SEGGER系统视图

记录和分析嵌入式系统的运行时行为

用户指南

文档:UM08027
软件版本:3.52
修订:0
日期:2023年7月4日

SEGGER Microcontroller GmbH的产品

www.segger.com

免责声明

本文档中的信息仅被假定为准确，不作任何保证。本手册中的信息可能因功能或性能改进而发生变更，恕不另行通知。SEGGER Microcontroller GmbH (SEGGER)不对本文中的任何错误或遗漏承担任何责任。SEGGER不承担关于产品适合特定用途的任何明示、暗示或法定的保证或条件。评估产品对任何特定用途的适用性是您的唯一责任。

版权声明

商标

本手册中提到的名称可能是其各自公司的商标。

品牌和产品名称是其各自持有人的商标或注册商标。

联系地址

SEGGER Microcontroller GmbH
Ecolab-Allee 5
D-40789 Monheim am Rhein

德国

电话+ 49-2173-99312-0。
传真。+ 49-2173-99312-28
电子邮件:support@segger.com
互联网:www.segger.com

学分

特别感谢Jean Labrosse的持续反馈、beta测试和好主意。

手册的版本

本手册描述了当前的软件版本。如果您发现手册中的错误或软件中存在问题，请告知我们，我们将尽可能尽快为您提供帮助。如需进一步了解尚未文档化的主题或功能，请与我们联系。

打印日期:2023年7月4日

Software	Revision	Date	By	Description
3.42	0	221208	PC	• Added Peak Load and Peak Used metrics to heap monitoring.
3.40	0	221208	PC	• Section "Heap" added. • Section "Dynamic memory monitor functions" added.
3.32	0	210426	JL	• Section "NuttX" added. • Section "Zephyr" added.
3.20	0	201013	PO	• Chapter "Overview" updated. • Section "Package Content" updated. • Section "Licensing" updatedt to refer to SFL. • Chapter "Getting Started" updated to match example recording. • Chapter "The SystemView Application" updated to match latest GUI. • Section "Command Line Options" added.
3.10	0	191213	JL	• Manual updated to SystemView V3.
2.52	4	180921	NV	• Section "FreeRTOS" added Note about number of tasks displayed by default.
2.52	3	180809	NV	• Section "No OS" added link to generic setup example.
2.52	2	180315	NV	• Section "FreeRTOS" updated for V10.
2.52	1	170907	JL	• Section "FreeRTOS" updated. • Section "uC/OS-II" added.
2.52	0	170818	JL	• Section "Command Line Options" added. • Section "Micrium OS Kernel" added. • Chapter "API reference" updated.
2.50	0	170426	JL	• Section "SystemView PRO" added. • Section "Event filter" added.
2.42	1	170306	AG	• Chapter "Supported CPUs" updated.
2.42	0	170209	JL	• Section "GUI Controls" updated. • Section "Trigger Modes" added.
2.40	0	160728	JL	• Chapter "Getting started with SystemView" updated. • Chapter "API reference" updated.
2.38	0	160624	JL	• Section "Renesas RX" to "Supported Devices" added.
2.36	0	160524	JL	• Section "Getting started with SystemView" updated. • Section "The SystemView Application" updated. • Section "Supported OSes" updated. • Section "Integration guide" updated.
2.34	0	160401	JL	• Section "Optimizing SystemView" added. • Section "uC/OS-III" to "Supported OSes" added.
2.32	1	160322	JL	• Chapter "Performance and resource usage" added.
2.32	0	160310	JL	• Section Supported OSes added.

Software	Revision	Date	By	Description
				• Post-Mortem mode added. • Chapters restructured by relevance. • Sample configuration for TI AM3350 Cortex-A8 added
2.30	0	160127	JL	• Section Using SystemViewer added. • Section Integrating SEGGER SystemView into an OS added. • Section Integrating SEGGER SystemView into a middleware module added. • Section Frequently asked questions added. • Section Supported CPUs added. • Section SEGGER SystemView API functions updated.
2.28	0	160114	JL	• Terminal Window description added. • Screenshots updated.
2.26	1	160106	JL	Configuration for embOS and FreeRTOS added.
2.26	0	151223	JL	• Printf functionality added.
2.24	0	151216	JL	• macOS and Linux version added.
2.22	0	151214	JL	• GUI and performance improvements.
2.20	1	151119	JL	• Screenshots updated. • Fixed defines in configuration.
2.20	0	151118	JL	• SystemViewer GUI elements restructured. • SystemView Config module added.
2.10	0	151106	JL	• Official Release.
2.09	0	151026	JL	• Initial Pre-Release.

关于本文档

假设

本文档假设您已经具备以下坚实的知识:

· 用于构建应用程序的软件工具(汇编器，链接器，C编译器)。
· C编程语言。
· 目标处理器。
· DOS命令行。

如果你觉得自己的C知识还不够，我们推荐Kernighan and Ritchie的《The *C Programming* language》(ISBN 0—13—1103628)，这本书描述了C编程中的标准，在更新的版本中，还涵盖了ANSI C标准。

如何使用这本手册呢

本手册解释了产品提供的所有功能和宏。它假设你有C语言的工作知识。不需要汇编编程知识。

语法的排版约定

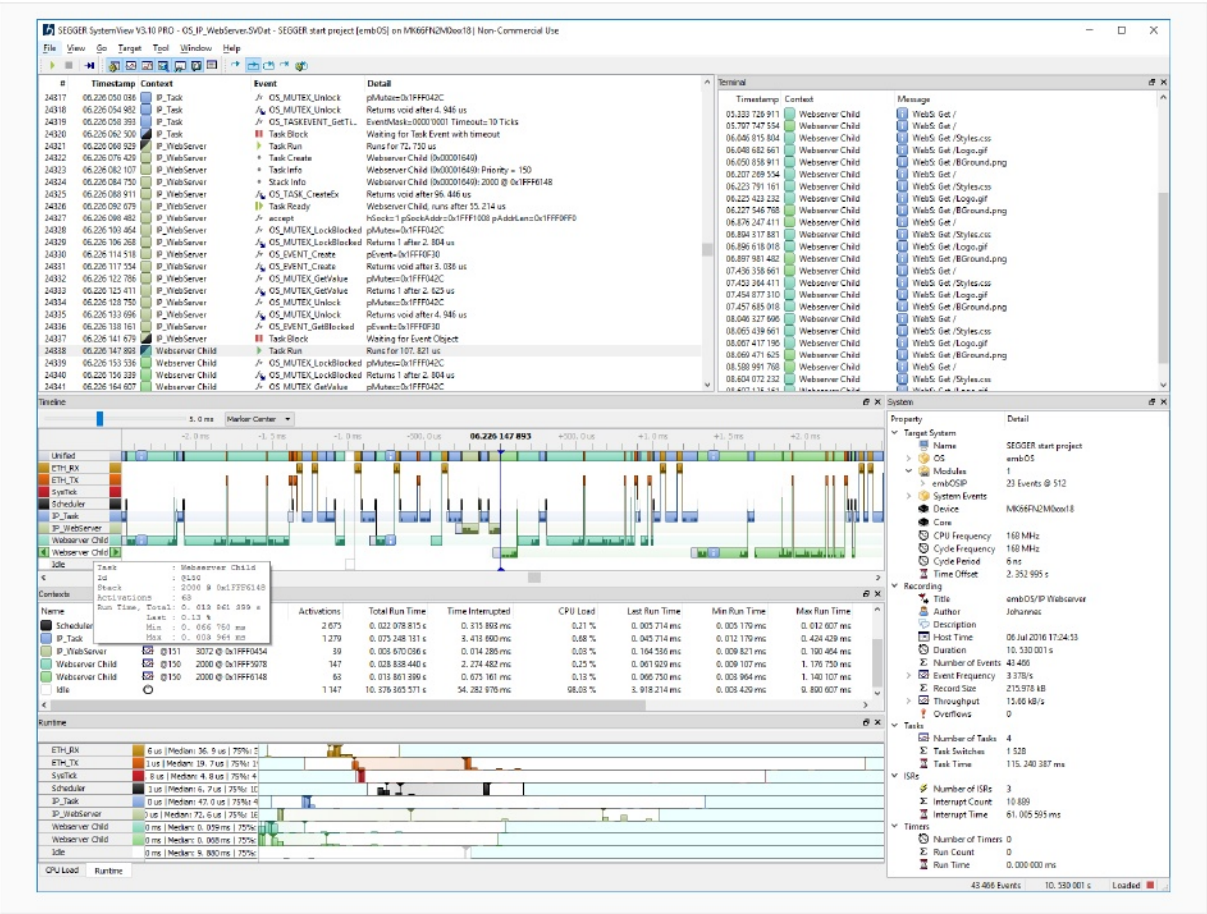
本手册使用以下排版约定:

Style	Used for
Body	Body text.
`Keyword`	Text that you enter at the command prompt or that appears on the display (that is system functions, file- or pathnames).
`Parameter`	Parameters in API functions.
`Sample`	Sample code in program examples.
`Sample comment`	Comments in program examples.
Reference	Reference to chapters, sections, tables and figures or other documents.
GUIElement	Buttons, dialog boxes, menu names, menu commands.
Emphasis	Very important sections.

目录表

第一章概述

本节概述SEGGER SystemView。



1.1 什么是SEGGER SystemView?

SystemView是一个用于对任何嵌入式系统进行可视化分析的工具包。SystemView提供了对应用程序的完整洞察，以获得对运行时行为的深刻理解，远远超出了调试器所提供的。当在具有多个任务和事件的复杂系统中开发和工作时，这是特别有利的。

SystemView由两部分组成:

· PC可视化*SystemView应用程序*
· 收集目标系统上遥测数据的代码。

SystemView应用程序允许对嵌入式系统的行为进行分析和概要分析。它记录由嵌入式系统产生的遥测数据，并以各种方式将这些信息可视化。这些记录可以保存到文件中，以供以后分析或作为系统的文档。

遥测数据通过调试接口、网络连接或串行线进行记录。当通过调试接口进行记录时，使用SystemView不需要额外的硬件(和额外的固定)。它可以在任何允许调试访问的系统上使用。

借助SEGGER J-Link及其*实时传输*(RTT)技术，SystemView可以连续记录、分析和实时可视化数据。

SystemView可以分析哪些中断、任务和软件计时器已经执行，执行的频率、准确时间和使用的时间。它揭示了到底发生了什么，以什么顺序，哪个中断触发了哪个任务切换，哪个中断和任务调用了底层模块的哪个API函数。

可以执行周期精确的分析，并且可以在系统中添加性能标记来测量计时。

SystemView可用于验证嵌入式系统的行为是否符合预期，并可用于发现问题和低效率，例如多余和虚假的中断、意外的任务更改或错误选择的任务优先级。它可以与任何(RT)操作系统一起使用，可以调用SystemView事件函数，也可以在没有仪表化RTOS或根本没有RTOS的系统中使用，用于分析中断执行和对用户功能进行计时，如时间关键子程序。

1.1.1 它是如何工作的?

在目标端，必须包含一个包含SYSTEMVIEW和RTT的小软件模块。SYSTEMVIEW模块收集并格式化监视器数据并将其传递给RTT。

目标系统在某些情况下(例如中断开始和中断结束)调用SYSTEMVIEW函数来监视事件。SystemView将这些事件与可配置的高精度时间戳一起存储。时间戳可以精确到1个CPU周期，相当于200 MHz CPU上的5 ns。

RTT模块将数据存储在目标缓冲区中，从而实现连续记录，以及单次记录和事后分析。

记录器接口从RTT缓冲区读取数据并将其发送到SystemView应用程序。

1.1.2 目标端需要什么资源?

RTT和SYSTEMVIEW模块的总ROM大小小于2kbyte。对于典型的系统来说，大约600字节的RAM就足够连续记录了。对于系统项触发的记录，缓冲区大小是由要记录的时间和事件的数量决定的。不需要其他硬件。典型事件(基于200 MHz的Cortex-M4 CPU)所需的CPU少于1 us，这在每秒10,000个事件的系统中导致不到1%的开销。由于使用调试接口(JTAG, SWD, FINE，…)来传输数据，因此不需要额外的引脚。

1.1.3 SystemView可以在哪些cpu上使用?

SystemView可以在任何CPU上使用。可以在J-Link RTT技术支持的任何系统上或使用网络连接或串行线进行连续实时记录。RTT要求在程序执行期间通过调试接口读取内存的能力，这通常在ARM Cortex-M0, M0+， M1, M3, M4, M7, M23, M33处理器以及所有瑞萨RX设备中得到支持。

在不支持RTT技术的系统上，也可以在系统停止时通过调试探针手动读取缓冲区内容，这允许单次记录，直到缓冲区被填满，并进行事后分析以捕获最新记录的数据。单发和死后记录可以由系统触发，以便能够控制记录何时开始和停止。

1.1.4 将其添加到目标系统中需要做多少工作?

不是很多。少量的文件需要添加到makefile或project中。如果操作系统支持SystemView，那么只需要调用一个函数。在没有RTOS或非仪表化RTOS的系统中，需要为每个应该被监视的中断或函数添加两行代码。这就是全部内容，应该不会超过几分钟。

1.2 SEGGER SystemView包

以下部分描述如何安装SEGGER SystemView包及其内容。

1.2.1 下载和安装

SEGGER SystemView包可用于Windows、macOS和Linux，作为安装程序设置和可移植存档。

从https://www.segger.com/sys-temview下载适用于您的操作系统的最新软件包。

为了进行实时录制，必须安装当前的J-Link软件和文档包。下载和使用说明可在https://www.segger.com/jlink上获得。

Windows安装程序

从http://www.segger.com/systemview下载最新的安装程序并执行。安装向导将引导您完成安装。

安装后，可以通过Windows*开始*菜单或文件资源管理器访问包内容。

Windows便携包

从http://www.segger.com/systemview下载最新的zip包，并将其解压缩到文件系统的任意目录。

无需安装，解压缩后的包内容即可直接使用。

macOS安装程序

从http://www.segger.com/systemview下载最新的pkg安装程序并执行。包安装程序将指导您完成安装。
安装后，SystemView应用程序可以通过Launchpad访问。

Linux系统要求

要在Linux上运行SystemView，必须在系统上安装Qt V4.8库。

Linux安装程序

从http://www.segger.com/sys-temview下载最新的Linux DEB或RPM安装程序并执行它。软件安装程序将指导您完成安装。

Linux便携包

从http://www.segger.com/systemview下载Linux的最新存档文件，并将其解压缩到文件系统上的任何目录。
无需安装，解压缩后的包内容即可直接使用。

目标的来源

从http://www.segger.com/systemview下载要包含在嵌入式应用程序中的最新源代码，并将其解压缩到您选择的文件夹中。
源接口与SEGGER软件，如embOS也包括在内。

1.2.2 包装内容

SEGGER SystemView包包括应用程序跟踪所需的一切-主机PC可视化SystemView应用程序和示例跟踪文件，以便快速轻松地开始。

下表列出了包的内容。

SystemView包

File	Description
./SystemView*	The SystemView analysis and visualization tool.
./Doc/UM08027_SystemView.pdf	This documentation.
./Doc/Release_SystemView.html	Release notes and revision history.
./Description/SYSVIEW_*.txt	SystemView API description files.
./Sample/FS_DeviceActivity.SVDat	Demonstrates the usage of the callback invoked on each device operation.
./Sample/FS_Performance.SVDat	SystemView Sample recording of SEGGER emFile, testing read and write performance of the Macronix NAND Flash (MX30LF1GE8ABTI) on the SEGGER emPower board (Freescale MK66FN2M0VMD18)
./Sample/OS_IP_WebServer.SVDat	SystemView sample trace file of a web server application.
./Sample/OS_Start_LEDBlink.SVDat	SystemView sample trace file of a simple embOS application.
./Sample/Sample_Overflow.SVDat	SystemView sample recording showing SystemView buffer overflows.
./Sample/uCOS_Start.SVDat	SystemView sample trace file of a simple uC/OS-III application.

