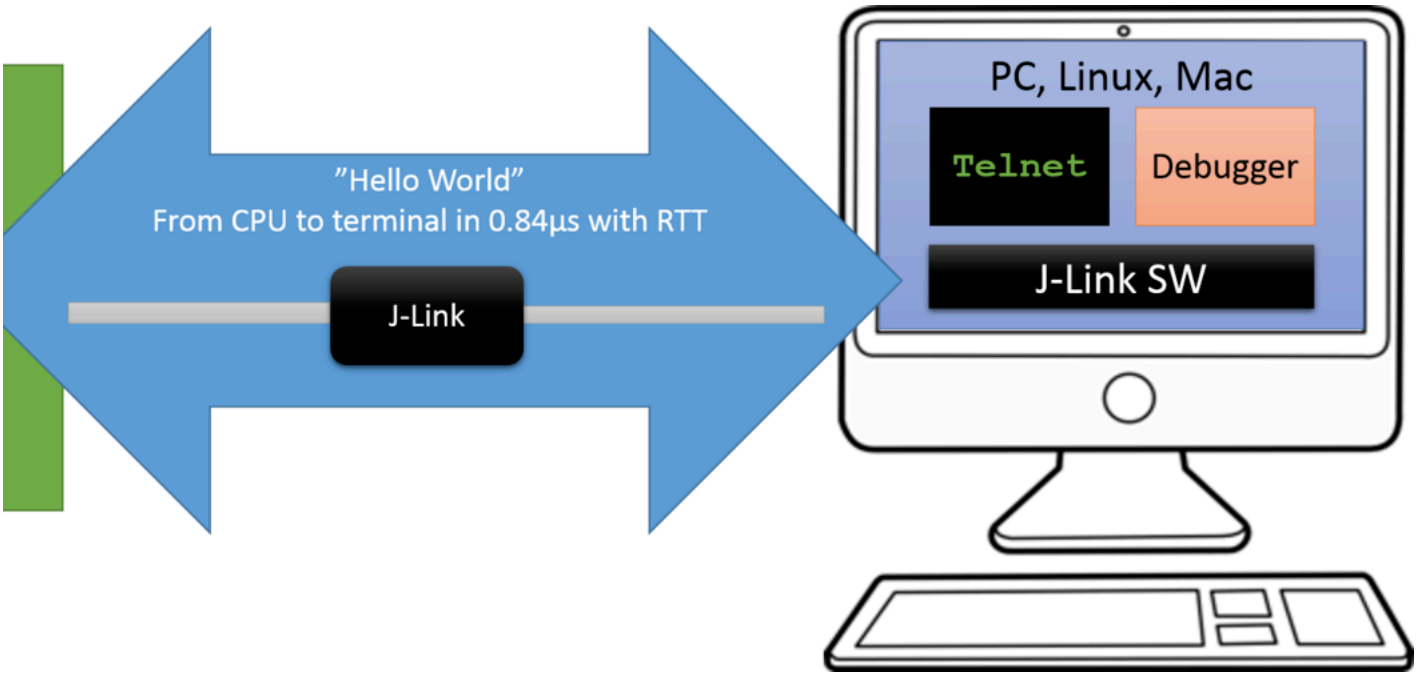
RTT的



SEGGER 的实时传输 （RTT） 是一种用于嵌入式应用中交互式用户 I/O 的技术。 它以非常高的性能结合了 SWO 和半托管的优势。
使用 RTT，可以从目标微控制器输出信息，并以非常高的速度将输入发送到应用 而不影响目标的实时行为。
SEGGER RTT 可与任何 J-Link 型号 (https://www.segger.com/products/debug-probes/j-link/?mtm_campaign=kb&mtm_kwd=RTT)和任何允许后台内存访问的受支持目标处理器一起使用， 即 Cortex-M 和 RX 目标。
RTT 支持双向多个通道，向上到主机，向下到目标， 可用于不同的目的，并为用户提供最大的自由。默认
实现每个方向使用一个通道，用于可打印的终端输入和输出。 使用 J-Link RTT Viewer，该频道可用于多个“虚拟”终端， 允许打印到多个窗口（例如，一个用于标准输出，一个用于错误输出，一个用于调试输出） 只有一个目标缓冲区。 例如，可以使用额外的向上（到主机）通道来发送分析或事件跟踪数据 （例如，对于 SEGGER SystemView (https://www.segger.com/products/development-tools/systemview/?mtm_campaign=kb&mtm_kwd=RTT)）。

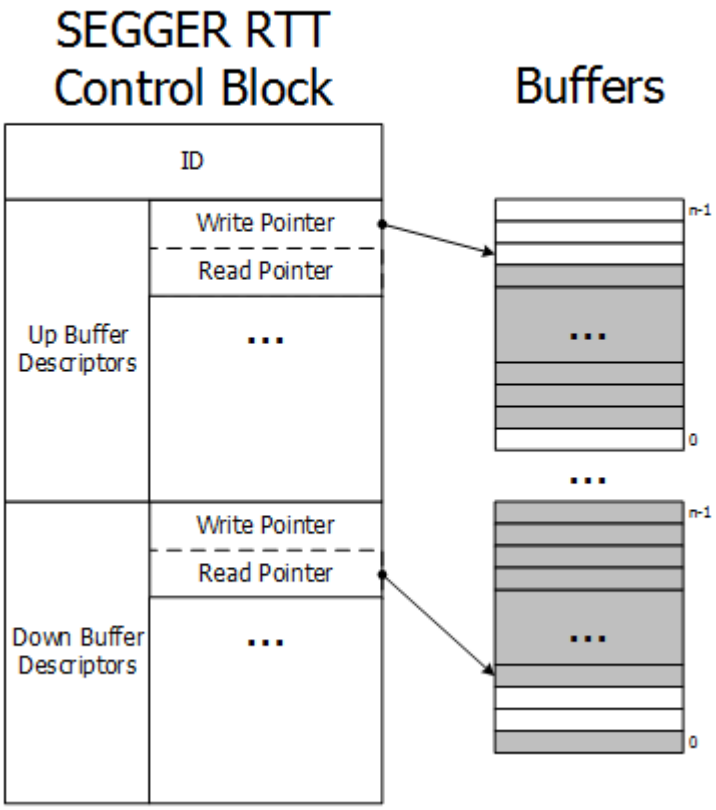
内容

RTT 的工作原理

RTT 使用 SEGGER RTT 控制块结构和环形缓冲区，用于位于 RAM 中的每个通道的每个配置方向。 可以在编译时配置最大可用通道数，并且每个缓冲区都可以由应用程序在运行时配置和添加。

