher priority = lower priority number)
// Default value for embOS: 128u
// Default configuration in FreeRTOS: configMAX_SYSCALL_INTERRUPT_PRIORITY:
//( configLIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY << (8 - configPRIO_BITS) )
// In case of doubt mask all interrupts: 1 << (8 - BASEPRI_PRIO_BITS) i.e. 1 << 5 when 3 bits are implemented
```

```
//in NVIC
// or define SEGGER_RTT_LOCK() to completely disable interrupts.
//

// Interrupt priority to lock on SEGGER_RTT_LOCK on Cortex-M3/4 (Default: 0x20)
#define SEGGER_RTT_MAX_INTERRUPT_PRIORITY         (0x20)
```

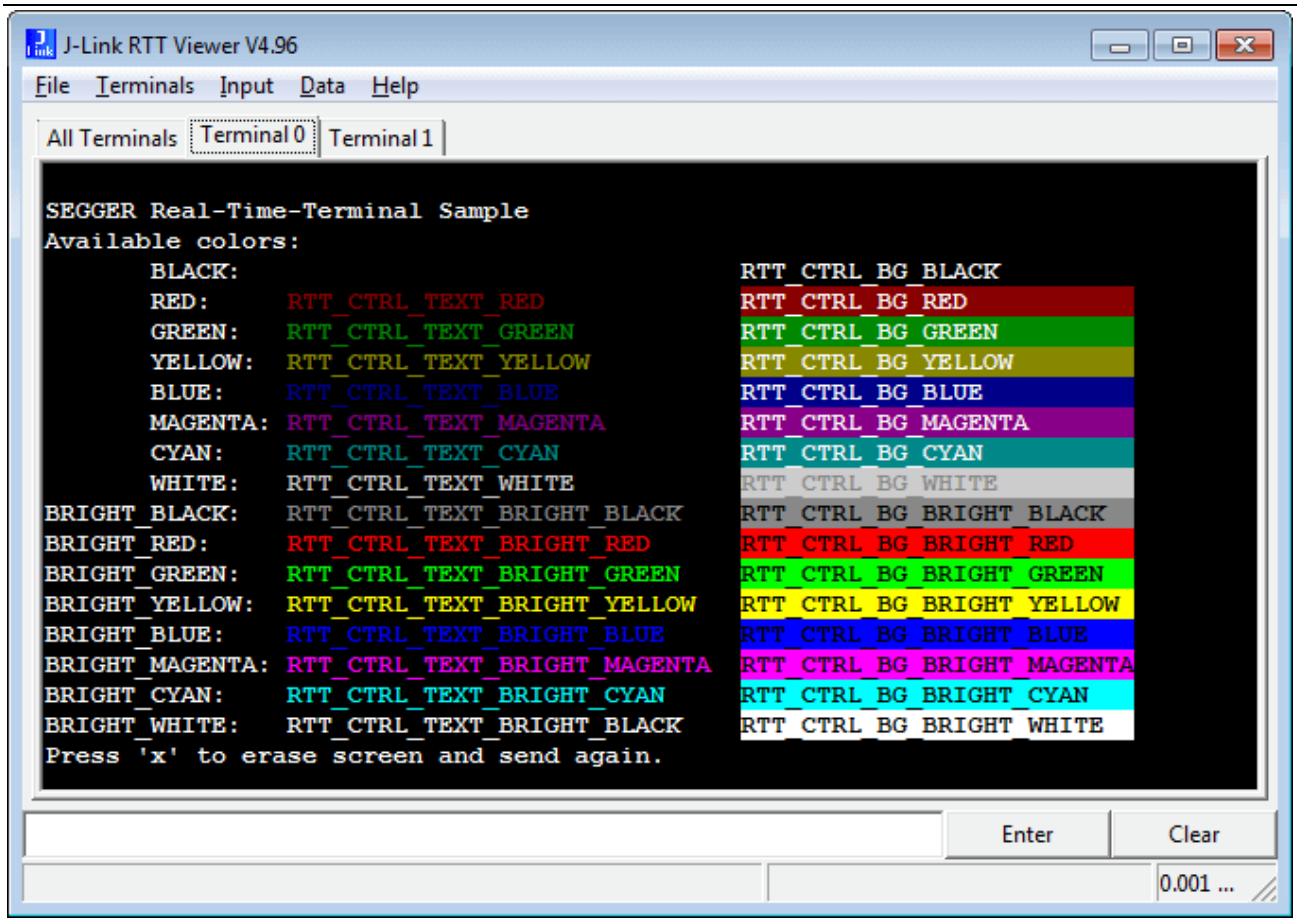
所以在多任务或者中断里面，大家可以放心使用。

1.10 RTT 输出颜色设置

常用的颜色设置标志如下所示（注意，非乱码）：