目标源包

File	Description
./Src/Config/Global.h	Global data types for SystemView.
./Src/Config/SEGGER_RTT_Conf.h	SEGGER Real Time Transfer (RTT) configuration file.
./Src/Config/SEGGER_SYSVIEW_Conf.h	SEGGER SYSTEMVIEW configuration file.
./Src/Sample/COMM	Recorder via network connection using embOS and emNet.
./Src/Sample/embOS	Initialization and configuration of SystemView with embOS.
./Src/Sample/FreeRTOSV8	Initialization and configuration of SystemView with FreeRTOS V8.
./Src/Sample/FreeRTOSV9	Initialization and configuration of SystemView with FreeRTOS V9.
./Src/Sample/FreeRTOSV10	Initialization and configuration of SystemView with FreeRTOS V10.
./Src/Sample/MicriumOSKernel	Initialization and configuration of SystemView with the Micrium OS Kernel.
./Src/Sample/NoOS	Initialization and configuration of SystemView with no OS.
./Src/Sample/uCOS-II	Initialization and configuration of SystemView with uC/OS-II.
./Src/Sample/uCOS-III	Initialization and configuration of SystemView with uC/OS-III.

File	Description
./Src/SEGGER/SEGGER.h	Global types & general purpose utility functions.
./Src/SEGGER/SEGGER_RTT.c	SEGGER RTT module source.
./Src/SEGGER/SEGGER_RTT.h	SEGGER RTT module header.
./Src/SEGGER/SEGGER_RTT_ASM_ARMv7M.S	Optimized RTT routines for Cortex-M.
./Src/SEGGER/SEGGER_SYSVIEW.c	SEGGER SYSTEMVIEW module source.
./Src/SEGGER/SEGGER_SYSVIEW.h	SEGGER SYSTEMVIEW module header.
./Src/SEGGER/SEGGER_SYSVIEW_ConfDefaults.h	SEGGER SYSTEMVIEW configuration fallback.
./Src/SEGGER/SEGGER_SYSVIEW_Int.h	SEGGER SYSTEMVIEW internal header.
./Src/SEGGER/Syscalls/SEGGER_RTT_Syscalls_*.c	Sources for toolchain dependent low level routines for I/O via RTT.

1.3 许可

根据SEGGER的友好许可(https://www.segger.com/license-sfl)， SystemView可以免费用于非商业目的。对于任何其他用途，都需要商业使用许可。
非商业许可没有功能限制。SystemView支持无限制的记录，并附带更好的分析，搜索和过滤功能。
SystemView的商业用途许可可作为单用户许可以及组或公司范围的许可。欲了解更多信息，请参阅SEGGER的商业用途许可(https://www.segger.com/license-cul)。

1.3.1 非商业许可证

SystemView可以在非商业许可下用于评估、教育和爱好者目的。

当您在非商业许可下使用SystemView时，不需要激活。在启动SystemView应用程序时，会出现一个弹出窗口。单击“继续”接受许可条款。

第二章

开始使用SystemView应用程序

本节介绍如何开始使用SEGGER SystemView。它解释了如何基于监控数据分析应用程序。
本章介绍示例数据文件OS_IP_WebServer。SVDat是SEGGER SystemView包的一部分。

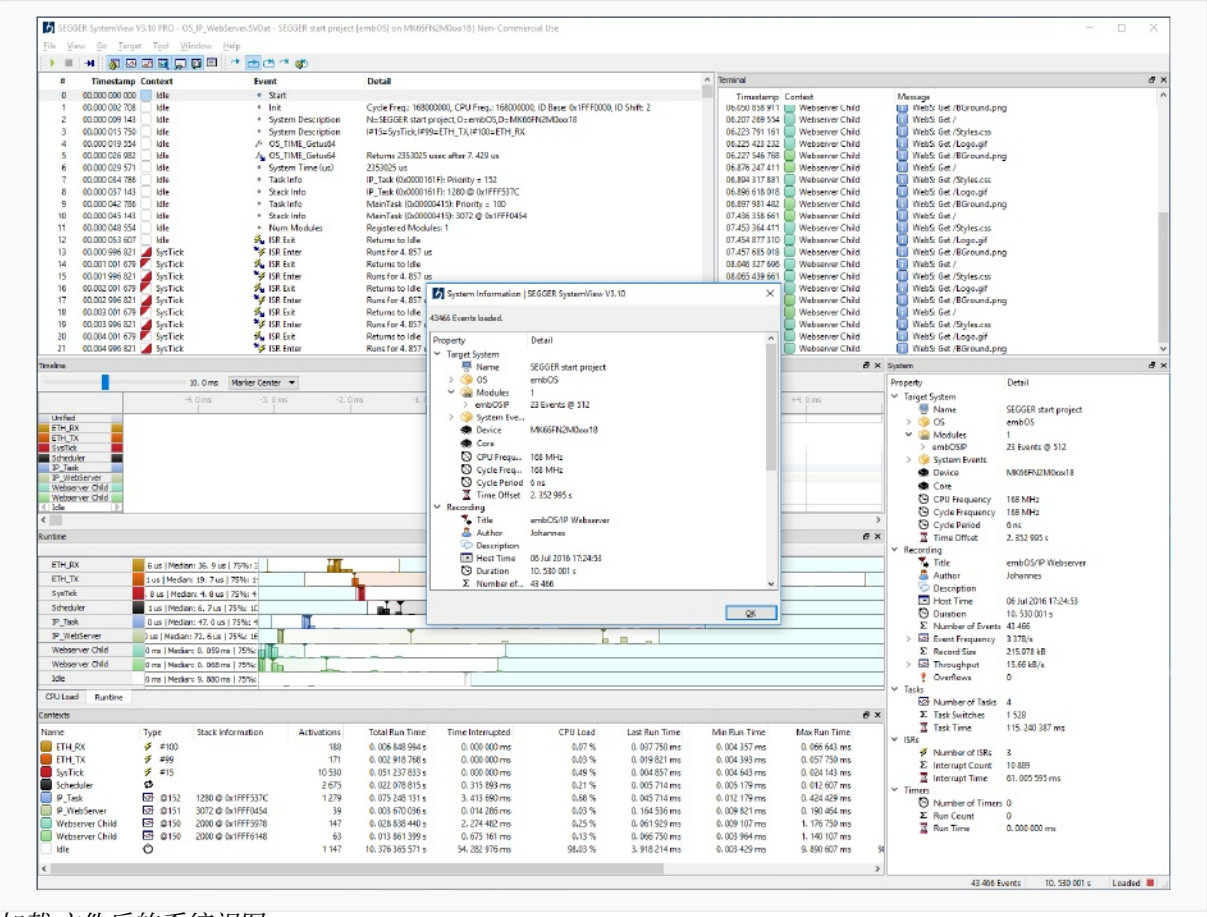
样本数据文件显示了运行emos RTOS、emNet TCP/IP栈和web服务器应用程序的目标系统的行为。
我们将分析应用程序如何处理来自SEGGER SystemView的信息。

2.1 启动SystemView并加载数据

SystemView可以实时监控来自目标应用程序的数据。被监控的数据可以保存到一个文件中，以供以后使用。保存的数据可以在没有J-Link甚至没有目标硬件或目标应用程序的情况下进行分析。这使得没有物理访问权限的开发人员可以对系统进行分析。

- 从Windows开始菜单或安装目录启动SystemView应用程序(SystemView.exe)。
- 在系统视图的第一次启动，它将打开样本记录。
- 在进一步启动时选择File→Recent Files→$PackageInstallationDir$/ Sample/OS_IP_WebServer.SVdat。

系统视图加载并分析数据，显示加载记录的系统信息，现在看起来应该是这样的:



加载文件后的系统视图

录音是为一个应用程序完成的，该应用程序创建了一个web服务器，当web浏览器访问时，该服务器提供了embOS/IP演示网页。样本数据已被在web服务器运行和浏览器多次加载网页时收集。

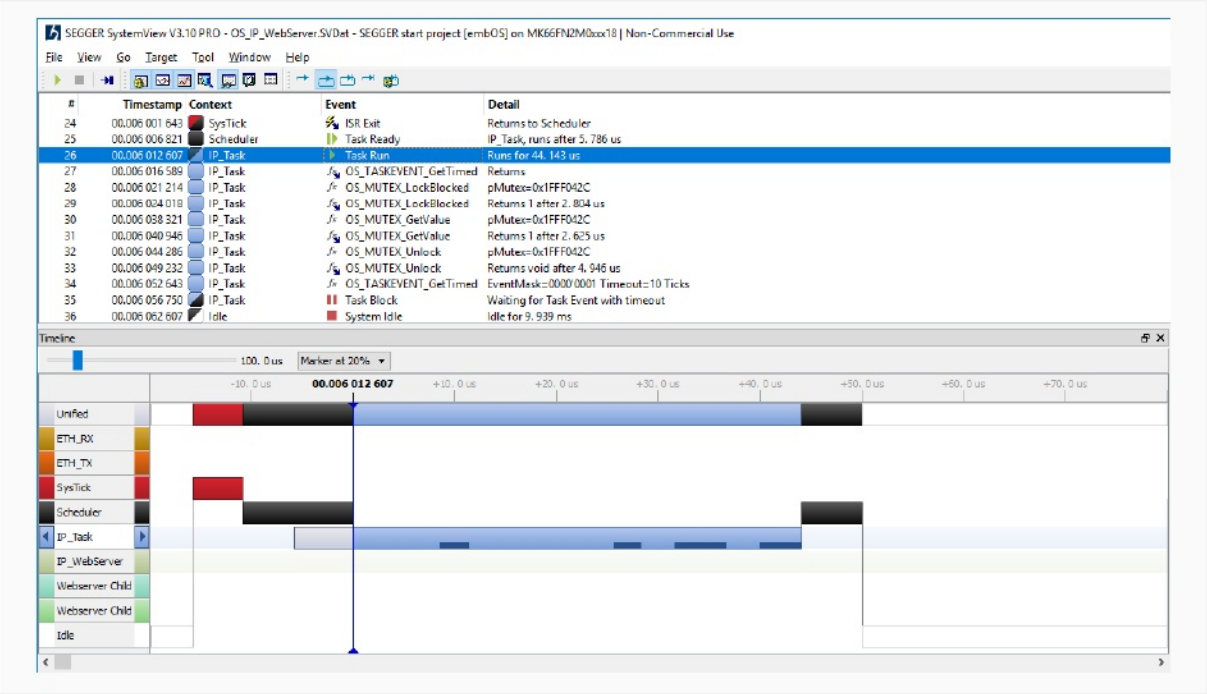
2.2 第一次看系统

我们将先看一下数据，以获得有关被监控系统的一些信息。

系统信息

加载数据后显示的系统信息对话框提供了对记录的第一个概述。它显示有关目标系统的信息，任务、中断和事件的记录和统计信息。系统信息由应用程序报告，因此系统视图不需要任何额外的配置来分析和显示系统行为。

时间轴



系统视图时间轴

*时间轴*窗口显示完整的监控数据。在“*事件*”列表中，滚动到第一个项目开始。

*时间轴*窗口通过*上下文*(任务，中断，调度程序和空闲)在系统时间内可视化系统活动。每行表示一个上下文项，我们可以看到应用程序在被监视时使用的所有项。

一开始，我们可以看到有两个任务，IP_Task和IP_WebServer，由上下文行中的浅色背景表示。

放大到2.0 ms的时间线宽度，双击“+1”下面的垂直线。0 ms’居中并选择项目。(使用鼠标滚轮或[+]/[-]键缩放，或使用菜单或上下文菜单将缩放级别设置为不同的值。)

从SysTick中断开始，每毫秒都有一些系统活动。

将鼠标移到上下文名称上，可以获得有关上下文类型和运行时信息的更多信息。

单击IP_Task上下文的右箭头按钮跳转到它的下一次执行。

放大或缩小以详细显示该活动。

我们可以看到SysTick中断返回到OS调度器，这使得IP_Task准备就绪，由IP_Task行中的灰色条表示，并让它运行。IP_Task从embOS API函数OS_TASKEVENT_GetTimed返回，返回值为0，表示没有及时触发任何事件。

IP_Task调用其他三个embOS API函数，这些函数会快速返回，而OS_TASKEVEN-T_GetTimed函数会激活调度器，停用任务，并将系统置于空闲状态。IP_Task将在事件(EventMask = 1)发生时或超时10个tick(即10.0 ms，因为每1.0 ms发生一个tick)之后再次激活。记录的函数调用在时间轴中以上下文行中的小条形显示。垂直的峰值线表示函数的调用，条形图显示调用的长度。堆叠的条形图显示嵌套的函数调用。

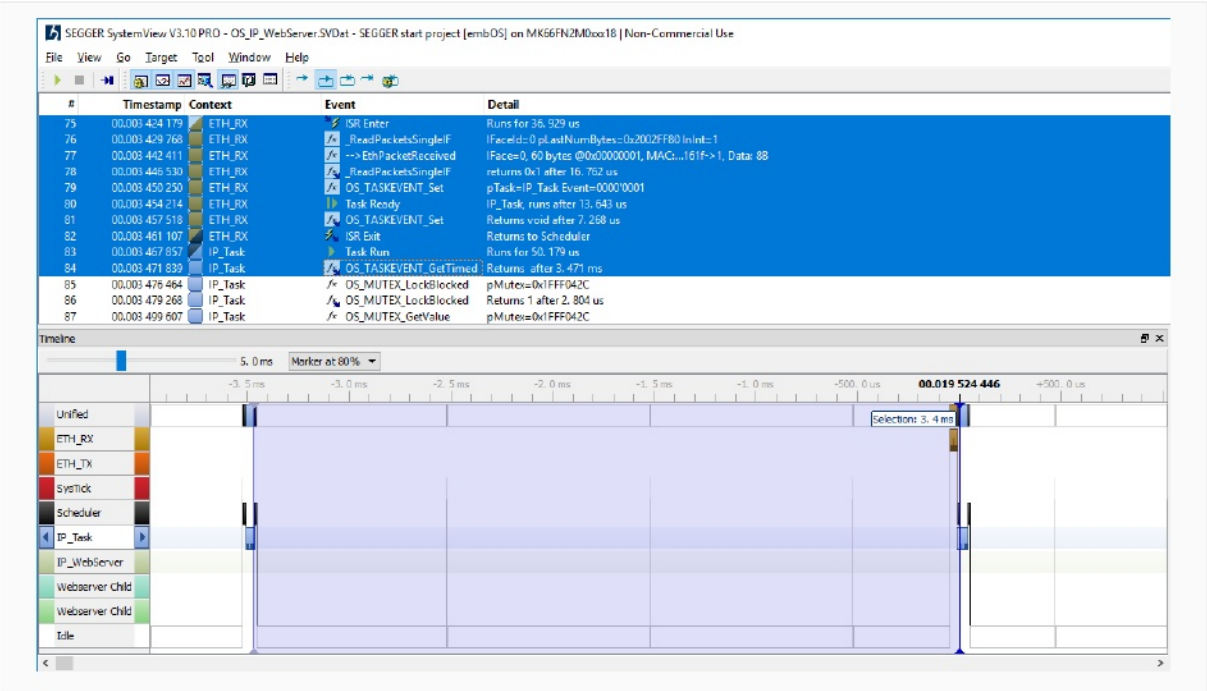
将鼠标移到上下文活动上，可以获得有关上下文运行时、事件和函数调用的更多信息。

结论

我们对监测系统有了初步了解。从时间线我们知道哪些任务和中断被应用程序使用，它是由1 kHz SysTick中断控制的，IP_Task至少每10毫秒激活一次。

2.3 分析系统活动

在获得系统的一些信息之后，我们将分析系统是如何被激活的。



SystemView事件列表和同步时间轴

事件列表

事件列表显示从系统报告的所有事件，并显示它们的信息，包括事件的时间戳、活动上下文、事件类型和事件详细信息。它与时间轴同步。

我们已经看到，SysTick ISR每隔一毫秒进入和退出一次，并且由于超时发生，它每隔10毫秒激活一次IP_Task。

用Go→Goto event…(键盘快捷键:Ctrl+G)切换到事件#66。它是IP_Task在00.016 052 607的OS_TASKEVENT_GetTimed调用，超时时间为10 ms。超时将发生在00.026 052 607。