- 可以单独处理上行和下行缓冲区（请参阅 RTT 通道）。
- 每个通道都可以配置为阻塞或非阻塞（请参阅缓冲区配置）：
  - 阻止：防止数据丢失，但可能会暂停应用程序。
  - 非阻塞：多余的信息将被丢弃，允许应用程序实时运行，即使没有连接调试器也是如此。

右图显示了目标中 RTT 的简化结构。每个要素将在下面进行解释。



RTT 控制块

RTT 控制块 （CB） 包含多个元素，以允许 RTT 工作。它位于 RAM 中。 它总是以一个 ID 开头，该 ID 用于

- 使 CB（自动）可通过连接的 J-Link 在内存中检测到，并且
- 用于 CB 有效性检查。

它后面是缓冲区描述符，其中包含所有必需的 RTT 通道信息。

缓冲区描述符

缓冲区描述符提供有关每个通道的环形缓冲区的信息，J-Link 使用这些缓冲器从目标读取信息并向目标写入信息。 可以有任意数量的 *Up*（目标 -> 主机）/*Down*（主机 -> 目标） *缓冲区描述符*，最多允许的最大通道数。
对于 Up 缓冲区，

- 写入指针仅由目标写入，并且
- 读取指针仅由调试探针（J-Link、Host）写入。

对于向下缓冲区：

- 写入指针仅由调试探针（J-Link、Host）写入，并且
- 读取指针仅由目标写入。

这可确保不会发生任何争用条件。 当读取和写入指针指向同一元素时，缓冲区为空。

缓冲区

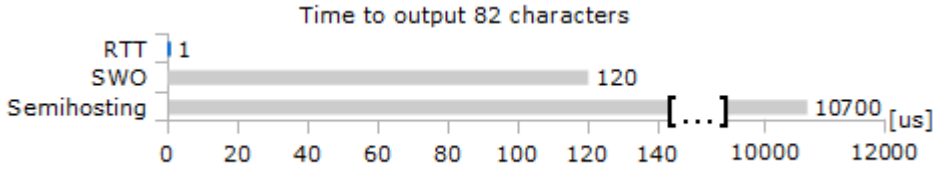
环形缓冲区缓冲区也位于 RAM 中，但不是 RTT CB 的一部分。 缓冲区大小可以针对每个通道和每个方向单独配置。 上图中缓冲区的灰色区域显示了包含有效数据的区域。

要求

SEGGER RTT 不需要任何额外的引脚或硬件，尽管 J-Link 通过标准调试端口连接到目标。 它不需要对目标或调试环境进行任何配置，甚至可以与不同的目标速度一起使用。
RTT 可以与正在运行的调试会话并行使用，而不会受到干扰，也无需任何 IDE 或调试器。

性能

SEGGER RTT 的性能明显高于用于将数据输出到主机 PC 的任何其他技术。 平均一行文本可以在一微秒或更短的时间内输出。 基本上只有做一个 memcopy（） 的时间。



内存占用

RTT 实现代码使用 ~500 字节的 ROM 和 24 字节的 ID + 每通道 24 字节的控制块作为 RAM 中的控制块。 每个通道都需要一些缓冲区内存。 建议的大小为上行通道 1 kByte，下行通道为 16 至 32 Bytes，具体取决于输入/输出的负载。

控制块检测

RTT 允许 J-Link （自动）检测给定 RAM 范围内的控制块 （CB）。默认情况下，使用自动检测，但用户也可以提供特定的控制块地址或要搜索 CB 的特定 RAM 范围。当 RTT 在主机上处于活动状态时，可以通过 J-Link RTT Viewer 等应用程序直接使用 RTT，或者通过 Telnet 连接到使用 J-Link 的应用程序（如调试器），J-Link 通过矢量表中指定的地址、J-Link 软件（自动检测）和/或用户（用户选择）指定的目标 RAM 区域自动搜索 SEGGER RTT 控制块。

自动检测

默认情况下，自动检测用于定位 RTT 控制块 （CB）。这意味着 J-Link 将首先尝试在偏移0x20的矢量表中找到控制块的地址。应将0x2添加到地址中，以将其标记为有效的 RTT 控制块地址。请参阅示例矢量表以获取帮助。如果找不到地址，J-Link 将搜索 J-Link 软件已知的指定目标的 RAM。

> **注意：**
>
> 对于大多数 MCU，在选择自动检测时，并非所有 RAM 区域都会被搜索。造成这种情况的原因有多种：

- RAM 中的搜索区域越大，定位控制块所需的时间就越长（在具有巨大 RAM 的目标上> 30 秒）
- 并非所有 RAM 区域都可能始终可用，具体取决于目标状态，在非活动状态时访问它们可能会导致未定义的行为或硬故障：
  - RAM 可以配置。
  - 在某些 MCU 状态下，RAM 可能无法访问（例如 TrustZone 安全/非安全区域）
  - 默认情况下可能未启用 RAM，或者可能由用户禁用。
  - RAM可能无法通过选定的MEM-AP访问（仅限非Cortex-M目标）
  - ...
- ...

手动指定控制块位置

虽然自动检测 RTT 控制块位置在大多数情况下工作正常，始终可以手动指定控制块的确切位置或特定地址范围J-Link 应在其中搜索控制块。
这可以通过多种方式完成。 例如：

- 使用 J-Link J-Link 命令字符串，例如在 J-Link 脚本文件中：
  - 设置RTTAddr
  - SetRTTSearchRanges
- 通过使用的工具（如果支持），例如在 J-Link RTT 查看器的配置对话框中
- 通过 J-Link TELNET 通道

RTT 频道

> **注意：**
> 本节涉及 RTT **通道**，不得与 RTT 终端混淆。
> 有关 RTT 终端的信息，请参阅 J-Link RTT 查看器文章。

RTT 通道由配置和环形缓冲区组成，用于在特定方向上传输数据（向上：目标 -> 主机，向下：主机 ->目标）。

通道配置

上下通道可以单独配置：

- 通道缓冲区大小：用于数据传输的通道缓冲区的大小。
- 通道缓冲模式（见下文）。

> **注意：**
>
> RTT 通道 0 有些特殊：
> 它是在编译时配置的，一旦调用 SEGGER_RTT_Init（） 或任何 RTT 写入函数，就会立即设置。 这样做是为了提供开箱即用的工作体验，不需要任何手动设置。

- 通道 0 将使用 RTT 配置部分中提到的参数定义进行设置。
- 只能手动（重新）配置通道 0 的标志。调用 SEGGER_RTT_ConfigDownBuffer（） 和 SEGGER_RTT_ConfigUpBuffer（） 时，所有其他配置参数都将被忽略。
- 所有其他通道都需要使用 SEGGER_RTT_ConfigDownBuffer（） 和 SEGGER_RTT_ConfigUpBuffer（） 手动设置各自的缓冲区。