```
#define RTT_CTRL_RESET                "[0m"         // Reset to default colors
#define RTT_CTRL_CLEAR                "[2J"         // Clear screen, reposition cursor to top left

#define RTT_CTRL_TEXT_BLACK           "[2;30m"
#define RTT_CTRL_TEXT_RED             "[2;31m"
#define RTT_CTRL_TEXT_GREEN           "[2;32m"
#define RTT_CTRL_TEXT_YELLOW          "[2;33m"
#define RTT_CTRL_TEXT_BLUE            "[2;34m"
#define RTT_CTRL_TEXT_MAGENTA         "[2;35m"
#define RTT_CTRL_TEXT_CYAN            "[2;36m"
#define RTT_CTRL_TEXT_WHITE           "[2;37m"

#define RTT_CTRL_TEXT_BRIGHT_BLACK    "[1;30m"
#define RTT_CTRL_TEXT_BRIGHT_RED      "[1;31m"
#define RTT_CTRL_TEXT_BRIGHT_GREEN    "[1;32m"
#define RTT_CTRL_TEXT_BRIGHT_YELLOW   "[1;33m"
#define RTT_CTRL_TEXT_BRIGHT_BLUE     "[1;34m"
#define RTT_CTRL_TEXT_BRIGHT_MAGENTA  "[1;35m"
#define RTT_CTRL_TEXT_BRIGHT_CYAN     "[1;36m"
#define RTT_CTRL_TEXT_BRIGHT_WHITE    "[1;37m"

#define RTT_CTRL_BG_BLACK             "[24;40m"
#define RTT_CTRL_BG_RED               "[24;41m"
#define RTT_CTRL_BG_GREEN             "[24;42m"
#define RTT_CTRL_BG_YELLOW            "[24;43m"
#define RTT_CTRL_BG_BLUE              "[24;44m"
#define RTT_CTRL_BG_MAGENTA           "[24;45m"
#define RTT_CTRL_BG_CYAN              "[24;46m"
#define RTT_CTRL_BG_WHITE             "[24;47m"