设置事件的时间参考(查看→切换参考、右键单击→Toggle参考或(键盘快捷键R))。“事件”列表中所有的时间戳都是从最近的参考时间开始计算的。

现在要查看IP_Task运行是因为超时还是因为它等待的事件，请使用go→Forward(键盘快捷键:F)转到IP_Task的下一个活动。

时间戳是00.003 467 857，所以在最后一次引用之后3毫秒，显然在10毫秒超时之前。所以任务已经被它等待的事件激活了。

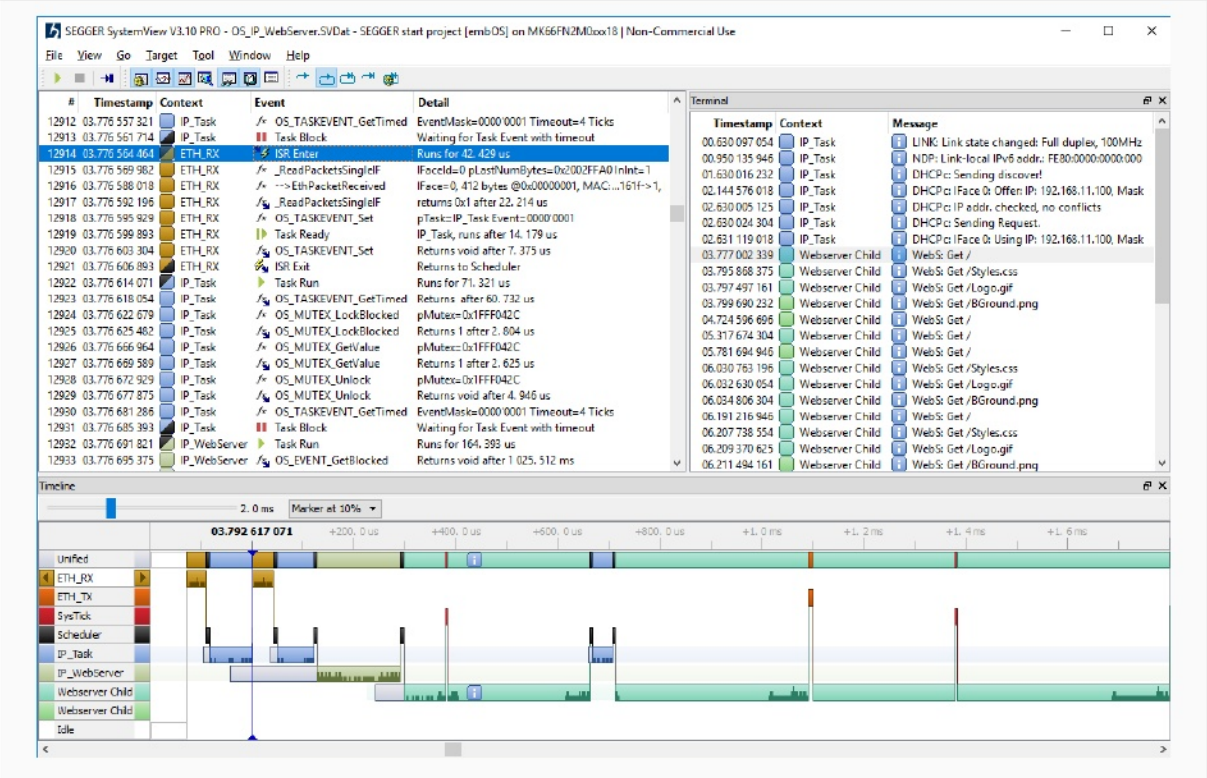
我们可以看到之前发生了ETH_Rx中断。我们通过以太网收到一个数据包(接口0上60字节)，因此ETH_Rx中断发出事件信号，这标志着任务如时间轴中所示准备就绪。ETH_Rx中断返回到调度程序。IP_Task运行，OS_TASKEVENT_GetTimed返回，返回值为0b1，表示该事件已经发生。

结论

进一步查看这些事件，我们可以看到IP_Task在10毫秒超时发生之后被激活，或者在我们接收到一些东西并且ETH_Rx中断发生之后被激活。

2.4 应用程序核心的进一步分析

我们现在知道系统主要由ETH_Rx中断控制。下一步是看系统在更活跃的时候是怎么做的。



SystemView应用分析

时间轴、事件列表、终端和上下文窗口

SystemView的窗口是同步的，当一起使用时，为系统分析提供了最好的可能性。

web服务器应用程序的日志输出也通过SystemView发送，并在终端窗口中显示已记录的时间戳和活动上下文。

在终端中选择一条消息，也可以在事件列表和时间轴中选择它。时间轴也表示所有终端输出。

浏览消息可以看到以太网连接建立时的系统初始化，选择“web: Get /”，这是浏览器请求获取根索引网页的请求。

转到事件#12894，就在消息之前进行详细分析。

这里我们看到发生了一个ETH_Rx中断，它调用embOS/IP函数_Read-PacketsSingleIF并接收数据包。在接收到embOS事件后，像前面看到的那样发出信号，中断退出到激活IP_Task的调度程序中。

IP_Task设置系统事件，该事件向IP_WebServer任务发出准备就绪的信号。另一个数据包被立即接收并由IP_Task处理。

当IP_WebServer开始运行时，它在accept()中调用一些操作系统函数，然后返回。然后检查Webserver子进程是否存在，并创建它(因为它不存在)。
在创建任务时，它被添加到上下文中，并在时间轴上标记为浅色背景，而它不是活动的。

IP_WebServer在accept()中等待另一个连接，Webserver子处理收到的HTTP请求并提供网页。当Webserver子进程处于活动状态时，它可能会被其他ETH_Rx中断中断，这会导致抢占任务切换到IP_Task，因为IP_Task具有比Webserver子进程更高的优先级。

注意:任务在时间轴中按优先级排序，确切的任务优先级可以在上下文窗口中看到。

2.5 分析结论

我们在不深入了解应用程序代码的情况下分析了系统的功能。有了应用程序源，我们可以用SEGGER SystemView检查系统是否做了它应该做的事情。

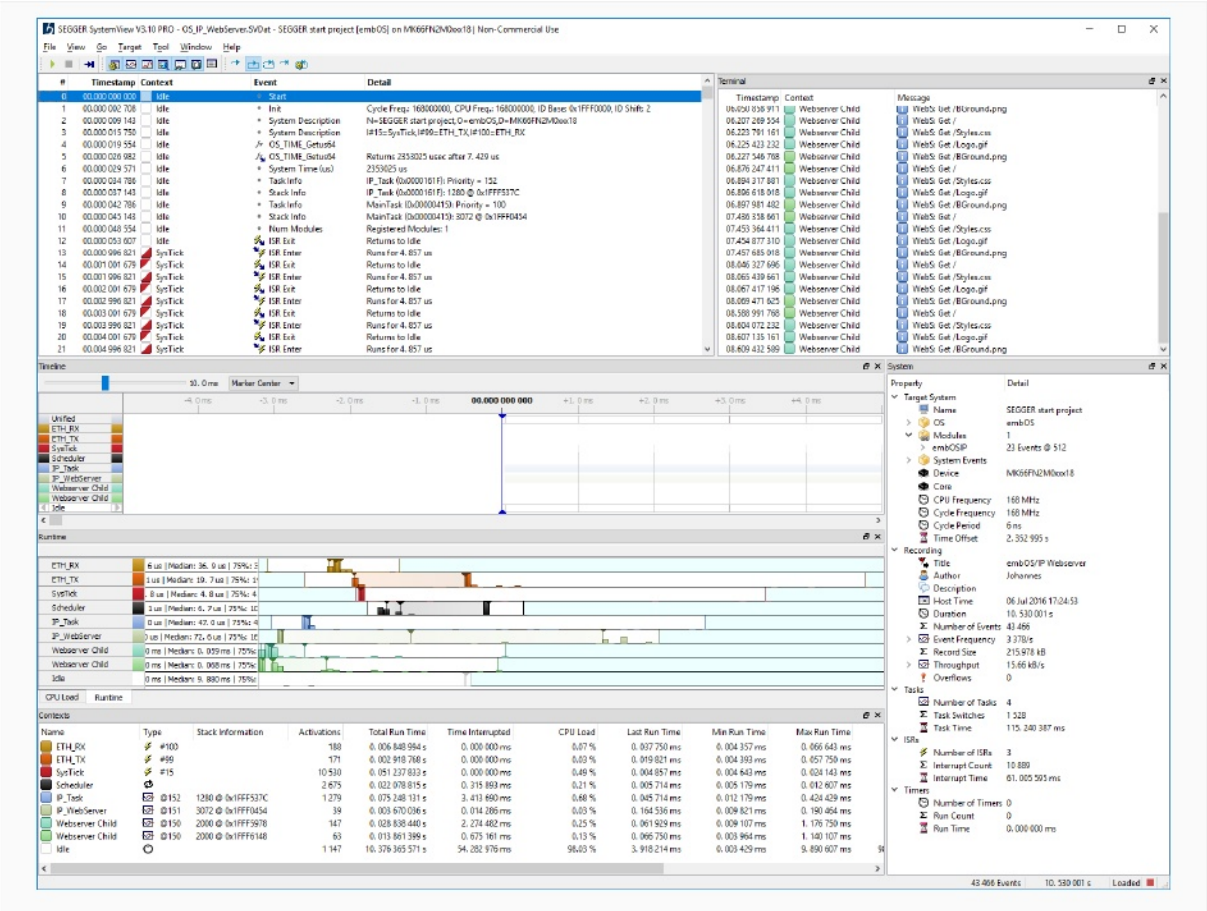
SEGGER SystemView可以积极地帮助开发应用程序，因为它不仅显示系统所做的工作，而且还允许精确的时间测量，并可视化中断和事件对应用程序流的影响。这为发现问题和改进系统提供了先进的可能性。

第三章

SystemView应用程序

介绍SystemView分析和可视化工具。

3.1 介绍



SystemView应用程序

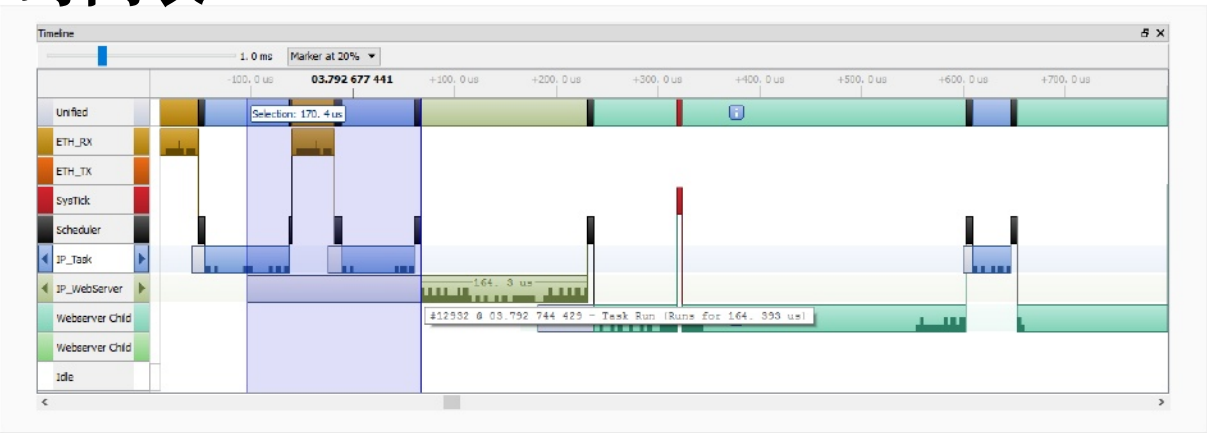
SystemView应用程序是主机PC的SEGGER SystemView可视化工具。它连接到目标应用程序，控制系统事件并读取其数据。mon - 监控数据在运行时进行分析，并在SystemView的不同窗口中进行可视化。记录停止后，数据可以保存到一个文件中，以便以后分析应用程序跟踪。
要开始使用SystemView，请参考上一章。

SystemView提供了不同的窗口来可视化系统中的执行，测量时间和分析CPU负载。所有窗口都是同步的，以便始终获得当前选定状态的所有信息。

有关应用程序窗口的描述，请参考以下部分。

SystemView允许浏览被监视的数据，并随时跟踪系统中发生的事情。

3.2 时间表



SystemView的时间表

*时间轴*窗口在一个视图中收集所有系统信息。它通过*上下文*(任务，中断，调度程序，计时器和空闲)显示系统时间内的系统活动。每一个行引用一个上下文项来显示应用程序在被监视期间使用的所有上下文项。
鼠标悬停在上下文项上的工具提示会显示有关上下文的更多详细信息和运行时信息。

上下文活动上的鼠标悬停工具提示显示当前事件和调用函数(如果可用)的详细信息。

鼠标悬停在上下文活动上时显示的标尺，标记活动的时间跨度。

任务生命周期从创建到终止用浅色背景标记，以提供任何时候存在的任务的快速概述。

上下文之间的切换显示为连接线，以便轻松识别哪些事件导致上下文切换以及何时发生。

标记为“准备执行”的任务在开始执行之前会用浅灰色条显示。

上下文按优先级排序。第一行显示统一上下文中的所有活动。中断在列表的顶部，按Id排序。其次是调度器和软件计时器(如果它们在系统中使用)。在调度器(和计时器)下面，任务按优先级排序。当没有其他上下文处于活动状态时，底部上下文显示空闲时间。