通道缓冲模式

用户可以选择 3 种通道模式之一（一种阻塞模式和两种非阻塞模式）：

- 在阻塞模式下，应用程序将在缓冲区已满时等待，直到可以写入所有内存，从而导致应用程序状态被阻止，但防止数据丢失。
- 在非阻塞模式下，只会写入缓冲区的数据，或者根本不写入缓冲区的数据，其余的将被丢弃。

这允许实时运行，即使没有连接调试器，因此开发人员不必创建特殊的调试版本，并且代码可以保留在发布应用程序中。

定义	意义
阻塞	当缓冲区已满时，应用程序将等待，直到可以写入所有内存，从而导致应用程序状态被阻止，但防止数据丢失。
非阻塞，跳过	如果 up 缓冲区没有足够的空间来容纳所有传入数据，则所有数据都将被丢弃。
无阻塞，修剪	如果上行缓冲区没有足够的空间来容纳所有传入数据，则可用空间将被传入数据填充，其余空间将被丢弃。

请参阅： 通道缓冲区配置

RTT作模式

SEGGER RTT 可以在三种不同的作模式下运行。

注意：
不要与通道缓冲区配置模式混淆。

模式	解释	探头支持
后台模式	它是最快的模式，传输速度高达 ~2MB/s	■ 所有当前的 J-Links / J-Trace Pro 型号。
旧版背景模式	后台模式的旧版本，不附带探头固件端的处理，导致传输速度降低。	■ 所有当前的 Flasher 型号 ■ J-Link 底座 / Plus < V9 ■ J-Link Pro / Ultra+ < V4 ■ J-Trace Pro < V2

停止模式	伪 RTT 模式。CPU 停止读取数据。为不支持后台访问的 CPU 引入	■ 所有 J-Link / J-Trace Pro / Flasher 型号

后台模式

与传统后台模式相同，但在探头固件端进行了额外的实现，这使得传输速度显着提高（高达 2 MB/s）。

旧版背景模式

这是 RTT 最初引入的模式。在此模式下，J-Link 可以在 MCU + 应用程序继续运行（后台内存访问）的同时访问目标系统的内存， 实际上不会影响应用程序的实时行为。为了使用这种模式，目标MCU需要支持后台内存访问。

支持后台模式的核心

- 皮质-M。有关概述，请参阅支持的核心列表 (https://www.segger.com/products/debug-probes/j-link/technology/cpus-and-devices/overview-of-supported-cpus-and-devices/)
- 基于瑞萨电子 RX 的器件。
- Cortex-A 32 位 （ARMv7-A）（可选。查看 Cortex-A 详细信息)
- Cortex-R 32 位 （ARMv7-R）（可选。查看 Cortex-R 详细信息)
- RISC-V（可选。查看 RISC-V 详细信息)

停止模式

在此模式下，J-Link 会暂时停止 CPU（中断目标应用程序的执行）以访问内存，并 内存访问完成后自动继续运行。 对实时行为的实际影响（停止时间）取决于设置（使用的目标接口速度、使用的目标接口、JTAG链的长度、实际使用的内核等）。J-Link 软件 V6.30 中引入
了此模式，以便还允许在不支持后台内存访问的设备/CPU 架构上使用 RTT。 此模式会影响应用程序的实时行为，但仍可用于大多数目标应用程序。

支持停止模式的内核

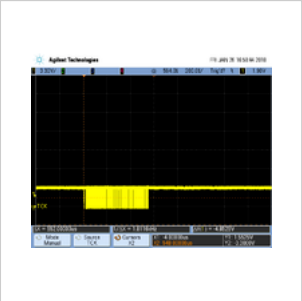
- 基于 Cortex-A 的设备（仅限 ARMv7-A!
- 基于 Cortex-R 的设备（仅限 ARMv7-R!
- 基于 RISC-V 的器件

> **注意：**
>
> - 不**支持** ARMv8-AR 上的停止模式。
> - 建议仅在目标接口速度为 **25 MHz 或更高的**界面速度（J-Link ULTRA+、J-Link PRO）下使用此模式，以尽可能减少对实时行为的影响。

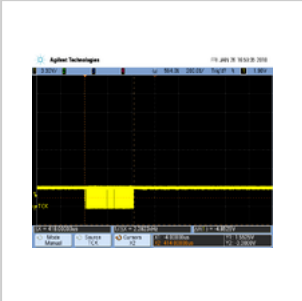
> 参考：J-Link 型号概述 (https://www.segger.com/products/debug-probes/j-link/models/model-overview/?mtm_campaign=kb&mtm_kwd=RTT)

对实时行为的影响

J-Link ULTRA+ @ 50 MHz 在停止模式下存储器读取访问的典型停止时间：

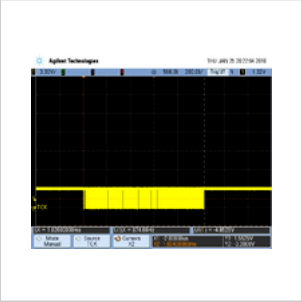


JTAG @ 50 MHz
256 字节，~552us

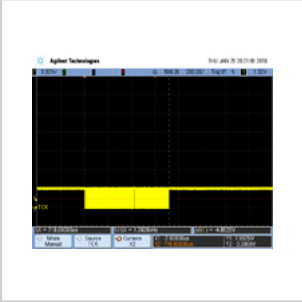


JTAG @ 50 MHz
4 字节，~418us

使用 **J-Link PLUS @ 15 MHz** 在停止模式下存储器读取访问的典型停止时间：



JTAG @ 15 MHz
256字节，~1.02ms



JTAG @ 15 MHz
4 字节，~718us

核心特定注意事项

Cortex-M 细节

如果 CPU 实现缓存：

- RTT 控制块以及所有 RTT 缓冲区必须开始缓存行对齐
- RTT 控制块以及所有 RTT 缓冲区的大小必须是缓存行的倍数
- 如果系统提供多个缓存级别，则控制块和缓冲区的对齐方式和大小必须以行大小最大的缓存为参考点。
- 控制块和 Up/Down 缓冲区 0 由 SEGGER_RTT.c 隐式定义。因此，为了确保这些满足对齐要求，需要指定SEGGER_RTT_CPU_CACHE_LINE_SIZE，以防所用系统中的缓存行大小不是 <= 32 字节。