#define RTT_CTRL_BG_BRIGHT_BLACK      "[4;40m"
#define RTT_CTRL_BG_BRIGHT_RED        "[4;41m"
#define RTT_CTRL_BG_BRIGHT_GREEN      "[4;42m"
#define RTT_CTRL_BG_BRIGHT_YELLOW     "[4;43m"
#define RTT_CTRL_BG_BRIGHT_BLUE       "[4;44m"
#define RTT_CTRL_BG_BRIGHT_MAGENTA    "[4;45m"
#define RTT_CTRL_BG_BRIGHT_CYAN       "[4;46m"
#define RTT_CTRL_BG_BRIGHT_WHITE      "[4;47m"
```

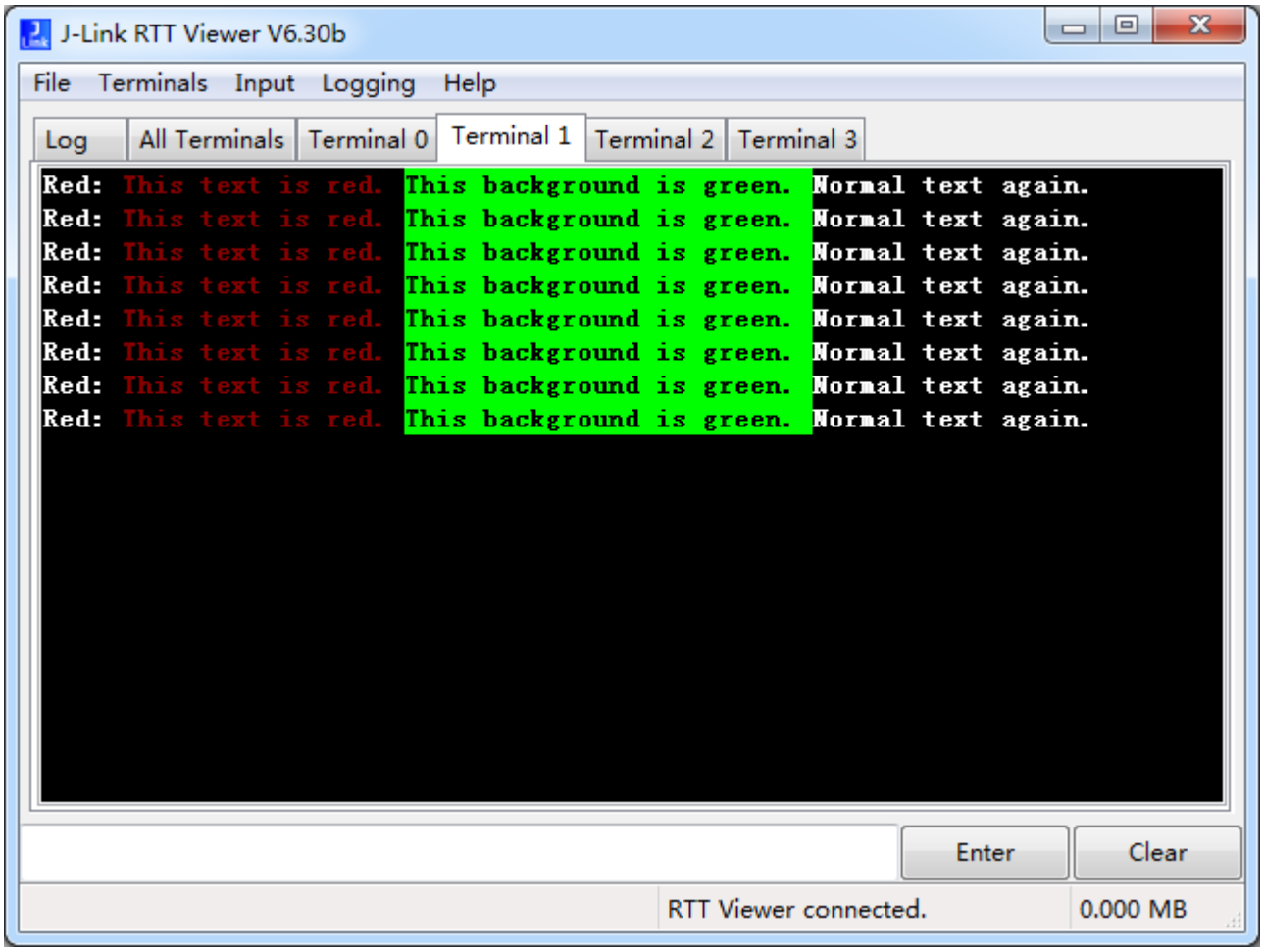
颜色效果如下：



比如使用函数 SEGGER_RTT_WriteString：