时间轴与事件列表同步。事件标记(蓝线或范围)与“事件”列表中的事件选择匹配。

当事件标记下的上下文处于活动状态时，相应的上下文标签将突出显示。

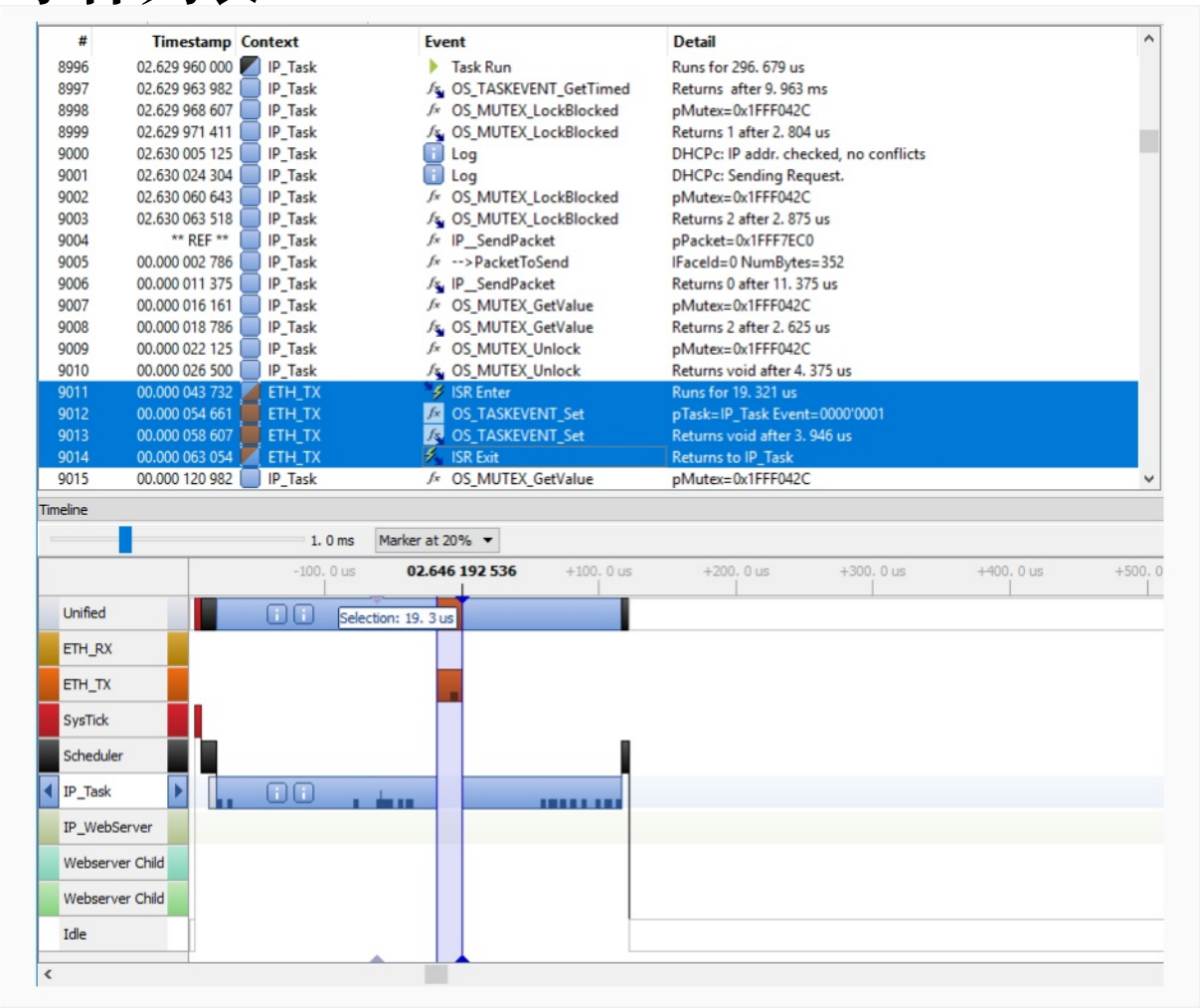
可以将标记固定在窗口的10%到90%，并在滚动时间轴时更新事件列表中的选择。

可以将事件拖放到事件标记下，以在事件列表中选择相应的事件，反之亦然。

要获得整个系统的概览或查看事件的确切持续时间，时间轴视图可以放大或缩小。

要跳转到上下文的下一个或上一个活动，上下文标签包括鼠标悬停时向前和向后导航的按钮，或分别使用快捷键F和B。

3.3 事件列表



系统视图事件列表和时间轴

*事件列表*窗口显示系统上报的所有事件及其相关信息。事件显示内容如下:

- 用于在列表中定位事件的ID。
- 时间戳可选择显示在目标时间或记录时间，分辨率低至纳秒，如果适用。
- 事件报告期间的活动上下文，即正在运行的任务。
- 事件描述，显示事件类型(IRS进入和退出，任务活动，API调用)。
- 描述事件参数的事件细节，例如API调用参数。

事件列表允许浏览列表，跳转到下一个或上一个上下文，或跳转到下一个或上一个类似事件。Timeline和CPU Load窗口同步，以匹配当前选择的事件。

事件列表中的时间戳可以显示为相对于记录开始或目标系统时间，当系统报告时(查看→显示目标时间和显示记录时间)。可以将事件设置为后续事件的时间参考，以便轻松度量一个事件在另一个事件之后发生的时间(快捷键R)。

事件列表有一个事件过滤器，允许显示或隐藏api、isr、系统信息、消息、任务和用户事件。

3.3.1 事件过滤器

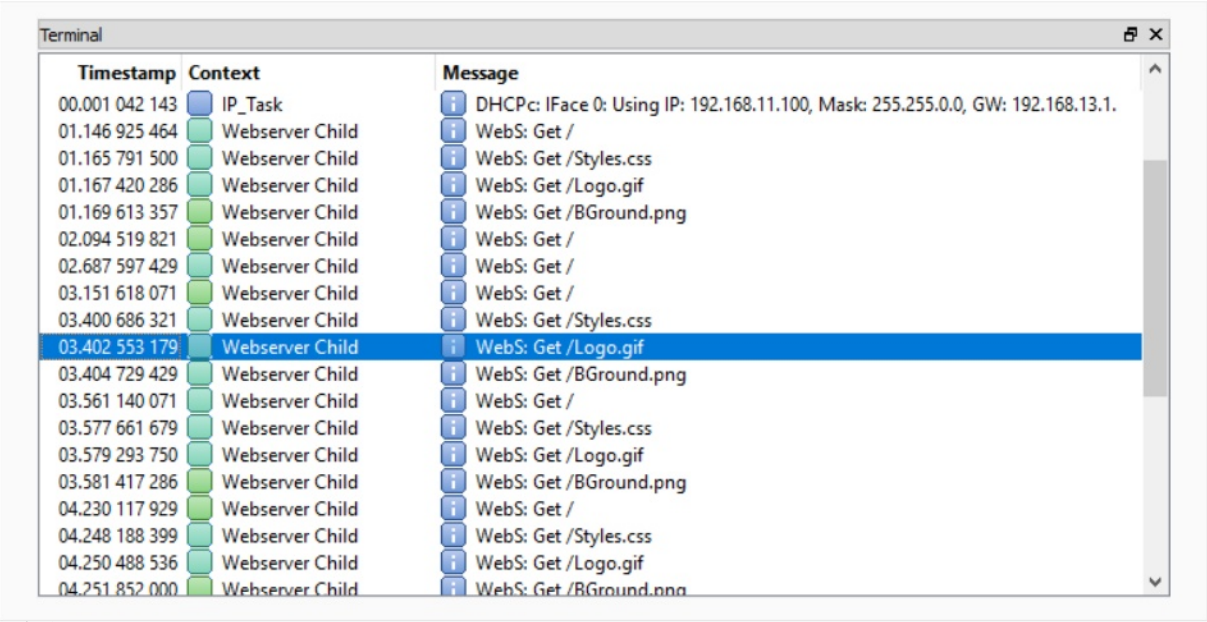
事件列表具有事件过滤功能。这可以用于隐藏中断事件或只显示任务执行。

在系统视图中，事件可以通过不同的组进行过滤:

- api -操作系统或模块生成的事件。
- isr -中断进入和退出。
- 消息-终端输出。
- 系统事件-系统和任务信息。
- 任务-任务执行。
- 标记-性能标记开始，停止和标记。

单个系统事件以及注册的OS或中间件事件的过滤器的设置可以在系统窗口中单独完成(系统在第37页)。

3.4 终端



系统视图终端

终端窗口显示了来自目标应用程序的printf()输出，旁边是发送输出的任务上下文和消息发生时的时间戳

发送。

双击一条消息，将其与事件列表中的所有信息一起显示。

“时间轴”窗口还会显示用于输出的指标。当指示器在显示中重叠时，它们按严重程度排序-错误总是显示在顶部。在时间线中显示的输出指标的最低严重级别可以通过查看→消息指标....进行配置

系统视图打印输出(SEGGER_SYSVIEW_Print*)可以由应用程序格式化发送，也可以不带格式化参数，由系统视图应用程序格式化显示。

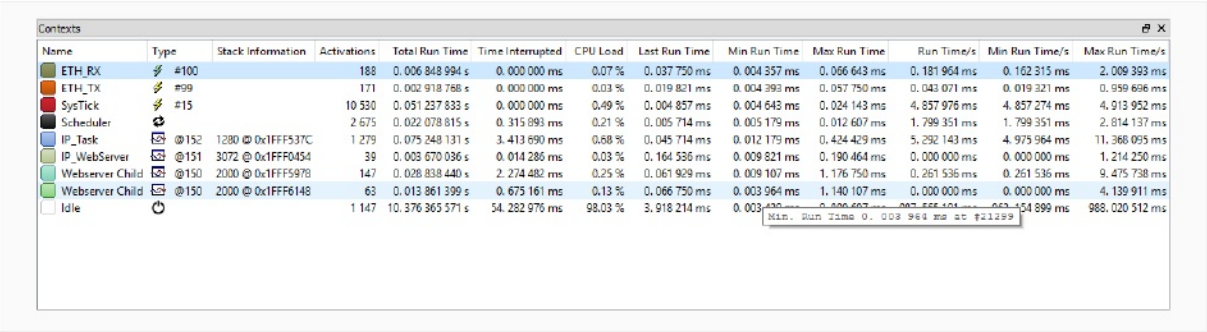
3.5 CPU负载

SystemView CPU负载

“CPU负载”窗口链接到“时间轴”中显示的时间跨度。
在“时间轴”窗口中显示的时间跨度被划分为可配置的箱数，显示在“CPU负载”窗口中。对于每个上下文，其活动时间相对于相应的bin宽度显示。bin中的CPU负载分布按照上下文优先级的顺序显示。

bin的数量可以根据更细或更粗的时间粒度进行调整。当使用单个bin时，CPU负载比率将在整个显示的Timeline部分上计算。

3.6 上下文



SystemView上下文环境

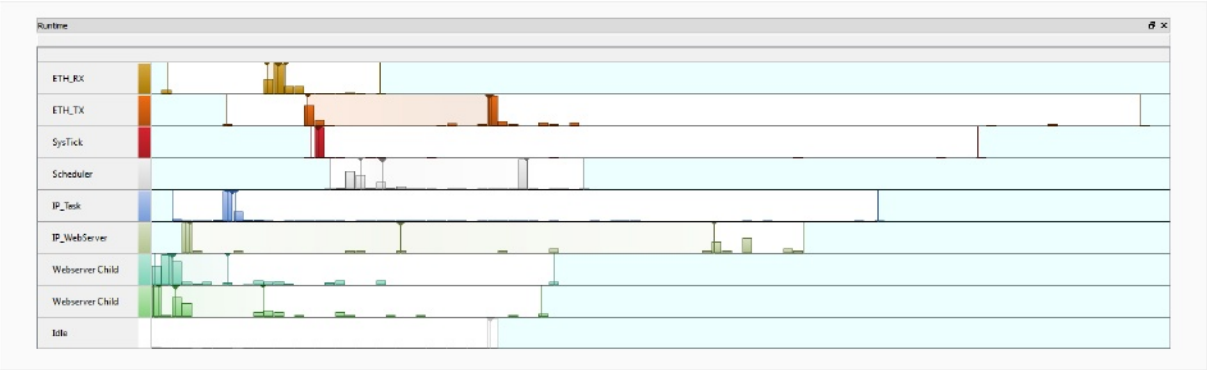
上下文窗口显示每个报告的上下文(任务、中断、调度、定时器和空闲)的统计信息。每个上下文都可以通过其名称和类型来标识。类型包括任务的优先级和中断的ID(例如，Cortex-M系统棒是中断ID #15)。

“上下文”窗口信息包括以下项目:

· 上下文的名称和类型。
· 任务的堆栈信息(如果可用)。
· 上下文的激活数。
· Total, Min和Max Blocked Time, Total, minimum和maximum duration of context ready, but not active, 分别。
· 总运行时间，上下文处于活动状态的总时间。
· 中断时间，上下文被中断挂起的总时间。
· CPU负载，上下文活动时间与完成记录时间的比率。
· 最后、最小和最大运行时间，分别是上下文处于活动状态的最近、最短和最长时间的持续时间。
· 最小和最大运行时间/秒，最小和最大上下文活动时间在最后一秒记录。

在录制过程中，上下文窗口会更新。

3.7 运行时



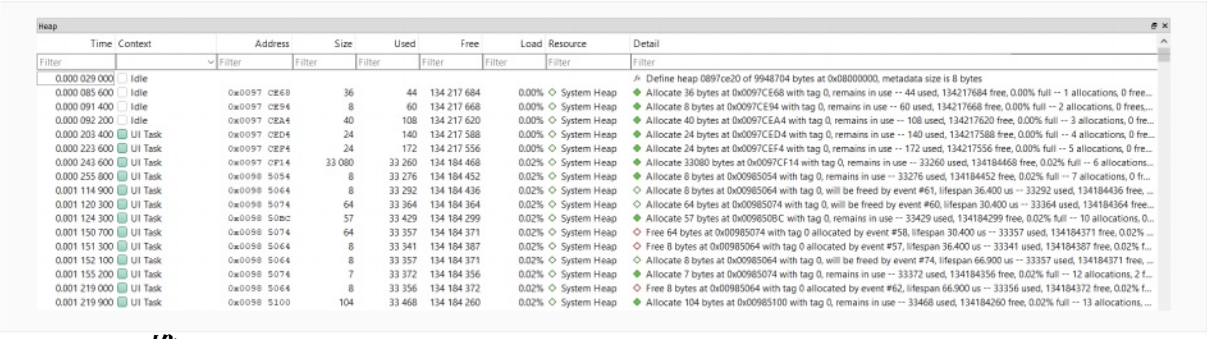
SystemView运行时

运行时窗口显示每个上下文有关其活动时间的统计度量。显示的度量是(在某个上下文的所有调用中):

· 最少的活动时间;
· 四分位数(25%，50%，75%)
· 最大活动时间。

统计测量将根据要求显示为激活时间的箱形图，作为目标报告的1或5 * 10N周期的倍数。N是动态选择的，以便在此之前出现的最大活动时间将适合。持续时间样本的直方图总是由特定上下文的箱形图跨度上的100个箱子组成。

3.8 堆



SystemView堆

Heap窗口记录动态内存的分配和释放，使用SEGGER_RTL_HeapAlloc()， SEGGER_RTL_HeapFree()和其他堆相关的API函数规划设计。

每个分配或释放事件都会更新应用程序维护的一个或多个堆的SystemView模型。SystemView维护这个模型，事件和堆窗口中的每个分配和回收事件显示了这个模型的状态。如果事件溢出导致分配或释放事件丢失，堆的模型就会失效。

3.8.1 堆事件

每个分配或释放事件都记录了在目标上分配的块的地址和大小。除此之外，在event和Heap窗口中显示的事件细节可以显示，对于每个被监控的堆，以每个堆为基础:

- 正在使用的字节数(已分配)
- 空闲字节数(未分配)
- 加载，即用于已分配字节的堆的百分比
- 分配事件的总数
- 分配事件的总数
- 用户为分配的块提供的标签
- 使用的峰值字节数
- 峰值加载
- 没有匹配释放的分配事件的数量
- 分配块在堆中的生命周期，即分配块的时间
- 块的匹配分配或释放事件