- **用户应用程序**负责在段初始化完成后但在首次使用 RTT 之前调用 RTT 控制块上的缓存清理 + 无效 + 所有 RTT 缓冲区。这样做的好时机是在 main（） 中。这可以确保没有缓存行包含脏 RTT 数据/信息，否则这些数据/信息可能会在运行时的某个时间点被逐出到内存中。如果使用 embOS，通常可以安全地确保在调用 OS_InitHW（） 后使用 RTT，这会对所有缓存行执行干净 + 失效。
- 对于在运行时指定和初始化的 RTT 缓冲区（>索引 0），需要在调用任何 AllocBuffer（） / ConfigBuffer（） 函数之前执行缓存清理 + 失效。
- 在应用程序中，RTT 控制块、缓冲区和指向其名称的指针必须与虚拟地址 == 物理地址链接
- 应用程序必须向控制块 + 缓冲区所在的内存提供未缓存的地址别名。控制块以及所有缓冲区都需要通过该未缓存的地址别名进行访问。

Cortex-A 细节

- RTT 后台内存访问通过 AHB-AP 或 AXI-AP 执行
- AHB-AP 和 AXI-AP 的存在是可选的，取决于实际的 MCU（如果供应商实施了它们）
- 对于 J-Link 不知道“RTT 可用 AP”的设备：
  - 需要通过 J-Link 脚本文件指定用于后台内存访问的 AP。有关 J-Link 脚本文件的更多信息，请参阅 J-Link 脚本文件一文
- 强烈建议将 RTT 控制块和缓冲区放在内部存储器中，而不是外部存储器中。在基于 Cortex-A 的系统上，外部地址空间通常非常敏感，如果在初始化之前访问它（例如 DDR 控制器初始化完成等），则很容易导致锁定等。由于 J-Link 没有关于外部存储器空间初始化何时完成的信息，因此在外部存储器可用之前，可能会从 J-Link 端进行 RTT 访问。
- AHB/AXI-AP 访问的行为类似于 DMA 访问，并绕过 CPU、L1 / L2 等缓存
- 如果 CPU 实现缓存，请参阅 Cortex-M 详细信息。

Cortex-A MCU 示例

开箱即用支持 RTT 的示例 MCU：

- Xilinx Zynq 7000 系列，基于 Cortex-A9（从 V6.85b 开始支持 RTT）
- 恩智浦 i.MX6Solo 系列，基于 Cortex-A9，器件名称模式：MCIMX6Sx（从 V6.85d 开始支持 RTT）

emPower Zynq 板 (https://www.segger.com/evaluate-our-software/segger/empower-zynq/?mtm_campaign=kb&mtm_kwd=RTT)的示例项目：

- 项目下载
- 要求：
  - SEGGER 嵌入式工作室 (https://www.segger.com/products/development-tools/embedded-studio/?mtm_campaign=kb&mtm_kwd=RTT)V5.10a 或更高版本

- J-Link 软件 V6.85b 或更高版本

Cortex-R 细节

对于 Cortex-R 上的 RTT，这与 Cortex-A 细节相同。

RISC-V 细节

- RTT 后台存储器访问通过 RISC-V 系统总线访问 （SBA） 或通过 AHB/AXI-AP 执行。
- SBA 支持的存在是可选的，取决于实际的 MCU。
- AHB/AXI-AP 访问仅适用于 RISC-V 内核位于 ARM CoreSight DAP 后面的设置。

通过 SBA 提供支持

- 无需特定设置。J-Link 会自动检测 SBA 支持的存在，并且可以开箱即用地使用 RTT。

通过 MEM-AP 支持

- AHB-AP 和 AXI-AP 的存在是可选的，取决于实际的 MCU（如果供应商实施了它们）
- 从 J-Link 软件 V7.50 开始支持这种访问方法
- 对于 J-Link 不知道“RTT 可用 AP”的设备：
  - 需要通过 J-Link 脚本文件指定用于后台内存访问的 AP。有关 J-Link 脚本文件的更多信息，请参阅 J-Link 脚本文件一文
- 强烈建议将 RTT 控制块和缓冲区放在内部存储器中，而不是外部存储器中。在基于 Cortex-A 的系统上，外部地址空间通常非常敏感，如果在初始化之前访问它（例如 DDR 控制器初始化完成等），则很容易导致锁定等。由于 J-Link 没有关于外部存储器空间初始化何时完成的信息，因此在外部存储器可用之前，可能会从 J-Link 端进行 RTT 访问。
- AHB/AXI-AP 访问的行为类似于 DMA 访问，并绕过 CPU、L1 / L2 等缓存
- 如果 CPU 实现缓存：
  - RTT 控制块以及所有 RTT 缓冲区必须开始缓存行对齐
  - RTT 控制块以及所有 RTT 缓冲区的大小必须是缓存行的倍数
  - 如果系统提供多个缓存级别，则控制块和缓冲区的对齐方式和大小必须以行大小最大的缓存为参考点。
  - 控制块和 Up/Down 缓冲区 0 由 SEGGER_RTT.c 隐式定义。因此，为了确保这些满足对齐要求，需要指定SEGGER_RTT_CPU_CACHE_LINE_SIZE，以防所用系统中的缓存行大小不是 <= 32 字节。
  - **用户应用程序**负责在段初始化完成后但在首次使用 RTT 之前调用 RTT 控制块上的缓存清理 + 无效 + 所有 RTT 缓冲区。这样做的好时机是在 main（） 中。这可以确保没有缓存行包含脏 RTT 数据/信息，否则这些数据/信息可能会在运行时的某个时间点被逐出到内存中。如果使用 embOS，通常可以安全地确保在调用 OS_InitHW（） 后使用 RTT，这会对所有缓存行执行干净 + 失效。

- 对于在运行时指定和初始化的 RTT 缓冲区（>索引 0），需要在调用任何 AllocBuffer（） / ConfigBuffer（） 函数之前执行缓存清理 + 失效。
- 在应用程序中，RTT 控制块、缓冲区和指向其名称的指针必须与虚拟地址 == 物理地址链接
- 应用程序必须向控制块 + 缓冲区所在的内存提供未缓存的地址别名。控制块以及所有缓冲区都需要通过该未缓存的地址别名进行访问。

在 ARMv8-AR 上选择 MEM-AP

请参阅 ARMv8-AR 调试细节页面。

实现

SEGGER RTT 实现代码是用 ANSI C 编写的，只需添加可用源即可集成到任何嵌入式应用程序中。 RTT 可以通过简单易用的 API 使用。 甚至可以覆盖与 RTT 一起使用的标准 printf（） 函数。 使用 RTT 可将输出所需的时间降至最低，并允许在应用程序执行时间关键的实时任务时将调试信息打印到主机。