```
SEGGER_RTT_WriteString(0,
                       RTT_CTRL_RESET"Red: " \
                       RTT_CTRL_TEXT_RED"This text is red. " \
                       RTT_CTRL_TEXT_BLACK"" \
                       RTT_CTRL_BG_BRIGHT_GREEN"This background is green. " \
                       RTT_CTRL_RESET"Normal text again.\r\n");
```

显示效果如下：



1.11 配套例子

专题教程配套了三个例子，每个例子里面都是 IAR 和 MDK 两个版本。

- STM32-V4 开发板配套的例子是：V4-工程调试利器 SEGGER 的 RTT 组件，替代串口调试。
- STM32-V5 开发板配套的例子是：V5-工程调试利器 SEGGER 的 RTT 组件，替代串口调试。
- STM32-V6 开发板配套的例子是：V6-工程调试利器 SEGGER 的 RTT 组件，替代串口调试。

针对 STM32H743 也做了支持和移植，详情看帖子：

http://forum.armfly.com/forum.php?mod=viewthread&tid=86071 。

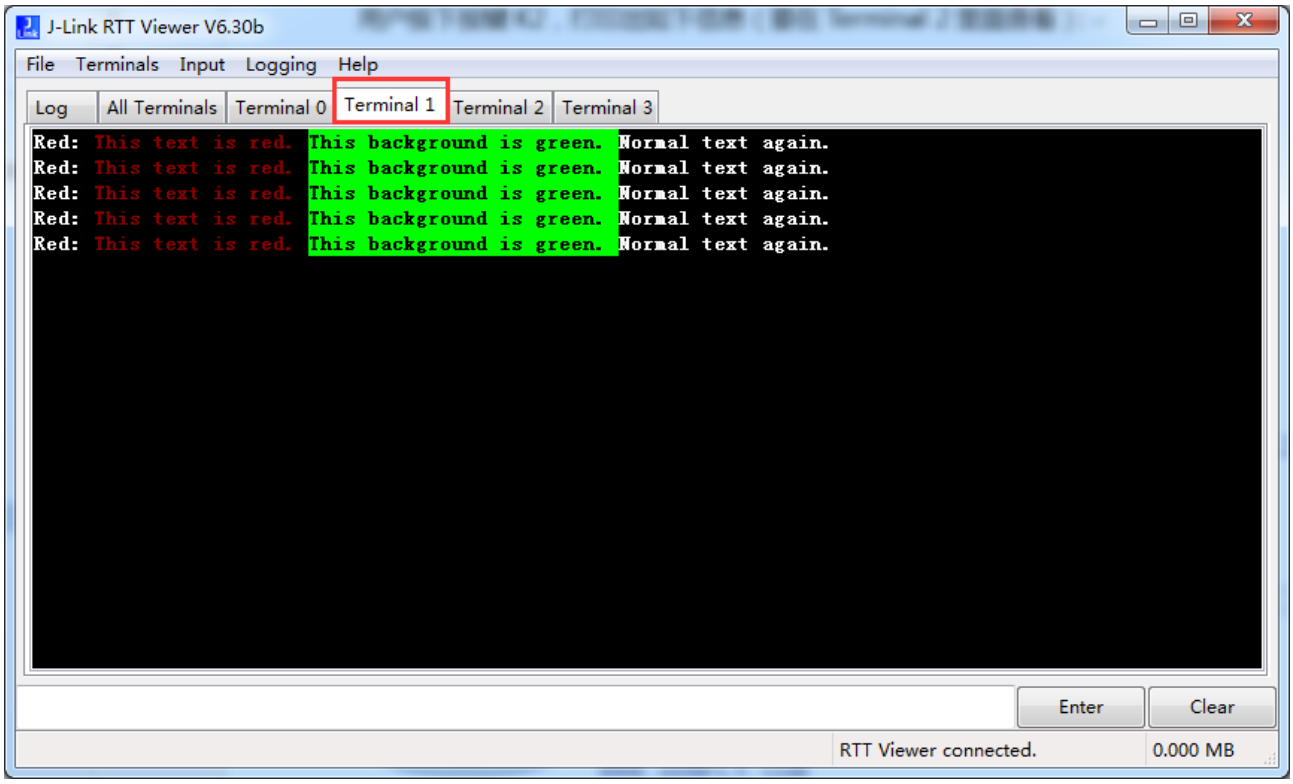
具体代码实现也比较简单，配置完毕上行和下行缓冲区就可以使用了。代码如下：

```
#include "bsp.h"            /* 底层硬件驱动 */
#include "SEGGER_RTT.h"