事件的彩色编码可以快速识别潜在的内存泄漏:

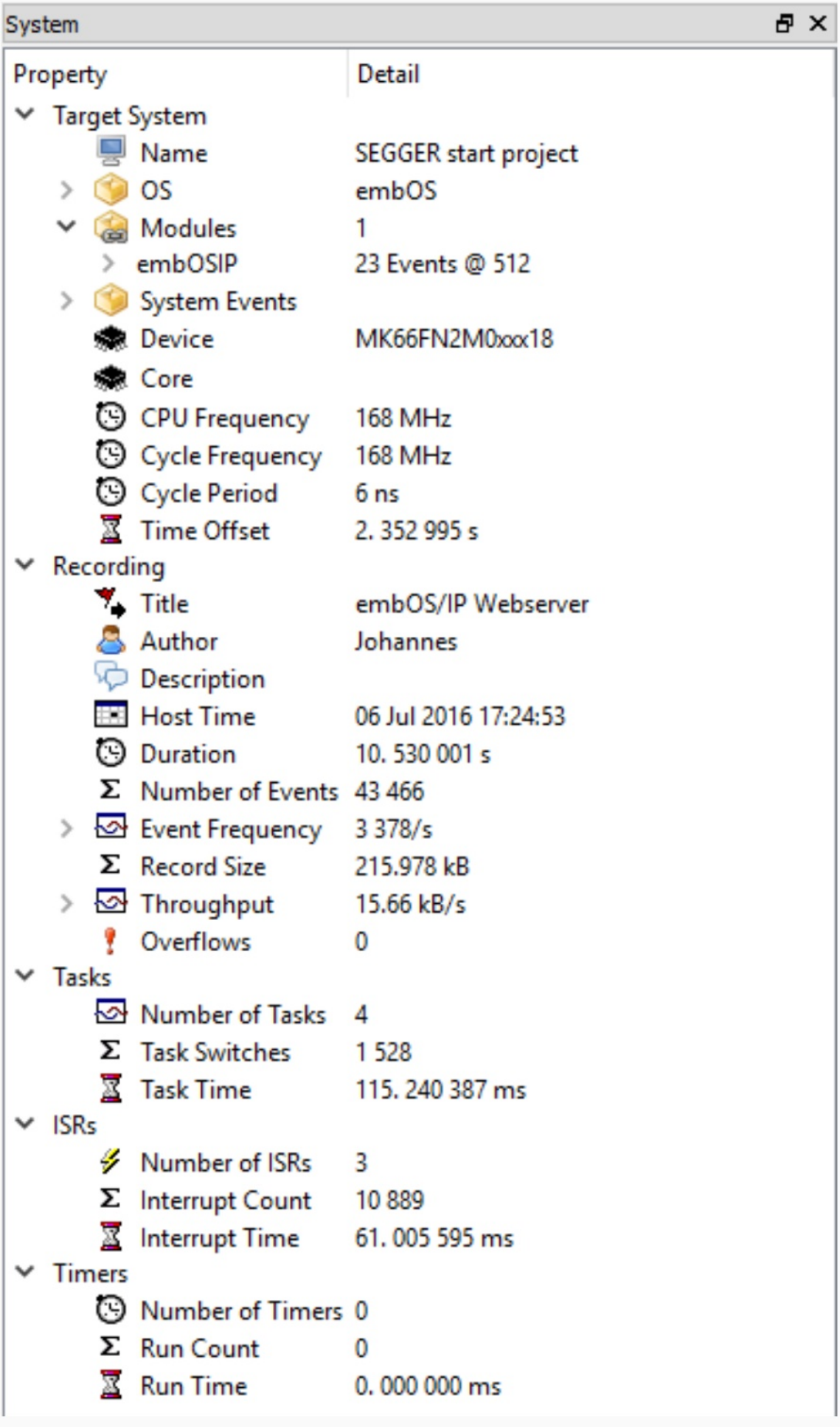
- 分配事件与匹配的脱位显示为绿色空心菱形
- 具有匹配分配的释放事件显示为红色空心菱形
- 没有匹配释放的分配事件显示为实绿色菱形

3.8.2 API函数

下面是监控堆的API函数:

- *SEGGER_SYSVIEW_HeapDefine*在185页
- 186页的*SEGGER_SYSVIEW_HeapAlloc*
- *SEGGER_SYSVIEW_HeapAllocEx*在187页
- *SEGGER_SYSVIEW_HeapFree*在188页

3.9 系统



SystemView系统

系统窗口显示:

- 目标系统，关于系统的信息，已由应用程序报告以识别它。这一部分还包含用户可设置的属性，用于配置操作系统、模块和系统事件的显示。
- 记录信息，如事件数，平均和峰值事件频率以及额外的用户提供的关于记录的元信息。
- 分析信息，关于记录分析阶段的统计信息。
- 统计任务，中断，定时器和其他SystemView事件。

目标系统信息包括应用程序名称、正在运行的操作系统、目标硬件信息和定时信息。关于任务切换和中断频率的附加信息提供了对系统的快速概述。

用户可设置的属性和元信息与记录一起保存，并允许对记录进行识别和预先设置配置，以供以后分析。

3.10 触发模式

在事件的实时连续记录和分析过程中，触发模式允许自动选择和集中显示符合可配置标准的事件。
触发模式可以在工具栏中进行选择。
在手动滚动模式下，选择不会自动更新，用户可以在完成记录的同时滚动事件并分析系统。
在自动滚动模式下，选择每100毫秒同步一次。选择最后一个倍数为100毫秒的事件。
在连续触发模式下，用户可以配置在哪个事件和哪个上下文(taks、中断或标记)发生触发。然后，SystemView总是选择最后一次出现的符合配置条件的事件。

在单触发模式下，SystemView在下一次接收到满足配置条件的事件时触发一次，并切换回手动滚动模式。

3.11 GUI控件

SystemView可以控制与鼠标和键盘，通过菜单和上下文菜单。最重要的控件也可以在工具栏中访问。

下表描述了SystemView的控件。

Action	Menu	Shortcut
Recording control		
Start recording on the target.	Target →Start Recording	F5
Stop recording.	Target →Stop Recording	Shift+F5
Read post-mortem or single-shot data from the system.	Target →Read Recorded Data	Ctrl+F5
Configuring the recorder interface to target	Target →Recorder Configuration	Alt+Enter
Configuring trigger modes	Target →Trigger →...	
Data handling		
Save recorded data to a file.	File →Save Data	Ctrl+S
Load a record file.	File →Load Data	Ctrl+O
Load a recently used file.	File →Recent Files	none
Load a sample recording.	File →Sample Recordings	none
Export recorded data as file with CSV (comma separated values).	File →Export Data	Ctrl+E
View, Timeline		
Set/clear the current event as time reference.	View →Toggle Reference	R
Remove all time references.	View →Clear References	Ctrl+Shift+R
Display timestamps as absolute target time.	View →Display Target Time	None
Display timestamps relative to start of recording.	View →Display Recording Time	None
Zoom in.	*when Timeline is focused*	+, scroll up
Zoom out.	*when Timeline is focused*	−, scroll down
Quick set defined Timeline width (in us, ms or s, in 1-2-5 steps …)	View →Zoom →View 10us, …, View 1ms, …, View 1s, …, View 100s	None
Set the marker to 0% of the timeline.	View →Marker →Marker Left	0
Set the marker to *x*% of the timeline. (x=10% .. 90%, 10% steps)	View →Marker →Marker at 10% … →Cursor at 90%	1 … 9
Set the marker to 100% of the timeline.	View →Marker →Marker Right	None
Show all output indicators in Timeline	View →Message Indicators →Show All Messages	None
Show output indicators with severity error and warning in Timeline	View →Message Indicators →Show Errors and Warnings	None
Show output indicators with severity error in Timeline	View →Message Indicators →Show Errors only	None
View, Events list		

Action	Menu	Shortcut
Show/Hide API calls in the Events list.	View →Event Filter →Show APIs	Shift+A
Show/Hide ISR Enter/Exit in the Events list.	View →Event Filter →Show ISRs	Shift+I
Show/Hide Messages in the Events list.	View →Event Filter →Show Messages	Shift+M
Show/Hide System events in Events list.	View →Event Filter →Show System Events	Shift+S
Show/Hide Task activity in Events list.	View →Event Filter →Show Tasks	Shift+T
Show/Hide output indicators in Events list.	View →Event Filter →Show Markers	Shift+E
Show only API calls in the Events list.	View →Event Filter →Show APIs only	Ctrl+Shift+A
Show only ISR Enter/Exit in the Events list.	View →Event Filter →Show ISRs only	Ctrl+Shift+I
Show only Messages in the Events list.	View →Event Filter →Show Messages only	Ctrl+Shift+M
Show only System events in Events list.	View →Event Filter →Show System Events only	Ctrl+Shift+S
Show only Task activity in Events list.	View →Event Filter →Show Tasks only	Ctrl+Shift+T
Show only output indicators in Events list.	View →Event Filter →Show Markers only	Ctrl+Shift+E
Hide registered API invocation (and corrsponding exit) event. (Only available on selected invocation event)	View →Event Filter →Hide This Event	Shift+Ctrl+H
Reset all event filters.	View →Event Filter →Reset all Filters	Ctrl+Shift +Space
Trigger control		
Manually Scroll mode while recording.	Target →Trigger →Manual Scroll	
Automatic Scroll mode while recording.	Target →Trigger →Auto Scroll	
Continuously trigger on a condition and focus triggering event while recording.	Target →Trigger →Continuous Trigger	
Trigger on a condition once and focus on triggering event while recording.	Target →Trigger →Trigger Once	
Configure the trigger condition.	Target →Trigger →configure Trigger	
View, Runtime		
Show/Hide statistic measures in Runtime window	View →Show Runtime Statistics	None
Show/Hide boxplots for statistic measures in Runtime window	View →Show Runtime Boxplot	None
Show/Hide histograms in boxplots in Runtime window	View →Show Runtime Histogram	None

Action	Menu	Shortcut
Navigation, Timeline		
Jump to the next context switch.	Go →Forward	F
Jump to the previous context switch.	Go →Back	B
Jump to the next similar event.	Go →Next [Event]	N
Jump to the previous similar event.	Go →Previous [Event]	P
Jump to the next similar event with the same context.	Go →Next [Event] in [Context]	Shift+N
Jump to the previous similar event with the same context.	Go →Previous [Event] in [Context]	Shift+P
Open dialog to go to an event by Id.	Go →Go to Event…	Ctrl+G
Open dialog to go to an event by timestamp.	Go →Go to Timestamp…	Ctrl+Shift+G
Scroll forward.	*when Timeline is focused*	Left, Ctrl +Scroll up, Click&Drag
Scroll back.	*when Timeline is focused*	Right, Ctrl +Scroll down, Click&Drag
Window		
Show/hide the System information window.	Window →System	None
Show/hide the Timeline window.	Window →Timeline	None
Show/hide the CPU Load window.	Window →CPU Load	None
Show/hide the Runtime window.	Window →Runtime	None
Show/hide the Contexts window.	Window →Context View	None
Show/hide the Terminal window.	Window →Terminal View	None
Show/hide the Log window.	Window →Log View	None
Show/Hide the Status bar	Window →Show Status Bar	None
Show/Hide the Tool bar	Window →Show Tool Bar	None
Miscellaneous		
Open application preferences dialog.	Tool →Preferences	Alt+.
Open License manager dialog.	Tool →Licence Manager	Alt+L
Help		
Open this SystemView Manual.	Help →User Guide	F11
Show SystemView information.	Help →About	F12

3.12 命令行选项

SystemView可以通过命令行选项进行控制和配置。要跳过记录开始时的配置对话框，可以通过命令行选项给出目标配置。

在配置选项之后，可以在命令行上给出0、1或多个选项来控制SystemView并自动执行部分操作。

如果以单实例模式启动，SystemView的第一个实例将正常启动并解析其命令行。任何后续实例将其命令行控制选项传递给已经运行的实例。

或者，可以通过在TCP/ IP套接字上向localhost:19050发送命令来控制正在运行的实例。

SystemView.exe <Filename>
在SystemView启动时加载选定的记录文件。
(Used for drag and drop on SystemView executable.)

```
C:> SystemView.exe [-recorder J-Link|UART|IP] [-device <Device>] [-usb [<SN>]]|[-
ip <Host>] [-if SWD|JTAG|FINE] [-speed <Speed>] [-rttcbaddr <Addr>]|[-rttcbrange
 auto|<Range>] [-start|-stop|-quit|-save [<Filename>]|-load[<Filename>]|-export
 [<Filename>]|-export-contexts [<Filename>]|-export-terminal [<Filename>]]*

Command Line Options:
-recorder   Select the recorder interface.              Parameter: J-Link,
 UART, IP.

-device     Set the target device.                      Parameter: Device name
 as supported by J-Link.

-usb        Connect to J-Link via USB.                  Parameter: S/N of J-
Link. (Optional)
-ip         Connect to J-Link via IP.                   Parameter: IP or S/N of
 J-Link.

-if         Set the target interface.                   Parameter: SWD, JTAG,
 or FINE.
-speed      Set the target interface speed.             Parameter: Speed in
 kHz.
-jtagconf   Set the JTAG scan chain configuration.      Parameter: IRPre and
 DRPRe of the target device.

-rttcbaddr  Set the RTT Control Block address.          Parameter: Address in
 hexadecimal.
-rttcbrange Set the search range for RTT Control Block.  Parameter: auto or
 ranges as "<Address> <Size>".

-single     Start SystemView in single instance mode.
-port       Set local port for single instance mode.    Parameter: Port.

Command Line Control:
-start              Start recording.
-stop               Stop recording.
-quit               Quit SystemView.
-load               Load a recording from file.                 Parameter: File
 to load.
-save               Save current recording.                     Parameter: File
 to save to.
-export             Export the current recording to a file.     Parameter: File
 to export to.
-export-contexts    Export the Contexts window to csv.          Parameter: File
 to export to.
-export-terminal    Export the Terminal window to csv.          Parameter: File
```

导出到。

延迟 在…	Delay before executing the next command.	Parameter: Time

3.13 SystemView录音

本节描述如何使用SystemView应用程序进行连续记录，以及如何使用调试器进行手动单镜头记录。

3.13.1 连续记录

SystemView可以在目标运行时连续实时记录目标的执行情况。

可以使用J-Link调试探针从外部和非侵入性地进行连续记录，该探针通过调试接口读取记录的事件，或者由目标应用程序控制，通过网络连接或串行线发送数据。

开始记录

要开始连续录制，请连接目标器和所选记录器接口。
选择“目标→开始录音”。在第一次启动SystemView时，打开记录器配置。保存该配置，供后续录音使用。若要切换到其他记录器或更改配置，请选择“目标→记录器配置”。