实现代码还包括一个简单的 printf（） 版本，可用于通过 RTT 编写格式化字符串。 它比大多数标准库 printf（） 实现更小，不需要堆，只需要可配置的堆栈量。

SEGGER RTT 实现在编译时可通过预处理器定义进行完全配置。 可以设置通道数和默认通道的大小。 使用可定义的 Lock（） 和 Unlock（） 例程可以使读取和写入成为任务安全。

源下载

RTT 源可在 SEGGER GitHub (https://github.com/SEGGERMicro/RTT) 上获得。

RTT 初始化

> **注意：**
>
> RTT 实现期望控制块的 RAM 位置处于以下两种状态之一：
>
> 1. 0 初始化，当 RTT 尚未设置时。这是大多数用户的默认情况。
> 2. 使用已设置的控制块。这是为了允许为多应用程序设置（例如引导加载程序 + 主应用程序）提供共享控制块。
>
> 在任何其他情况下，**必须**先调用 SEGGER_RTT_Init（） 才能调用任何其他 RTT 函数！

API 函数

RTT 实现中提供了以下 API 函数。 要使用它们，必须将 SEGGER_RTT.h 包含在调用源中。

API 函数
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（）
SEGGER_RTT_GetAvailWriteSpace（）

SEGGER_RTT_ConfigDownBuffer（）

通过指定其名称、大小和标志来配置或添加关闭缓冲区。

语法

```
int SEGGER_RTT_ConfigDownBuffer （无符号 BufferIndex， const char* sName， char* pBuffer， int BufferSize， int Flags）；
```

参数	意义
缓冲区索引	要配置的缓冲区的索引。 必须低于 SEGGER_RTT_MAX_NUM_DOWN_CHANNELS。
s名称	指向要显示为通道名称的 0 结尾字符串的指针。
p缓冲区	指向通道使用的缓冲区的指针。
缓冲区大小	缓冲区的大小（以字节为单位）。
标志	通道的标志（阻塞或非阻塞）。 标志[31：24]：用于有效性检查，必须为零。 标志[23：2]：保留供将来使用。 标志[1：0]：RTT作模式。

返回值

价值	意义
>= 0	好吧

< 0	错误

例

```
//
// Configure down buffer 1
//
SEGGER_RTT_ConfigDownBuffer(1, "DataIn", &abDataIn[0], sizeof(abDataIn),
                            SEGGER_RTT_MODE_NO_BLOCK_SKIP);
```

其他信息

配置缓冲区后，只需更改缓冲区的标志。

SEGGER_RTT_ConfigUpBuffer（）

通过指定其名称、大小和标志来配置或添加启动缓冲区。

语法

```
int SEGGER_RTT_ConfigUpBuffer （无符号 BufferIndex， const char* sName， char* pBuffer， int BufferSize， int Flags）；
```

参数	意义
缓冲区索引	要配置的缓冲区的索引。 必须低于SEGGER_RTT_MAX_NUM_UP_CHANNELS。
s名称	指向要显示为通道名称的 0 结尾字符串的指针。
p缓冲区	指向通道使用的缓冲区的指针。
缓冲区大小	缓冲区的大小（以字节为单位）。
标志	通道的标志（阻塞或非阻塞）。 标志[31：24]：用于有效性检查，必须为零。 标志[23：2]：保留供将来使用。 标志[1：0]：RTT作模式。

返回值

价值	意义
>= 0	好吧
< 0	错误

例

```
//
// Configure up buffer 1 to work in blocking mode
```

```
//
SEGGER_RTT_ConfigUpBuffer(1, "DataOut", &abDataOut[0], sizeof(abDataOut),
                          SEGGER_RTT_MODE_BLOCK_IF_FIFO_FULL);
```

其他信息

配置缓冲区后，只需更改缓冲区的标志。

SEGGER_RTT_GetKey（）

从 SEGGER RTT 缓冲区 0 中读取一个字符。 主机之前已将数据存储在那里。

语法

```
int SEGGER_RTT_GetKey（无效）；
```

返回值

价值	意义
>= 0	已阅读的字符 （0 - 255）。
< 0	没有可用字符（空缓冲区）。

例

```
int c;
c = SEGGER_RTT_GetKey();
if (c == 'q') {
  exit();
}
```

SEGGER_RTT_HasKey（）

检查 SEGGER RTT 缓冲区中是否至少有一个字符可供读取。

语法

```
int SEGGER_RTT_HasKey（无效）；
```

返回值

价值	意义
1	缓冲区中至少有一个字符可用。

0	没有字符可供读取。

例

```
if (SEGGER_RTT_HasKey()) {
  int c = SEGGER_RTT_GetKey();
}
```

SEGGER_RTT_Init（）

初始化 RTT 控制块。

语法

无效SEGGER_RTT_Init（无效）；

其他信息

- 应在应用程序启动时用于 RAM 目标。
- 如果 RTT CB 链接到非 0 初始化的 RAM 部分，则**必须**是第一个 SEGGER_RTT_* 调用。

SEGGER_RTT_printf（）

向主机发送格式化的字符串。

语法

int SEGGER_RTT_printf （无符号 BufferIndex， const char * sFormat， ...）

参数	意义
缓冲区索引	要将字符串发送到的向上通道的索引。
s格式	指向格式字符串的指针，后跟用于转换的参数。

返回值

价值	意义
>= 0	已发送的字节数。

< 0	错误。

例

```
SEGGER_RTT_printf(0, "SEGGER RTT Sample. Uptime:  %.10dms.", /*OS_Time*/ 890912);
// Formatted output on channel 0: SEGGER RTT Sample. Uptime: 890912ms.
```

其他信息

1. 转换规范具有以下语法：
   %[flags][FieldWidth][。精度]转换说明符%

   > **注意：**
   > 除了以下说明符之外，所有说明符都将忽略 Flags、FieldWidth 和 Precision：d、u、x

2. 支持的标志：

旗	意义
-	在字段宽度内左对齐
+	始终打印签名扩展名以进行签名转换
0	用 0 代替空格填充。使用 '-'-flag 或 precision 时忽略

3. 支持的转换说明符：

转换说明符	意义
c	Print the argument as one char
d	Print the argument as a signed integer
u	Print the argument as an unsigned integer
x	Print the argument as an hexadecimal integer
s	Print the string pointed to by the argument
p	Print the argument as an 8-digit hexadecimal integer. (Argument shall be a pointer to void.)

SEGGER_RTT_Read()

Read characters from any RTT down channel which have been previously stored by the host.