/*
*********************************************************************************************************
```

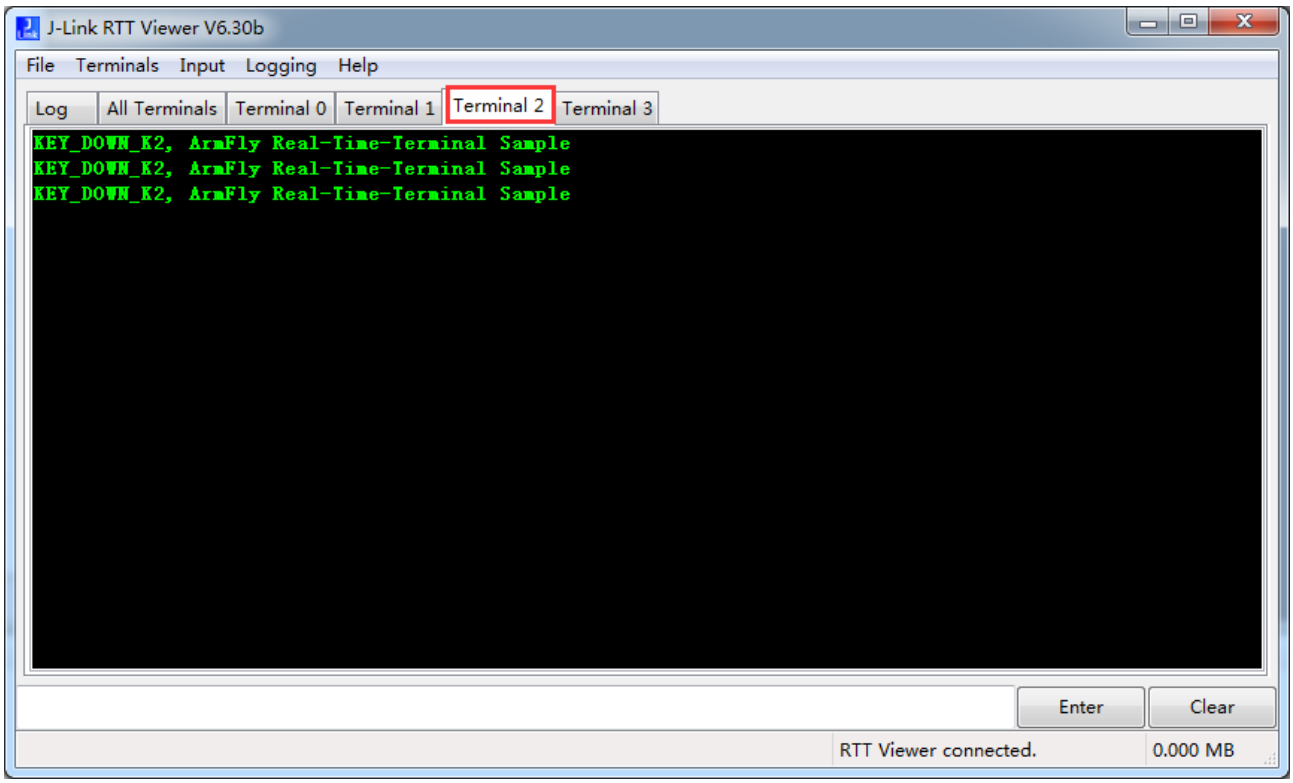
```
*    函 数 名: main
*    功能说明: c 程序入口
*    形    参: 无
*    返 回 值: 错误代码(无需处理)
*********************************************************************************************************
*/
int main(void)
{
    uint8_t ucKeyCode;      /* 按键代码 */
    uint32_t i = 0;
    int GetKey;

    /*
        由于 ST 固件库的启动文件已经执行了 CPU 系统时钟的初始化，所以不必再次重复配置系统时钟。
        启动文件 startup_stm32f4xx.s 会调用 system_stm32f4xx.c 中的 void SystemInit(void)。
        SystemInit()函数配置了 CPU 主时钟频率、内部 Flash 访问速度和可选的外部 SRAM FSMC 初始化。

        安富莱 STM32-V5 开发板主晶振是 25MHz，内部 PLL 倍频到 168MHz。如果需要更改主频，可以修改下面的文件：
        \User\bsp_stm32f4xx\system_stm32f4xx.c
        文件开头的几个宏是 PLL 倍频参数，修改这些宏就可以修改主频，无需更改硬件。
    */

    bsp_Init();        /* 硬件初始化 */

    bsp_StartAutoTimer(0, 100); /* 启动 1 个 100ms 的自动重装的定时器 */

    /* 配置通道 0，上行配置*/
    SEGGER_RTT_ConfigUpBuffer(0, "RTTUP", NULL, 0, SEGGER_RTT_MODE_NO_BLOCK_SKIP);

    /* 配置通道 0，下行配置*/
    SEGGER_RTT_ConfigDownBuffer(0, "RTTDOWN", NULL, 0, SEGGER_RTT_MODE_NO_BLOCK_SKIP);

    /* 进入主程序循环体 */
    while (1)
    {
        bsp_Idle();        /* 这个函数在 bsp.c 文件。用户可以修改这个函数实现 CPU 休眠和喂狗 */