配置记录仪后，SystemView连接并开始录音。

停止记录

要停止录制，选择“目标→停止录制”。

3.13.1.1 J-Link记录仪

要使用J-Link记录器，需要配置与J-Link的连接、与目标的连接以及RTT控制块的位置。

选择通过USB或IP连接到J-Link，并可选择输入序列号或IP以选择特定的J-Link。

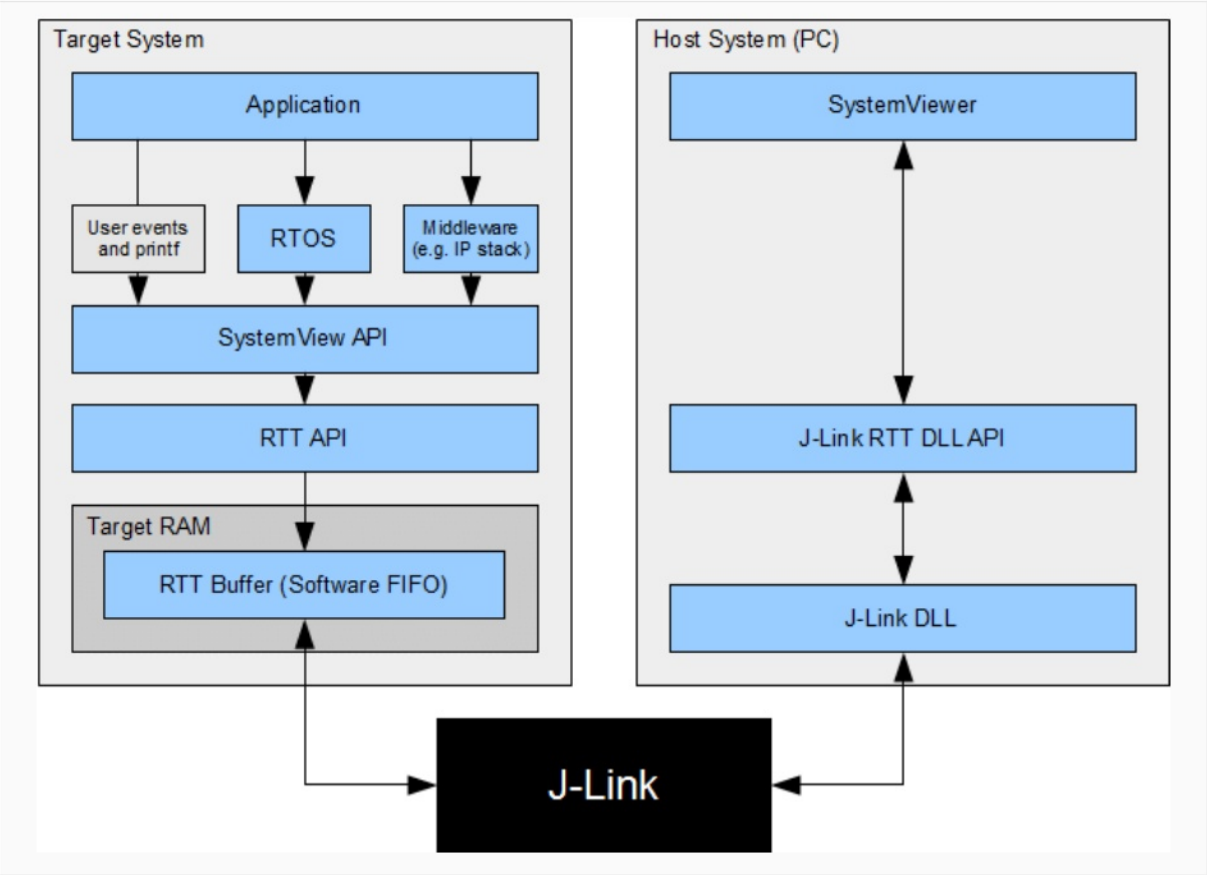
输入或选择设备名称。如果当前设备不在列表中，则可以手动输入或从设备选择对话框中选择。

> **请注意**
> 对于RTT控制块自动检测，以及正确连接到设备，必须知道确切的设备。建议不要选择通用核心代替。

请选择所连接设备的目标接口和目标接口速度。
配置RTT控制块检测。在大多数情况下，可以使用自动检测。如果无法检测到RTT控制块，则从应用程序或其映射文件中获取_SEGGER_RTT的地址并输入它，或者输入符号可能位于的搜索范围，格式为<StartAddress> <Size>，例如0x10000000 0x10000。

> **请注意**
> SystemView可以与调试器并行使用。在这种情况下，可以在调试器运行时进行记录。确保所有需要的配置都在调试器中完成。当调试器停止时，SystemView记录也将停止。

通过J-Link调试探头和SEGGER实时传输技术(RTT)， Sys-temView可以在目标运行时连续实时记录目标的执行情况。RTT需要在程序执行期间通过调试接口读取内存的能力。这尤其包括ARM Cortex-M0, M0+， M1, M3, M4和M7处理器以及所有瑞萨RX设备。



Systemview如何与J-Link一起工作

3.13.1.2 IP记录器

SystemView IP记录器连接到目标设备上运行的对应设备。
在目标上，“IP记录器主机”正在运行并接受来自Sys-temView应用程序的连接，以便将其数据发送到该应用程序。
选择目标设备的IP地址和端口(默认:19111)。

3.13.1.3 UART记录仪

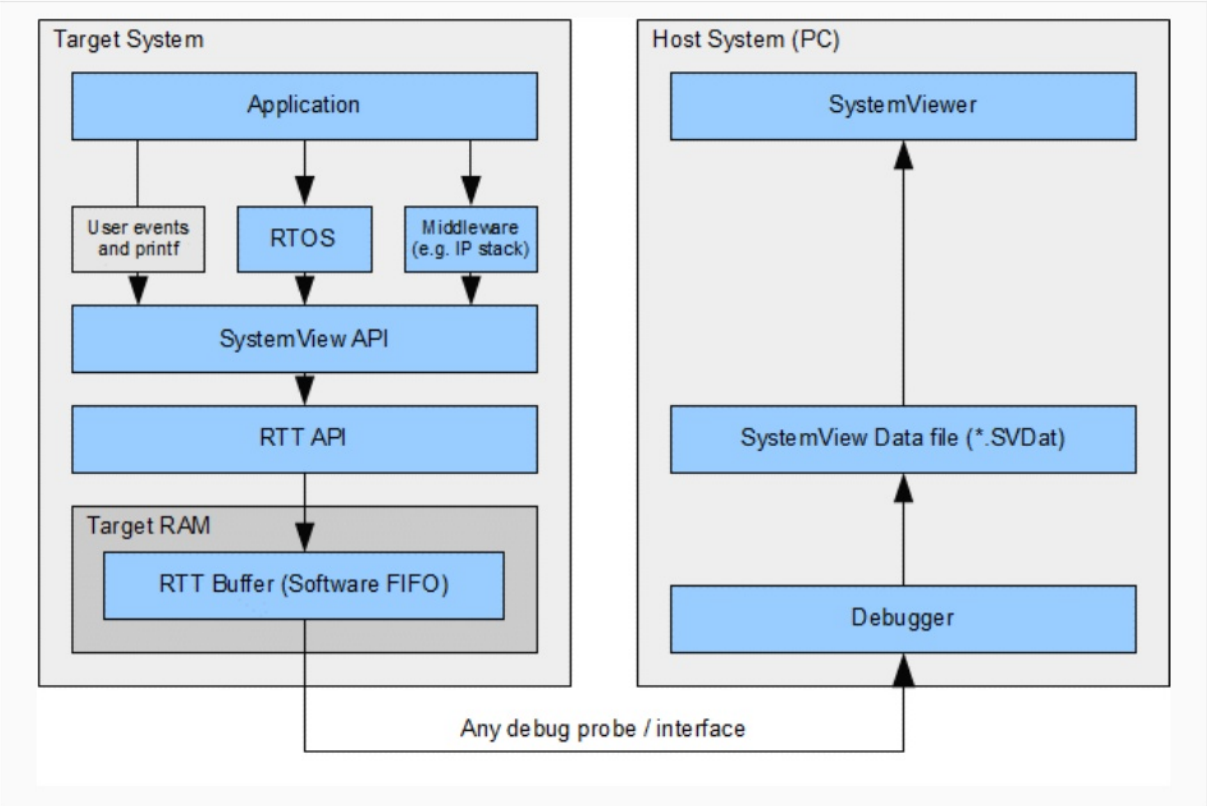
UART记录仪通过串行线连接到目标，即RS232上的UART。在现代计算机上，通常使用USB到RS232转换器。
在目标上，需要将UART配置为接收命令并将其存储在SystemView缓冲区中，以及在SystemView缓冲区可用时从该缓冲区发送数据。

选择目标器连接的COM端口和通过UART通信的波特率。

3.13.2 单发记录

当目标设备不支持RTT或没有使用J-Link时，SEGGER Sys-temView可以用来记录数据，直到其目标缓冲区被填满。
在单发模式下，记录是在应用程序中手动开始的，这允许只记录特定的部分，这是感兴趣的。

由于通常的应用程序每秒生成大约5到15 kByte的记录数据，并且只有在满载时才能达到更高的峰值速率，因此即使内部RAM中的一个小缓冲区也可以用于记录数据以分析关键部件。当使用外部RAM时，即使在单镜头模式下，SystemView也可以长时间记录。



Systemview如何在单镜头模式下工作

从系统中获取单发数据

要获得在单镜头模式下记录的数据，必须通过SystemView应用程序或外部调试器读取SystemView缓冲区。

· 连接调试器并加载目标应用程序。
· 从应用程序(SEGGER_SYSVIEW_Conf()或SEGGER_SYSVIEW_Init)配置和初始化SystemView。
· 在应用程序中从应该分析的地方开始记录(SEGGER_SYSVIEW_Start)。

使用J-Link SystemView可以自动从目标读取单次射击数据。;;

· 启动SystemView应用程序，选择“目标→读取记录数据”。;如果没有J-Link或SystemView，可以使用以下步骤读取数据:
· 当缓冲区已满或记录完成后，在调试器中停止应用程序。
· 获取SystemView RTT缓冲区地址和使用的字节数(通常为_SEGGER_RTT.aUp[1])。_SEGGER_RTT.aUp[1]. wroff)。
· 将缓冲区中的字节数保存到扩展名为. svdat的文件中。
· 使用SystemView应用程序打开该文件。

为了能够记录多次，可以在读取数据时将缓冲区写偏移量(_SEGGER_RTT.aUp[1]. wroff)设置为0。为了防止SystemView溢出事件发生，应用程序应该在缓冲区被填满后立即停止，并且不能再容纳另一个SystemView事件。

3.13.3 事后分析

事后分析类似于单次记录，但有一点不同:SystemView事件是连续记录的，当缓冲区被填满时，SystemView缓冲区会包裹起来覆盖旧的事件。当读取缓冲区时，最新的事件可用。

当系统长时间运行并突然崩溃时，事后分析可能很有用。在这种情况下，SystemView缓冲区可以从目标中读取，SystemView可以显示崩溃之前系统中发生的事情。

> **请注意**
> 事后分析要求调试器或调试探针能够连接到目标系统，而无需重新设置或修改RAM。

为了获得尽可能多的有用数据进行分析，建议为SystemView使用大的缓冲区，8 kByte或更多。外部RAM可以用于SystemView缓冲区。
要将目标系统配置为验尸模式，请参考第66页的*SEGGER_SYSVIEW_POST_MORTEM_MODE*和第67页的*SEGGER_SYSVIEW_SYNC_PERIOD_SHIFT*。

从系统中获取验尸数据

要获得在事后分析模式下记录的数据，必须通过SystemView应用程序或外部调试器读取SystemView缓冲区。

- 配置和初始化SystemView从应用程序(SEGGER_SYSVIEW_Conf()或SEGGER_SYSVIEW_Init)。
- 在应用程序中从应该分析的地方开始记录(SEGGER_SYSVIEW_Start)。
- 连接调试器，加载目标应用程序，并让系统运行。

使用J-Link SystemView可以自动从目标读取死后数据。

- 启动SystemView，选择“Target→Read Recorded Data”。

如果没有J-Link或没有SystemView，可以使用以下步骤读取数据:

由于SystemView缓冲区是一个环形缓冲区，因此数据可能必须分两个块读取，以便从开始读取并保存尽可能多的数据。

- 配置和初始化SystemView从应用程序(SEGGER_SYSVIEW_Conf()或SEGGER_SYSVIEW_Init)。
- 在应用程序中从应该分析的地方开始记录(SEGGER_SYSVIEW_Start)。
- 连接调试器，加载目标应用程序，并让系统运行。
- 当系统崩溃或所有测试完成时，将调试器附加到系统并停止它。
- 获取SystemView RTT缓冲区(通常为_SEGGER_RTT.aUp[1]. pbuffer)。
- 保存数据从pBuffer + WrOff直到缓冲区结束到一个文件。
- 将pBuffer中的数据附加到文件中，直到pBuffer + RdOff - 1。
- 保存文件为*。SVdat或*.bin。
- 使用SystemView应用程序打开该文件。

3.13.4 保存并加载录音

停止录音时，可以将录制的数据保存到文件中，以备以后分析和记录。选择“文件→保存数据”。弹出录制属性对话框，允许将标题、作者和描述与数据文件一起保存。单击OK。选择保存数据的位置，单击“保存”。
保存的数据可以通过File→Load data打开。最近使用的数据文件也可以通过“文件→最近文件”的菜单获得。SystemView可以打开*.bin和*.bin。SV-Dat文件。

3.13.5 出口记录

为了在外部工具中进一步分析，可以将记录的事件导出为csv文件。选择“文件→导出数据”。每个事件将被导出到csv文件中，如事件窗口所示。

此外，上下文窗口和终端窗口的内容可以导出为csv文件。从窗口的上下文菜单中选择Export.…

第四章

在目标上开始使用SystemView

介绍将SystemView模块添加到目标系统的操作步骤。

4.1 在应用程序中包含SystemView

以下文件是SEGGER SystemView目标实现的一部分。我们建议将所有文件复制到应用程序项目中，并保持给定的目录结构。

File	Description
/Config/Global.h	Global type definitions for SEGGER code.
/Config/SEGGER_RTT_Conf.h	SEGGER Real Time Transfer (RTT) configuration file.
/Config/SEGGER_SYSVIEW_Conf.h	SEGGER SYSTEMVIEW configuration file.
/Config/SEGGER_SYSVIEW_Config_[SYSTEM].c	Initialization of SystemView for [SYSTEM].
/Sample/OS/SEGGER_SYSVIEW_[OS].c	Interface between SYSTEMVIEW and [OS].
/Sample/OS/SEGGER_SYSVIEW_[OS].h	Interface header.
/SEGGER/SEGGER.h	Global header for SEGGER global types and general purpose utility functions.
/SEGGER/SEGGER_RTT.c	SEGGER RTT module source.
/SEGGER/SEGGER_RTT.h	SEGGER RTT module header.
/SEGGER/SEGGER_SYSVIEW.c	SEGGER SYSTEMVIEW module source.
/SEGGER/SEGGER_SYSVIEW.h	SEGGER SYSTEMVIEW module header.
/SEGGER/SEGGER_SYSVIEW_ConfDefaults.h	SEGGER SYSTEMVIEW configuration fallback.
/SEGGER/SEGGER_SYSVIEW_Int.h	SEGGER SYSTEMVIEW internal header.