Syntax

```
unsigned SEGGER_RTT_Read (unsigned BufferIndex, char* pBuffer, unsigned BufferSize);
```

Parameter	Meaning
BufferIndex	Index of the down channel to read from.
pBuffer	Pointer to a character buffer to store the read characters.
BufferSize	Number of bytes available in the buffer.

Return value

Value	Meaning
>= 0	已读取的字节数。

例

```
char acIn[4];
unsigned NumBytes = sizeof(acIn);
NumBytes = SEGGER_RTT_Read(0, &acIn[0], NumBytes);
if (NumBytes) {
  AnalyzeInput(acIn);
}
```

SEGGER_RTT_SetTerminal（）

设置“虚拟”终端以在通道 o 上发送以下数据。

语法

```
void SEGGER_RTT_SetTerminal（char TerminalId）；
```

参数	意义
终端 ID	虚拟终端的 ID （0-9）。

例

```
//
// Send a string to terminal 1 which is used as error out.
//
SEGGER_RTT_SetTerminal(1); // Select terminal 1
SEGGER_RTT_WriteString(0, "ERROR: Buffer overflow");
SEGGER_RTT_SetTerminal(0); // Reset to standard terminal
```

其他信息

通过通道 o 发送的所有后续数据都将打印在设置的终端上，直到下一次更改。

SEGGER_RTT_TerminalOut（）

将一个字符串发送到特定的“虚拟”终端。

语法

```
int SEGGER_RTT_TerminalOut （char TerminalID， const char* s）；
```

参数	意义
终端 ID	虚拟终端的 ID （0-9）。
s	指向要发送的以 0 结尾的字符串的指针。

返回值

价值	意义
>= 0	发送到终端的字节数。
< 0	错误。

例

```
//
// Sent a string to terminal 1 without changing the standard terminal.
//
SEGGER_RTT_TerminalOut(1, "ERROR: Buffer overflow.");
```

其他信息

SEGGER_RTT_TerminalOut不会影响通过通道 0 发送的后续数据。

SEGGER_RTT_Write（）

在 RTT 通道上向主机发送数据。

语法

```
无符号SEGGER_RTT_Write（无符号 BufferIndex、const char* pBuffer、无符号 NumBytes）；
```

参数	意义
缓冲区索引	要将数据发送到的向上通道的索引。
p缓冲区	指向要发送的数据的指针。
字节数	要发送的字节数。

返回值

价值	意义

>= 0	已发送的字节数。
< 0	错误。

其他信息

使用 SEGGER_RTT_Write（） 各种数据，不仅可以发送可打印的数据。

SEGGER_RTT_WaitKey（）

等待，直到 SEGGER RTT 缓冲区 0 中至少有一个字符可用。 一旦字符可用，就会读取并返回该字符。

语法

```
int SEGGER_RTT_WaitKey（无效）；
```

返回值

价值	意义
>= 0	已阅读的字符 （0 - 255）。

例

```
int c = 0;
do {
  c = SEGGER_RTT_WaitKey();
} while (c != 'c');
```

SEGGER_RTT_WriteString（）

通过 RTT 将 0 结尾的字符串写入向上通道。

语法

```
无符号SEGGER_RTT_WriteSting（无符号 BufferIndex，const char* s）；
```

!参数	意义
缓冲区索引	要将字符串发送到的向上通道的索引。
s	指向要发送的以 0 结尾的字符串的指针。

返回值

价值	意义

>= 0	已发送的字节数。

例

```
SEGGER_RTT_WriteString(0, "Hello World from your target.\n");
```

SEGGER_RTT_GetAvailWriteSpace（）

返回环形缓冲区中可用的字节数。

语法

无符号SEGGER_RTT_GetAvailWriteSpace （无符号 BufferIndex）；

参数	意义
缓冲区索引	应检查空间的上行通道的索引。

返回值

价值	意义
>= 0	所选上行缓冲区中可用字节数。

例

```
unsigned NumBytesFree;

NumBytesFree = SEGGER_RTT_GetAvailWriteSpace(0);
```

配置定义

RTT 配置

定义/例程	意义
SEGGER_RTT_MAX_NUM_DOWN_BUFFERS	下行（到目标）通道的最大数量。
SEGGER_RTT_MAX_NUM_UP_BUFFERS	最大向上（到主机）通道数。
BUFFER_SIZE_DOWN	默认下行通道 0 的缓冲区大小。
BUFFER_SIZE_UP	默认上行通道 0 的缓冲区大小。
SEGGER_RTT_PRINT_BUFFER_SIZE	SEGGER_RTT_printf批量发送字符的缓冲区大小。
SEGGER_RTT_LOCK（）	锁定例程，以防止 RTT作中的中断和任务切换。
SEGGER_RTT_UNLOCK（）	解锁例程，以允许在 RTT作后进行中断和任务切换。

SEGGER_RTT_IN_RAM	指示整个应用程序都在 RAM 中，以防止通过定义为 1 来错误地识别 init 段中的 RTT 控制块。
SEGGER_RTT_SECTION	（可选）要将 RTT 控制块链接到的部分名称。可用于通过链接器文件控制 RTT 控制块位置。默认值（未定义）： ■ 如果设置了SEGGER_RTT_BUFFER_SECTION：与SEGGER_RTT_BUFFER_SECTION相同 ■ 如果**未**设置SEGGER_RTT_SECTION：没有特定部分。
SEGGER_RTT_BUFFER_SECTION	（可选）要将默认缓冲区链接到的节名名称。可用于通过链接器文件控制 RTT CB 默认缓冲区位置。默认值（未定义）： ■ 如果设置了SEGGER_RTT_SECTION：与SEGGER_RTT_SECTION相同 ■ 如果**未**设置SEGGER_RTT_SECTION：没有特定部分。

通道缓冲区配置

定义	意义
SEGGER_RTT_MODE_BLOCK_IF_FIFO_FULL	如果启动缓冲区已满，则对写入函数的调用将阻塞。
SEGGER_RTT_MODE_NO_BLOCK_SKIP	如果 up 缓冲区没有足够的空间来容纳所有传入数据，则不会将任何内容写入缓冲区。
SEGGER_RTT_MODE_NO_BLOCK_TRIM	如果 up 缓冲区没有足够的空间来容纳所有传入数据，则可用空间将被传入数据填充，同时丢弃任何多余的数据。

注意：

- SEGGER_RTT_TerminalOut（） 确保始终发送隐式终端切换命令，即使在使用非阻塞模式时也是如此。
- 缓冲区配置**仅影响目标设备缓冲区，而不影响主机缓冲区**。

颜色控制序列

配置	意义
RTT_CTRL_RESET	重置文本颜色和背景颜色。
RTT_CTRL_TEXT_*	将文本颜色设置为以下颜色之一。 ■ 黑 ■ 红 ■ 绿 ■ 黄色 ■ 蓝