        /* 判断定时器超时时间 */
        if (bsp_CheckTimer(0))
        {
            bsp_LedToggle(4);
        }

        /* 做一个简单的回环功能 */
        if (SEGGER_RTT_HasKey())
        {
            GetKey = SEGGER_RTT_GetKey();
            SEGGER_RTT_SetTerminal(0);
            SEGGER_RTT_printf(0, "SEGGER_RTT_GetKey = %c\r\n", GetKey);
        }

        /* 按键滤波和检测由后台 systick 中断服务程序实现，我们只需要调用 bsp_GetKey 读取键值即可。 */
        ucKeyCode = bsp_GetKey();   /* 读取键值，无键按下时返回 KEY_NONE = 0 */
        if (ucKeyCode != KEY_NONE)
        {
            switch (ucKeyCode)
            {
                case KEY_DOWN_K1:           /* K1 键按下 */
```

```
                    SEGGER_RTT_SetTerminal(1);
                    SEGGER_RTT_WriteString(0,
                                           RTT_CTRL_RESET"Red: " \
                                           RTT_CTRL_TEXT_RED"This text is red. " \
                                           RTT_CTRL_TEXT_BLACK"" \
                                           RTT_CTRL_BG_BRIGHT_GREEN"This background is green. " \
                                           RTT_CTRL_RESET"Normal text again.\r\n");

                    break;

                case KEY_DOWN_K2:           /* K2 键按下 */
                    SEGGER_RTT_SetTerminal(2);
                    SEGGER_RTT_WriteString(0, RTT_CTRL_TEXT_BRIGHT_GREEN"KEY_DOWN_K2, ArmFly
                    Real-Time-Terminal Sample\r\n");
                    break;

                case KEY_DOWN_K3:           /* K3 键按下 */
                    SEGGER_RTT_SetTerminal(3);
                    SEGGER_RTT_printf(0, "KEY_DOWN_K3, i = %d\r\n", i++);
                    break;