4.1.1 通用文件

通用文件SEGGER_SYSVIEW和SEGGER_RTT位于/SEGGER/中。它们需要添加到任何项目中，文件夹应该设置为include目录。

4.1.2 通用配置

SYSVIEW和RTT的通用配置文件位于/配置目录下。该文件夹需要设置为包含目录。

可以修改SEGGER_SYSVIEW_Conf.h和SEGGER_RTT_Conf.h以匹配目标系统。

4.1.3 特定于操作系统和特定于目标的文件

SystemView目标源包括与已经测试过的不同rtos的集成，以及针对不同目标系统的配置。

目标系统的匹配文件需要添加到项目中。

例子

对于在Cortex-M3上具有embOS的系统，包括/Sample/embOS/Config/Cortex-M/SEGGER_SYSVIEW_Config_embOS.c，/Sample/embOS/SEGGER_SYSVIEW_embOS.c 和 /Sample/embOS/SEGGER_SYSVIEW_embOS.h。
对于没有OS或在Cortex-M3上没有仪表化OS的系统，只包含/Sample/NoOS/ 配置/Cortex-M/SEGGER_SYSVIEW_Config_NoOS.c。

4.1.4 记录文件

当SystemView事件不是通过J-Link记录，而是通过IP连接或串行线记录时，也需要将记录器源添加到项目中。

SystemView目标源包括在/Sample/COMM/中使用embOS和emNet的示例记录器。

4.2 初始化SystemView

系统信息由应用程序发送。这些信息可以通过SEGGER_SYSVIEW_Config_[SYSTEM].c中的定义来配置。在main函数中添加对SEGGER_SYSVIEW_Conf()的调用来初始化SystemView。

```
#include "SEGGER_SYSVIEW.h"

/*********************************************************************
*
*       main()
*
* Function description
*   Application entry point
*/
int main(void) {
  OS_IncDI();                      /* Initially disable interrupts         */
  OS_InitKern();                   /* Initialize OS                        */
  OS_InitHW();                     /* Initialize Hardware for OS           */
  BSP_Init();                      /* Initialize BSP module                */

  SEGGER_SYSVIEW_Conf();           /* Configure and initialize SystemView  */

  /* You need to create at least one task before calling OS_Start() */
  OS_CREATETASK(&TCB0, "MainTask", MainTask, 100, Stack0);
  OS_Start();                      /* Start multitasking                   */
  return 0;
}
```

SEGGER SystemView的通用部分现在已经准备好监视应用程序了。
当使用embOS V4.12或更高版本并启用了分析时，将生成isr、Task和API调用的SystemView事件。
当不使用embOS时，应用程序必须生成适当的事件。

将应用程序下载到目标并让其运行。只要SystemView应用程序没有连接，并且没有调用SEGGER_SYSVIEW_Start()，应用程序就不会生成SystemView事件。当SystemView连接或SEGGER_SYSVIEW_Start()被调用时，它将激活记录SystemView事件。

4.3 发送系统信息

The included files `SEGGER_SYSVIEW_Config_[SYSTEM].c` provide the system information to the SystemView Application and can in most cases be used without modification.

```
/*********************************************************************
*                    (c) SEGGER Microcontroller GmbH                 *
*                        The Embedded Experts                        *
*                           www.segger.com                           *
**********************************************************************

---------------------------- END-OF-HEADER -----------------------------

File    : SEGGER_SYSVIEW_Config_embOS.c
Purpose : Sample setup configuration of SystemView with embOS.
Revision: $Rev: 25330 $
*/
#include "RTOS.h"
#include "SEGGER_SYSVIEW.h"
#include "SEGGER_SYSVIEW_embOS.h"

/*********************************************************************
*
*       Defines, fixed
*
**********************************************************************
*/
#define DEMCR         (*(volatile U32*) (0xE000EDFCuL))
  // Debug Exception and Monitor Control Register
#define TRACEENA_BIT  (1uL << 24)                       // Trace enable bit
#define DWT_CTRL      (*(volatile U32*) (0xE0001000uL))  // DWT Control Register
#define NOCYCCNT_BIT  (1uL << 25)
  // Cycle counter support bit
#define CYCCNTENA_BIT (1uL << 0)
  // Cycle counter enable bit
//
// If events will be recorded without a debug probe (J-Link) attached,
// enable the cycle counter
//
#define ENABLE_DWT_CYCCNT (SEGGER_SYSVIEW_POST_MORTEM_MODE || SEGGER_SYSVIEW_USE_INTERNAL_RECORD

/*********************************************************************
*
*       Local functions
*
**********************************************************************
*/
/*********************************************************************
*
*       _cbSendSystemDesc()
*
*  Function description
*    Sends SystemView description strings.
*/
static void _cbSendSystemDesc(void) {
  SEGGER_SYSVIEW_SendSysDesc("N=" SEGGER_SYSVIEW_APP_NAME ",O=embOS,D=" SEGGER_SYSVIEW_DEVICE_NA
#ifdef SEGGER_SYSVIEW_SYSDESC0
  SEGGER_SYSVIEW_SendSysDesc(SEGGER_SYSVIEW_SYSDESC0);
#endif
#ifdef SEGGER_SYSVIEW_SYSDESC1
  SEGGER_SYSVIEW_SendSysDesc(SEGGER_SYSVIEW_SYSDESC1);
#endif
#ifdef SEGGER_SYSVIEW_SYSDESC2
  SEGGER_SYSVIEW_SendSysDesc(SEGGER_SYSVIEW_SYSDESC2);
#endif
}
```



```
/*********************************************************************
*
*       Global functions
*
**********************************************************************
*/
/*********************************************************************
*
*       SEGGER_SYSVIEW_Conf()
*
* Function description
*   Configure and initialize SystemView and register it with embOS.
*
* Additional information
*   If enabled, SEGGER_SYSVIEW_Conf() will also immediately start
*   recording events with SystemView.
*/
void SEGGER_SYSVIEW_Conf(void) {
#if ENABLE_DWT_CYCCNT
  //
  // If no debugger is connected, the DWT must be enabled by the application
  //
  if ((DEMCR & TRACEENA_BIT) == 0) {
    DEMCR |= TRACEENA_BIT;
  }
#endif
  //
  //  The cycle counter must be activated in order
  //  to use time related functions.
  //
  if ((DWT_CTRL & NOCYCCNT_BIT) == 0) {       // Cycle counter supported?
    if ((DWT_CTRL & CYCCNTENA_BIT) == 0) {    // Cycle counter not enabled?
      DWT_CTRL |= CYCCNTENA_BIT;              // Enable Cycle counter
    }
  }
  SEGGER_SYSVIEW_Init(SEGGER_SYSVIEW_TIMESTAMP_FREQ, SEGGER_SYSVIEW_CPU_FREQ,
                      &SYSVIEW_X_OS_TraceAPI, _cbSendSystemDesc);
  OS_SetTraceAPI(&embOS_TraceAPI_SYSVIEW);   // Configure embOS to use SYSVIEW.
#if SEGGER_SYSVIEW_START_ON_INIT
  SEGGER_SYSVIEW_Start();
  // Start recording to catch system initialization.
#endif
}

/************************* 文件结束 **************************/
```


4.4 开始和停止录音

当SystemView连续读取数据时，记录由SystemView应用程序自动启动和停止。当SystemView不记录目标系统时，将不会生成SystemView事件，从而最小化系统开销。

对于单镜头记录，必须在应用程序中调用SEGGER_SYSVIEW_Start()来激活记录SystemView事件。事件被记录，直到SystemView缓冲区被填满或SEGGER_SYSVIEW_Stop()被调用。

对于事后分析，必须在应用程序中调用SEGGER_SYSVIEW_Start()来激活记录SystemView事件。事件被记录，直到SEGGER_SYSVIEW_Stop*(被调用。当SystemView缓冲区被填满时，旧的事件将被覆盖。

4.5 编译时配置

SEGGER系统视图是可配置的，以匹配目标设备和应用。默认的编译时配置标志是用有效值预先配置的，以匹配大多数系统的要求，通常不需要修改。
系统视图的默认配置可以通过编译时标志来改变，这些标志可以添加到SEGGER_SYSVIEW_Conf.h中。

4.5.1 系统配置

下面的编译时配置需要与目标系统匹配。SEGGER_SYSVIEW_Conf.h中的示例配置定义了匹配大多数系统的配置(例如带有Embedded Studio、GCC、IAR或Keil ARM的Cortex-M设备)。如果样例配置不包括使用的系统，则必须相应地调整配置。

有关系统特定配置的详细描述，请参见第82页的“*支持*的cpu”。

4.5.1.1 SEGGER_SYSVIEW_GET_TIMESTAMP ()

函数宏用于检索系统视图事件的系统时间戳。

在Cortex-M3/4/7设备上，Cortex-M周期计数器可以用作系统时间戳。

默认Cortex-M3/4/7: (*(U32 *)(0xE0001004))

在大多数其他设备上，系统时间戳必须由计时器生成。在去故障配置中，系统时间戳通过用户提供的函数SEG-GER_SYSVIEW_X_GetTimestamp()来检索。

*其他内核的默认值:*SEGGER_SYSVIEW_X_GetTimestamp()

例 如 :Sample/NoOS/Config/ Cortex-M0/ segger_sysview_config - fig_embos_cmm .c 或 Sample/NoOS/Config/RX/SEGGER_SYSVIEW_Config_NoOS_RX.c

> **请注意**
> 系统时间戳的频率必须在SEG-GER_SYSVIEW_Init()中提供。

4.5.1.2 SEGGER_SYSVIEW_TIMESTAMP_BITS

时钟源作为系统时间戳发送的有效低阶比特数。
如果使用未修改的时钟源作为系统时间戳，则有效位的数量是时钟源的位宽(例如32位或16位)。

*默认值:*32位(使用32位时钟源)

例如节省带宽

由于SystemView数据包使用变长编码，移动时间戳可以节省缓冲区空间和带宽。

32位时钟源，例如，在Cortex-M4上的Cortex-M周期计数器可以移动4，导致有效的时间戳比特数为28，时间戳频率，如在SEGGER_SYSVIEW_Init中使用的，是核心时钟频率除以16。

```
#define SEGGER_SYSVIEW_GET_TIMESTAMP() ((*(U32 *)(0xE0001004)) >> 4)

#define SEGGER_SYSVIEW_TIMESTAMP_BITS 28;
```


4.5.1.3 SEGGER_SYSVIEW_GET_INTERRUPT_ID ()

获取当前活动中断的函数宏。

在Cortex-M设备上，可以从ICSR读取有源矢量。

默认的Cortex-M3/4/7: ((*(U32*)(0xE000ED04)) & 0x1FF)

默认设置:(((U32*)(0xE000ED04)) & 0x3F)

在其他设备上，活动中断可以直接从中断控制器中检索，可以保存在通用中断处理程序中的变量中，或者必须在每个中断例程中手动分配。

默认情况下，这可以通过用户提供的函数SEGGER_SYSVIEW_X_GetInter-ruptId()或替换宏定义来完成。

举个例子，参考93页的Sample/embOS/Config/RX/SEGGER_SYSVIEW_Config_embOS_RX.c或*Cortex-A/R Interrupt ID*。

4.5.1.4 SEGGER_SYSVIEW_LOCK ()

函数宏递归地锁定系统视图传输以免被中断。即禁用中断。

SEGGER_SYSVIEW_LOCK()必须保留之前的锁状态，以便在SEG-GER_SYSVIEW_UNLOCK()中恢复。
记录一个系统视图事件不能因为记录另一个事件而中断。因此，所有被系统视图记录的中断(调用segger_sysview_recorden_terisr / SEGGER_SYSVIEW_RecordExitISR)，调用仪器化函数(例如OS API函数)，导致立即上下文切换，或可能创建任何其他系统视图事件必须被禁用。

SEGGER_SYSVIEW_LOCK()可以使用与SEGGER_RTT_LOCK()相同的锁定机制。

*默认值:*SEGGER_RTT_LOCK()

SEGGER_RTT_LOCK()是为大多数系统(例如带有嵌入式Studio、GCC、IAR或Keil ARM的Cortex-M设备，以及带有IAR的RX设备)在SEGGER_RTT_Conf.h中定义的。如果宏没有定义，或者为空，则必须提供它以匹配目标系统。

4.5.1.5 segger_sysview_unlock ()

函数宏递归解锁被中断的系统视图传输。即恢复先前的中断状态。

SEGGER_SYSVIEW_UNLOCK()可以使用与SEGGER_RTT_UN-LOCK()相同的锁定机制。

*默认值:*SEGGER_RTT_UNLOCK()

SEGGER_RTT_UNLOCK()是为大多数系统(例如带有Embedded Studio、GCC、IAR或Keil ARM的Cortex-M设备，以及带有IAR的RX设备)在SEGGER_RTT_Conf.h中定义的。如果宏没有定义，或者为空，则必须提供它以匹配目标系统。

4.5.2 通用配置

以下编译时标志可用于调优或更改系统视图事件的记录方式。

默认编译时配置标志是用有效值预先配置的，以匹配大多数系统的要求，通常不需要修改。

4.5.2.1 SEGGER_SYSVIEW_RTT_BUFFER_SIZE

SystemView用于记录缓冲区的字节数。
对于连续记录，在大多数情况下1024字节的缓冲区就足够了。根据目标接口速度、目标速度和系统负载的不同，缓冲区的大小可能会增加到最高4096字节。

对于单次记录，缓冲区的大小决定了可以记录的事件的数量。根据负载的不同，系统可能产生10到200 kByte/s的数据。建议使用至少8 kByte的缓冲区，最多可占用整个可用RAM空间。缓冲区也可以在外部RAM中。

对于事后分析，缓冲区大小决定了可用于分析的事件的最大数量。一个系统可能产生10到200 kByte/秒，这取决于它的负载。建议使用至少8 kByte的缓冲区，最多可占用整个可用RAM空间。缓冲区也可以在外部RAM中。

*默认值:*1024字节

4.5.2.2 SEGGER_SYSVIEW_RTT_CHANNEL

RTT通道，用于SystemView事件的记录和通信。0:自动选择

> **Note**
>
> SEGGER_RTT_MAX_NUM_UP_BUFFERS, defined in SEGGER_RTT_Conf. h has to be greater than SEGGER_SYSVIEW_RTT_CHANNEL.