	■ 品红 ■ 青色 ■ WHITE（浅灰色） ■ BRIGHT_BLACK（深灰色） ■ BRIGHT_RED ■ BRIGHT_GREEN ■ BRIGHT_YELLOW ■ BRIGHT_BLUE ■ BRIGHT_MAGENTA ■ BRIGHT_CYAN ■ BRIGHT_WHITE
RTT_CTRL_BG_*	将背景色设置为以下颜色之一。 ■ 黑 ■ 红 ■ 绿 ■ 黄色 ■ 蓝 ■ 品红 ■ 青色 ■ WHITE（浅灰色） ■ BRIGHT_BLACK（深灰色） ■ BRIGHT_RED ■ BRIGHT_GREEN ■ BRIGHT_YELLOW ■ BRIGHT_BLUE ■ BRIGHT_MAGENTA ■ BRIGHT_CYAN ■ BRIGHT_WHITE

优化性能

用户可以采取一些措施来提高 RTT 性能。当应该使用 RTT 传输大量数据时，这可能是必需的。

- 升级到 J-Link PRO (https://www.segger.com/products/debug-probes/j-link/models/j-link-pro/) 或 J-Link ULTRA+ (https://www.segger.com/products/debug-probes/j-link/models/j-link-ultra-plus/) 等高端 J-Link 型号，并使用更高的目标接口速度。
  注意：提高目标接口速度时，请务必保持在目标硬件的规格范围内。
- 增加目标设备上的缓冲区大小。

- 以多个小块而不是一个大块写入数据。
- 如果可能，防止额外的调试任务（例如并行调试），因为这些任务优先于 RTT 后台任务。
- 确保 J-Link 和主机（USB/以太网）之间的连接稳定。

ARM Cortex - 后台内存访问

在 ARM Cortex 目标上，RTT 所需的后台内存访问是通过所谓的 AHB-AP 执行的，该 AHB-AP 类似于 DMA，但只能由调试探针访问。 虽然在 Cortex-M 目标上始终存在 AHB-AP，但在 Cortex-A 和 Cortex-R 目标上，这是一个可选组件。 Cortex-A/R 目标可能实现多个 AP（有些甚至根本不是 AHB-AP），因此为了在 Cortex-A/R 目标上使用 RTT， 需要手动指定用于 RTT 后台内存访问的 AP 的索引，即 AHB-AP。
这是通过以下 J-Link 命令字符串完成的： CORESIGHT_SetIndexAHBAPToUse。

示例代码

示例主

```
/*********************************************************************
*                    SEGGER Microcontroller GmbH                     *
*       Solutions for real time microcontroller applications         *
**********************************************************************
*                                                                    *
*            (c) 1995 - 2018 SEGGER Microcontroller GmbH             *
*                                                                    *
*      www.segger.com Support: support@segger.com                    *
*                                                                    *
**********************************************************************

----------------------------------------------------------------------
File     : RTT.c
Purpose  : Simple implementation for output via RTT.
It can be used with any IDE.
---------------------------- END-OF-HEADER ---------------------------
*/

#include "SEGGER_RTT.h"

static void _Delay(int period) {
  int i = 100000*period;
  do { ; } while (i--);
}

int main(void) {
  int Cnt = 0;

  SEGGER_RTT_Init(void);
  SEGGER_RTT_WriteString(0, "Hello World from SEGGER!\n");
  do {
    SEGGER_RTT_printf(0, "%sCounter: %s%d\n",
                          RTT_CTRL_TEXT_BRIGHT_WHITE,
                          RTT_CTRL_TEXT_BRIGHT_GREEN,
                          Cnt);
    if (Cnt > 100) {
      SEGGER_RTT_TerminalOut(1, RTT_CTRL_TEXT_BRIGHT_RED"Counter overflow!");
      Cnt = 0;
    }
    _Delay(100);
    Cnt++;
  } while (1);
  return 0;
}
```

```
/************************** End of file ****************************/
```

示例矢量表

```
_vectors:
        VECTOR __stack_end__
        VECTOR Reset_Handler
        ISR_HANDLER NMI_Handler
        VECTOR HardFault_Handler
        ISR_HANDLER MemManage_Handler
        ISR_HANDLER BusFault_Handler
        ISR_HANDLER UsageFault_Handler
        ISR_RESERVED
        .word (_SEGGER_RTT + 2)  // RTT Control Block address. Needs +2 to be marked as a valid address.
        ISR_RESERVED
        ISR_RESERVED
        ISR_HANDLER SVC_Handler
        ISR_HANDLER DebugMon_Handler
        ISR_RESERVED
        ISR_HANDLER PendSV_Handler
        ISR_HANDLER SysTick_Handler
        //
        // Add external interrupt vectors here.
        // Example:
        //   ISR_HANDLER ExternalISR0
        //   ISR_HANDLER ExternalISR1
        //   ISR_HANDLER ExternalISR2
        //   ISR_HANDLER ExternalISR3
        //

        .section .vectors, "a"
        .size _vectors, .-_vectors
_vectors_end:
```

RTT 通信

可以使用不同的应用程序与目标上的 RTT 实现进行通信。 该功能甚至可以使用 J-Link SDK (https://www.segger.com/products/debug-probes/j-link/technology/j-link-sdk/?mtm_campaign=kb&mtm_kwd=RTT) 集成到定制应用程序中。
在目标应用程序中使用 RTT 变得容易。实现代码可免费下载，并且 可以集成到任何现有应用程序中。要通过 RTT 进行通信，可以使用任何 J-Link。通过 RTT 进行通信
的一种简单方法是使用 Telnet 客户端或类似客户端创建与 localhost：19021 的连接， 当与 J-Link 的连接（例如通过调试会话）处于活动状态时。在这种情况下，可以通过 SEGGER TELNET 配置字符串设置 RTT 信道。 默认通道为通道 0（终端）J-Link
软件和文档包附带一些更高级的应用程序，它们演示了用于不同目的的 RTT 功能：

- J-Link RTT 查看器。
- J-Link RTT 客户端。
- J-Link RTT 记录仪

将 RTT 集成到主机应用程序中

RTT 还可以通过以下两种方式之一集成到任何其他 PC 应用程序中，例如调试器或数据可视化器：

- 当 J-Link 连接处于活动状态时，应用程序可以建立与在 localhost：19021 上打开的 RTT Telnet 服务器的套接字连接（请参阅 J-Link 软件的 TELNET 通道）。