                default:
                    /* 其它的键值不处理 */
                    break;
            }
        }
    }
}
```

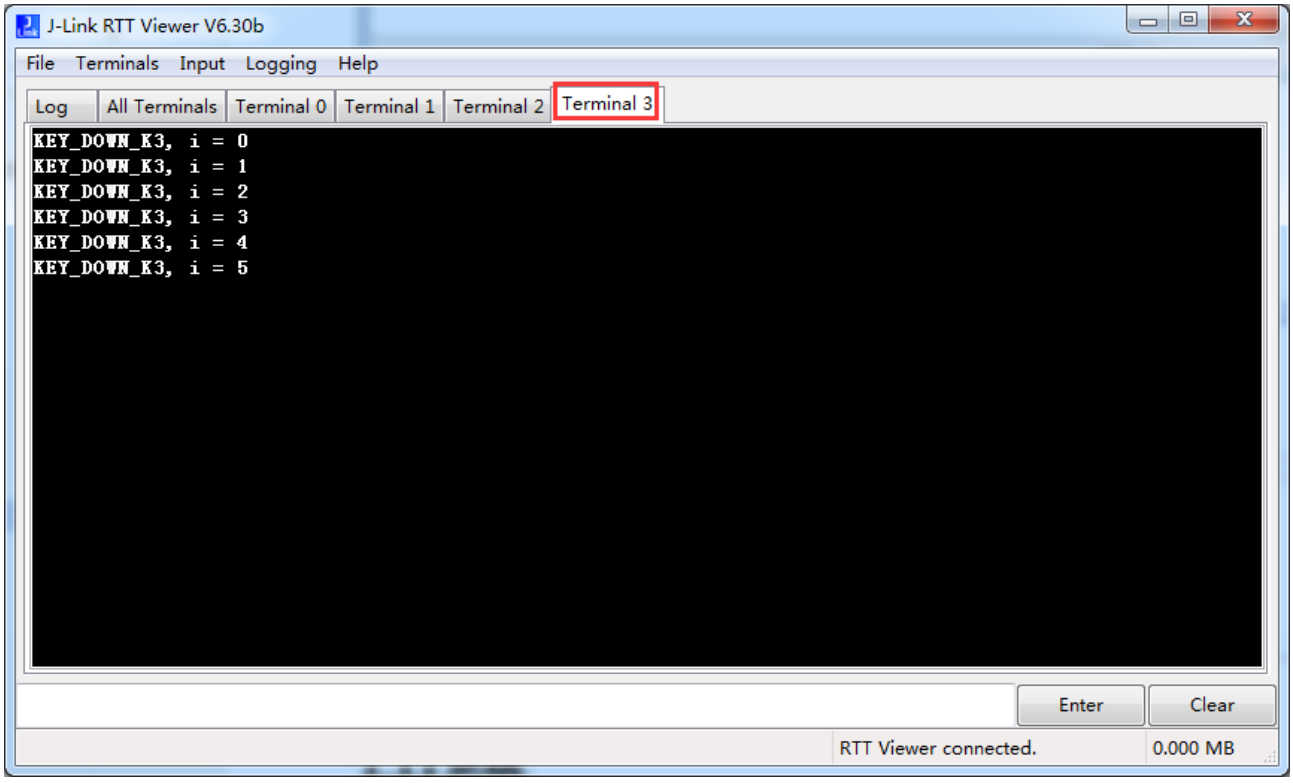
用户按下按键 K1，打印出如下信息（要在 Terminal 1 里面查看）：



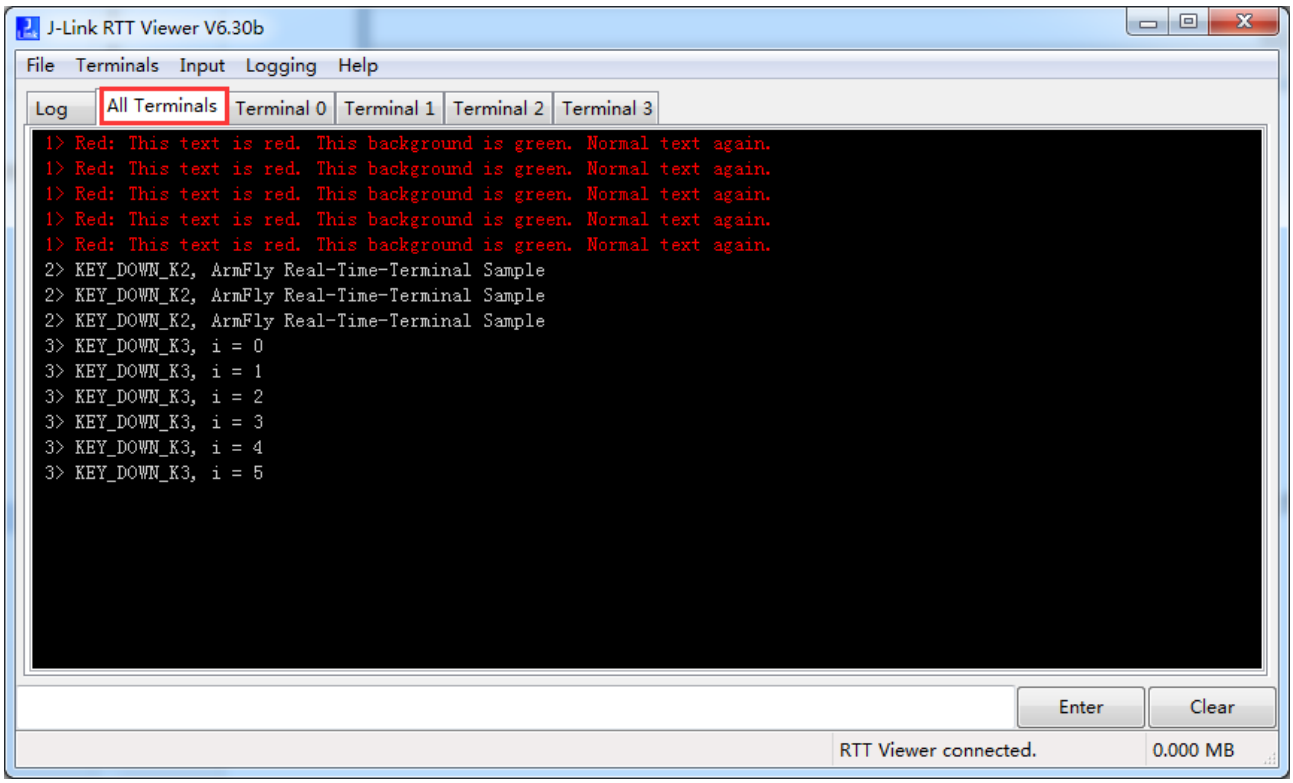
用户按下按键 K2，打印出如下信息（要在 Terminal 2 里面查看）：