- 该应用程序创建了自己的 J-Link 连接，并使用 J-Link SDK (https://www.segger.com/products/debug-probes/j-link/technology/j-link-sdk/?mtm_campaign=kb&mtm_kwd=RTT) 中的 J-Link RTT API 直接配置和使用 RTT。

J-Link 软件的 TELNET 通道

J-Link 软件在本地主机端口（默认：**19021**）上提供了一个类似 TELNET 的通道，以允许从可能与调试会话并行运行的单独第三方应用程序轻松访问 RTT 数据。TELNET 通道使用起来非常简单，基本上它所需要的只是打开与 TELNET 端口的本地 TCP/IP 连接并开始接收（或发送）数据。

欲了解更多信息，请参阅：J-Link RTT TELNET 通道。

低功耗模式

在使用低功耗模式的目标应用上运行时，RTT 可能无法按预期工作。使用 RTT 时，请确保**不使用**低功耗模式。
有关 J-Link 和低功耗模式的更多信息，请参阅 J-Link 低功耗模式调试。

样品

- RTT 在STM32F3上
- RTT 在STM32F4上
- STM32H7上的 RTT

故障 排除

如果您在让 RTT 与您的设置配合使用时遇到问题，请尝试以下作：

- 确保目标设备支持 RTT。更多信息，请参阅模式
- 确保使用最新版本的 J-Link 软件包 (https://www.segger.com/downloads/jlink#J-LinkSoftwareAndDocumentationPack?mtm_campaign=kb&mtm_kwd=RTT)（例如 RTT 源代码、JLinkARM.dll、J-Link RTT 查看器、J-Link RTT 客户端等）
- 确保目标应用程序对每个缓冲区使用正确的 RTT 模式（**注意**：如果目标写入 RTT 数据的速度快于主机获取 RTT 数据的速度，则除 **SEGGER_RTT_MODE_BLOCK_IF_FIFO_FULL** 之外的每种模式都可能导致 RTT 数据丢失）
- 确保目标应用程序不使用低功耗模式。有关详细信息，请参阅低功耗模式
- 确保 RTT 控制块放置在 RAM 中，并且可通过调试接口读取。这也适用于 RTT 缓冲区。
- 确保主机端只有一个应用程序获取 RTT 数据。当并行运行多个应用程序（例如 J-Link RTT Viewer 和 J-Link RTT Client）时，一个应用程序可能会从另一个应用程序“*窃取*”数据，从而导致 RTT 日志碎片化
- 确保 Rtt 控制块可以通过 J-Link DLL 找到：
  - 要使自动检测正常工作，需要向 J-Link 传递正确的目标设备名称（仅目标内核名称**不足以进行**自动检测）

    - **注意**：对于许多设备，J-Link DLL 中仅指定了部分可用 RAM。如果 RTT 控制块位于此指定区域之外，则自动检测功能**将不起作用**，需要指定 RTT 控制块所在的 RAM 区域的地址或搜索范围。
  - RTT 控制块的地址可以使用 J-Link 命令字符串设置，或者在使用 J-Link RTT 查看器时，在配置对话框中设置。
  - J-Link 查找 RTT 控制块的地址范围可以使用 J-Link 命令字符串设置，或者在使用 J-Link RTT 查看器时在配置对话框中设置。
- 确保目标应用程序使用的正确 RTT 通道（例如 J-Link RTT Viewer 和 J-Link RTT 客户端仅支持通过通道 0 进行数据通信）。

常见问题

- **问：**J-Link 如何找到 RTT 缓冲液?
- **一个：** 有两种方法：如果调试器 （IDE） 知道 SEGGER RTT 控制块的地址，它可以将其传递给 J-Link。例如，这是由 J-Link 调试器完成的。如果使用另一个不感知 SEGGER RTT 的应用程序，则 J-Link 会在后台执行应用程序期间在已知目标 RAM 中搜索 ID。此过程通常只需几分之一秒，并且不会延迟程序执行。

- **问：**我正在调试一个纯 RAM 应用程序。J-Link 找到一个 RTT 缓冲区，但我没有得到任何输出。我该怎么办?
- **一个：** 如果应用程序的 init 部分存储在 RAM 中，J-Link 可能会错误地识别 init 部分中的块，而不是数据部分中的实际块。为防止这种情况，请将定义SEGGER_RTT_IN_RAM设置为 1。现在，J-Link 将找到正确的 RTT 缓冲区，但只有在调用应用程序中的第一个 SEGGER_RTT 函数之后。建议在应用程序开始时调用 SEGGER_RTT_Init（）。

- **问：**这也可以用于没有 SWO 引脚的目标吗?
- **一个：** 是的，使用调试接口。这可以是大多数 Cortex-M 设备上的 JTAG 或 SWD（仅限 2 引脚！），甚至可以是某些瑞萨设备上的 FINE 接口，就像英飞凌 SPD 接口（单引脚！

- **问：**这也可以用于 Cortex-M0 和 M0+ 吗?
- **一个：** 是的。

- **问：**某些终端输出 （printf） 解决方案在调试环境之外执行时会“崩溃”程序执行，因为它们使用软件断点来触发没有调试器的硬故障或停止，因为 SWO 未初始化。这使得无法在独立模式下运行调试构建。SEGGER-RTT 呢?
- **一个：** SEGGER-RTT 默认使用非阻塞模式，这意味着如果不存在调试器并且甚至没有连接 J-

Link，它不会停止程序执行。应用程序将继续工作。

- **问：**我没有看到任何输出，尽管在我的应用程序中使用 RTT 是正确的。我该怎么办？
- **一个：** 在某些情况下，J-Link 无法在已知的 RAM 区域中找到 RTT 缓冲区。在这种情况下，可以通过 J-Link exec 命令手动设置可能的区域或确切地址：
  - 设置要搜索 RTT 缓冲区的范围：SetRTTSearchRanges <RangeStart [Hex]> <RangeSize > [, <Range1Start [Hex]> <Range1Size>, ...]（例如“SetRTTSearchRanges 0x10000000 0x1000, 0x2000000 0x1000”）
  - 设置 RTT 缓冲区的地址：SetRTTAddr <RTTBufferAddress [十六进制]>（例如“SetRTTAddr 0x20000000”）
  - 通过 J-Link 控制面板设置 RTT 缓冲区的地址 -> RTTerminal

  注意：
  J-Link exec 命令可以在大多数应用程序中执行，例如在

  - J-Link Commander 通过“exec <Command>"
  - 通过“monitor exec <命令>的 J-Link GDB 服务器"
  - IAR EW 通过宏文件中的“__jlinkExecCommand（”<Command>“）;”。

本页面最后编辑于 2025 年 9 月 11 日 （星期五） 06：57。