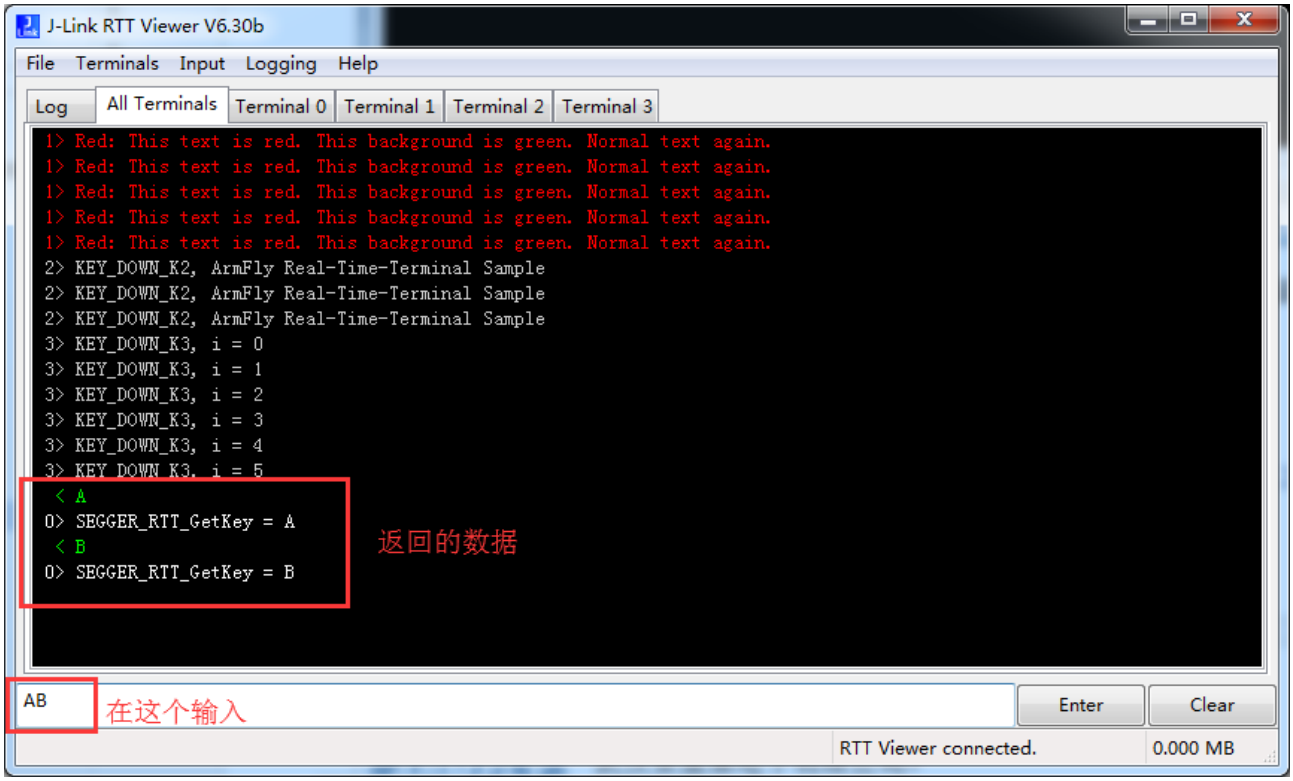
用户按下按键 K3，打印出如下信息（要在 Terminal 3 里面查看）：



同时查看所有打印（在 ALL Terminals 里面查看）：



程序里面还做了一个简单的回环功能，使用 RTT Viewer 的输入功能，输入什么，返回什么。



1.12 总结

RTT 在快速数据上传、多任务分析、中断里数据发送都是非常有优势的，建议大家掌握其使用方法。

1.13 文档更新记录

版本	更改说明	作者	发布日期
V1.0	初版首发，制作于2018-04-01	白永斌 （Eric2013）	2018-04-12 （25页）