*默认值:*0

4.5.2.3 SEGGER_SYSVIEW_USE_STATIC_BUFFER

如果设置为1，SystemView使用静态缓冲区来创建SystemView事件。这通常会节省空间，因为只需要一个缓冲区，任务栈可以尽可能小。当使用静态缓冲区时，在SystemView锁定调用之间执行的关键代码需要稍长的时间。

如果设置为0，则在堆栈上创建SystemView事件。确保所有的任务栈，以及用于中断的C栈都足够大，以容纳最大的SystemView事件(~228字节)。SystemView仅在将堆栈缓冲区传输到RTT缓冲区时锁定。
*默认值:*1

4.5.2.4 SEGGER_SYSVIEW_POST_MORTEM_MODE

如果设置为1，则启用尸检分析模式。
在事后分析模式下，SystemView使用循环缓冲区并保留所有事件，直到最后记录，而不是在缓冲区已满时删除事件。

> **请注意**
> 当附加的J-Link主动读取RTT数据时，不要使用事后分析模式。

*默认值:*0

4.5.2.5 SEGGER_SYSVIEW_SYNC_PERIOD_SHIFT

配置Sync和System Info事件以事后模式发送的频率。确保SystemView缓冲区中至少有一个同步可用。

建议同步频率为Buffer Size / 16

默认值:8 =每256个包同步一次

4.5.2.6 SEGGER_SYSVIEW_ID_BASE

从SystemView报文中记录的id中减去的值。
id是TaskIds、timerid和resourceid，它们通常是指向RAM结构的指针。在OS和中间件API事件中发送的参数也可以由工具编码为id。

> **请注意**
> 如果被测OS不使用TaskIds、timerid或resourceid的指针，则SEGGER_SYSVIEW_ID_BASE必须设置为0。

由于SystemView数据包使用可变长度的指针编码，正确的重基地址可以节省缓冲区空间和带宽。

定义为系统中使用的最低RAM地址。

可以在初始化时通过SEGGER_SYSVIEW_SetRAMBase()被应用程序覆盖。如果有疑问，则将SEGGER_SYSVIEW_ID_BASE定义为0。

*默认值:*0 x10000000

4.5.2.7 SEGGER_SYSVIEW_ID_SHIFT

SystemView报文中记录的id的移位位数。
id是TaskIds、timerid和resourceid，它们通常是指向RAM结构的指针。在OS和中间件API事件中发送的参数也可以由工具编码为id。

> **请注意**
> 如果被测OS不使用TaskIds、timerid或resourceid的指针，则SEGGER_SYSVIEW_ID_SHIFT必须设置为0。

由于SystemView数据包使用可变长度的指针编码，正确移动地址地址可以节省缓冲区空间和带宽。

对于大多数32位处理器上的应用程序，SystemView事件中记录的所有id实际上都是指针，并且是4的倍数，因此可以安全地忽略最低的2位。

如果有疑问，则将SEGGER_SYSVIEW_ID_SHIFT定义为0。

*默认值:*2

4.5.2.8 SEGGER_SYSVIEW_MAX_STRING_LEN

SystemView事件记录的最大字符串长度。

字符串用于SystemView printf风格的用户函数，以及SEGGER_SYSVIEW_SendSysDesc()和SEGGER_SYSVIEW_RecordModuleDescription。确保SEGGER_SYSVIEW_MAX_STRING_LEN与这些函数中使用的字符串长度匹配。

*默认值:*128

4.5.2.9 SEGGER_SYSVIEW_MAX_ARGUMENTS

SEGGER_SYSVIEW_PrintfHost, SEGGER_SYSVIEW_PrintfHostEx, SEGGER_SYSVIEW_WarnfHost和SEGGER_SYSVIEW_Er-rorfHost发送的最大参数数。

如果在应用程序中不使用这些函数，则可以将SEGGER_SYSVIEW_MAX_ARGUMENTS设置为0,以最小化静态缓冲区大小。
*默认值:*16

4.5.2.10 segger_sysview_buffer_section

SystemView RTT缓冲区可以放在一个专用的部分，而不是默认的数据部分。这允许将缓冲区放入外部存储器或给定地址。
当定义segger_sysview_buffer_section时，必须在链接器脚本中定义该节。

*默认值:*SEGGER_RTT_SECTION或未定义

Embedded Studio中的示例

```
//
// SEGGER_SYSVIEW_Conf.h
//
#define SEGGER_SYSVIEW_BUFFER_SECTION "SYSTEMVIEW_RAM"

//
// flash_placement.xml
//
<MemorySegment name="ExtRAM">
  <ProgramSection load="No" name="SYSTEMVIEW_RAM" start="0x40000000" />
</MemorySegment>
```

4.5.3 RTT配置

以下编译时标志可用于调优或更改RTT。
默认编译时配置标志是用有效值预配置的，以匹配大多数系统的要求，通常不需要修改。

4.5.3.1 BUFFER_SIZE_UP

用于RTT终端输出通道的字节数。
RTT可用于printf终端输出，无需修改。BUFFER_SIZE_UP定义了可以为此缓冲多少字节。

如果不使用RTT终端输出，则将BUFFER_SIZE_UP定义为最小值4。

*默认值:*1024 Bytes

4.5.3.2 BUFFER_SIZE_DOWN

用于RTT终端输入通道的字节数。

RTT可以在终端输入通道上接收来自主机的输入。BUFFER_SIZE_DOWN定义了可以缓冲多少字节，从而一次从主机发送多少字节。

如果不使用RTT终端输入，则将BUFFER_SIZE_DOWN定义为最小值4。

*默认值:*16字节

4.5.3.3 segger_rtt_max_num_up_buffers

RTT up(到主机)缓冲区的最大数目。缓冲区0总是用于RTT终端输出，因此要与SystemView SEGGER_RTT_MAX_NUM_UP_BUFFERS一起使用它必须至少为2。

*默认值:*2

4.5.3.4 segger_rtt_max_num_down_buffers

RTT down (to target)缓冲区的最大数目。缓冲区0总是用于RTT终端输入，因此要与SystemView SEGGER_RTT_MAX_NUM_UP_BUFFERS一起使用它必须至少为2。

*默认值:*2

4.5.3.5 segger_rtt_mode_default

预初始化RTT终端通道模式(缓冲区0)。*默认值:*SEGGER_RTT_MODE_NO_BLOCK_SKIP

4.5.3.6 segger_rtt_printf_buffer_size

用于RTT打印以通过RTT批量发送字符的缓冲区大小。如果不使用SEGGER_RT-T_Printf, 可以定义为0。

*默认值:*64

4.5.3.7 SEGGER_RTT_SECTION

RTT控制块可以放在一个专用的部分，而不是默认的数据部分。这允许将其放置在一个已知的地址，以便能够使用J-Link自动检测或轻松指定搜索范围。

当定义了SEGGER_RTT_SECTION后，应用程序必须通过在启动代码中用0初始化它或在应用程序开始时显式调用SEGGER_RT-T_Init()来确保该节有效。SEGGER_RTT_Init()由SEG-GER_SYSVIEW_Init()隐式调用。

*默认值:*未定义

4.5.3.8 segger_rtt_buffer_section

RTT终端缓冲区可以放在一个专用的部分，而不是默认的数据部分。这允许将缓冲区放入外部存储器或给定地址。
*默认值:*SEGGER_RTT_SECTION或未定义

4.5.4 优化SystemView

为了从目标系统获得最精确的运行时信息，记录仪器代码必须快速、最少干扰、小而高效。SystemView代码的编写是为了高效和最少的干扰。SystemView的速度和大小取决于目标和编译器的配置。下面几节介绍如何优化SystemView。

4.5.4.1 编译器优化

即使在调试构建中，也应该始终打开SystemView目标实现的编译器优化，以生成快速的记录例程，从而减少开销并减少干扰。

支持速度或大小优化的配置依赖于编译器。在某些情况下，平衡配置可能比只考虑速度的配置更快。

4.5.4.2 记录优化

SystemView使用可变长度编码来存储和传输事件，这可以节省调试接口上的缓冲区空间和带宽。

一些事件参数的大小可以通过编译时配置进行优化。

收缩id

id是指向RAM中符号的指针，例如，任务ID是指向任务控制块的指针。为了尽量减少记录的id的长度，可以缩小它们。
从指针中减去SEGGER_SYSVIEW_ID_BASE以获得其ID。它可以设置为从指针中减去基本RAM地址，这仍然会产生唯一的，但更小的id。例如，如果RAM范围是0x20000000到0x20001000，建议将SEGGER_SYSVIEW_ID_BASE定义为0x20000000，这将导致指针0x20000100具有ID 0x100，并且需要两个而不是四个位来存储它。
SEGGER_SYSVIEW_ID_SHIFT是指针向右移动以获得其ID的位数。如果所有记录的指针都是4字节对齐，SEGGER_SYSVIEW_ID_SHIFT可以定义为2。指针0x20000100的ID将是0x8000040，或者用前面的SEGGER_SYSVIEW_ID_BASE减去0x20000000，ID将是0x40，只需要记录一个字节。

时间戳的来源

SystemView中的事件时间戳记录为与前一个事件的时间戳之差。这本身就节省了缓冲空间。

虽然建议使用具有CPU时钟频率的时间戳源以获得最高的时间分辨率，但较低的时间戳频率可能会节省额外的缓冲区空间，因为时间戳增量较低。

在CPU时钟频率为160 MHz的情况下，时间戳可能会移动4，从而导致时间戳频率为10 MHz (100 ns分辨率)，并且要编码的时间戳减少4位。

当时间戳大小不再是32位时，即它在0xFFFFFFFF之前包装，SEGGER_SYSVIEW_TIMESTAMP_BITS必须被定义为时间戳大小，例如，当将32位时间戳移动4时为28。

4.5.4.3 缓冲区配置

SystemView和RTT的记录和通信缓冲区大小可以在目标配置中设置。

对于连续记录，在大多数情况下，1到4 kByte的小缓冲区是足够的，并且允许使用SystemView，即使内部RAM很小。

对于单镜头和后期模式，更大的缓冲区可能是可取的。在这种情况下，SEG-GER_SYSVIEW_RTT_BUFFER_SIZE可以设置为更大的值。为了将SystemView记录缓冲区放入外部RAM，可以定义一个SEGGER_SYSVIEW_BUFFER_SECTION，并相应地调整链接器脚本。

如果只使用SystemView并且没有RTT终端输出，可以将SEGGER_RT-T_Conf.h中的BUFFER_SIZE_UP设置为较小的值以节省内存。

4.6 支持的cpu

介绍如何针对不同的目标cpu设置和配置SystemView模块。

SEGGER SystemView实际上支持任何目标CPU，但是，只有支持后台内存访问的cpu (ARM Cortex-M和Renesas RX)才能实现连续记录。在其他cpu上，SystemView可以在单镜头或事后分析模式下使用。参考第47页的*单镜头记录*。

为了使SystemView正常运行，必须进行一些特定于目标的配置。下面将介绍一些cpu的这种配置。

4.6.1 Cortex-M3 / Cortex-M4

Recording mode	Supported?
Continuous recording	Yes
Single-shot recording	Yes
Post-mortem analysis	Yes

4.6.1.1 事件的时间戳

Cortex-M3 / Cortex-M4上的时间戳源可以是周期计数器，它允许周期精确的事件记录。

为了在记录事件时节省带宽，周期计数器可以选择性地右移，例如右移4位，这将导致核速度除以16的时间戳频率。

配置:

```
//
// Use full cycle counter for higher precision
//
#define SEGGER_SYSVIEW_GET_TIMESTAMP()   (*(U32 *)(0xE0001004))
#define SEGGER_SYSVIEW_TIMESTAMP_BITS    (32)
//
// Use cycle counter divided by 16 for smaller size / bandwidth
//
#define SEGGER_SYSVIEW_GET_TIMESTAMP()   ((*(U32 *)(0xE0001004)) >> 4)
#define SEGGER_SYSVIEW_TIMESTAMP_BITS    (28)
```

4.6.1.2 中断ID

当前活动的中断可以通过读取Cortex-M ICSR[8:0]直接识别，它是中断控制器状态寄存器(ICSR)中的活动向量场。

配置:

```
//
// Get the interrupt Id by reading the Cortex-M ICSR[8:0]
//
#define SEGGER_SYSVIEW_GET_INTERRUPT_ID()    ((*(U32 *)(0xE000ED04)) & 0x1FF)
```

4.6.1.3 系统视图锁定和解锁

锁定和解锁系统视图以防止传输记录被中断可以通过禁用中断来完成。在Cortex-M3 / Cortex-M4上，不是所有的中断都需要禁用，只需要禁用那些本身可能产生系统视图事件或导致操作系统任务切换的中断。

默认情况下，优先级掩码设置为32，禁用所有优先级为32或更低(更高的数值)的中断。

确保屏蔽所有可以发送RTT数据的中断，即生成系统视图事件或导致任务切换。当发送RTT数据时不能屏蔽高优先级中断时，必须相应地调整SEGGER_RTT_MAX_INTERRUPT_PRIORITY。(高优先级=低优先级号)

embOS的默认值:128u

FreeRTOS 中的默认配置:configMAX_SYSCALL_INTERRUPT_PRIORITY:(configLIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY<< (8 - configPRIO_BITS))

如有疑问，禁用所有中断。

系统视图和RTT的Lock和unlock可以是相同的。

配置:

```
//
// RTT locking for GCC toolchains in SEGGER_RTT_Conf.h
//
#define SEGGER_RTT_LOCK()      {                                                \
                                   unsigned int LockState;                      \
                                 __asm volatile ("mrs   %0, basepri  \n\t"      \
                                                 "mov   r1, $32      \n\t"      \
                                                 "msr   basepri, r1  \n\t"      \
                                                 : "=r" (LockState)             \
                                                 :                              \
                                                 : "r1"                         \
                                                 );                             \
                                                                                \
#define SEGGER_RTT_UNLOCK()    __asm volatile ("msr   basepri, %0  \n\t"        \
                                                 :                              \
                                                 : "r" (LockState)              \
                                                 :                              \
                                                 );                             \
                               }
//
// Define SystemView locking in SEGGER_SYSVIEW_Conf.h
//
#define SEGGER_SYSVIEW_LOCK()   SEGGER_RTT_LOCK()
#define SEGGER_SYSVIEW_UNLOCK() SEGGER_RTT_UNLOCK()
```

4.6.1.4 示例配置

SEGGER_SYSVIEW_Conf.h

```
/*********************************************************************
*            (c) 1995 - 2018 SEGGER Microcontroller GmbH             *
**********************************************************************
---------------------------- END-OF-HEADER -----------------------------

File        : SEGGER_SYSVIEW_Conf.h
Purpose     : SEGGER SysView configuration for Cortex-M3 / Cortex-M4.
*/

#ifndef SEGGER_SYSVIEW_CONF_H
#define SEGGER_SYSVIEW_CONF_H

/*********************************************************************
*
*       SysView timestamp configuration
*/
// Cortex-M cycle counter.
#define SEGGER_SYSVIEW_GET_TIMESTAMP()      ((*(U32 *)(0xE0001004)))
// Number of valid bits low-order delivered as timestamp.
```



```
#define SEGGER_SYSVIEW_TIMESTAMP_BITS        32

/*********************************************************************
*
*       SysView Id configuration
*/
// Default value for the lowest Id reported by the application.
// Can be overridden by the application via SEGGER_SYSVIEW_SetRAMBase().
#define SEGGER_SYSVIEW_ID_BASE          0x20000000
// Number of bits to shift the Id to save bandwidth.
// (e.g. 2 when all reported Ids (pointers) are 4 byte aligned)
#define SEGGER_SYSVIEW_ID_SHIFT         0

/*********************************************************************
*
*       SysView interrupt configuration
*/
// Get the currently active interrupt Id. (read Cortex-M ICSR[8:0]
 = active vector)
#define SEGGER_SYSVIEW_GET_INTERRUPT_ID()    ((*(U32 *)(0xE000ED04)) & 0x1FF)

/*********************************************************************
*
*       SysView locking
*/
// Lock SysView (nestable)
#define SEGGER_SYSVIEW_LOCK()               SEGGER_RTT_LOCK()
// Unlock SysView (nestable)
#define SEGGER_SYSVIEW_UNLOCK()             SEGGER_RTT_UNLOCK()

#endif

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

SEGGER_SYSVIEW_Config_NoOS_CM3.c

```
/*********************************************************************
*                    (c) 1995 - 2018 SEGGER Microcontroller GmbH                    *
*                                嵌入式专家                                         *
*                                www.segger.com                                     *
**********************************************************************

---------------------- END-OF-HEADER  ----------------------------

文件            : SEGGER_SYSVIEW_Config_NoOS.c
目的:没有操作系统的SystemView的样例设置配置。
修订版:$Rev: 9599 $
* /
# include “SEGGER_SYSVIEW.h”
# include “SEGGER_SYSVIEW_Conf.h”

// SystemcoreClock可以在大多数CMSIS兼容项目中使用。
//在非cmsis项目中定义SYSVIEW_CPU_FREQ。
extern unsigned int SystemCoreClock;

/*********************************************************************
*
*            定义,可配置
*
**********************************************************************
* /
//要在SystemViewer中显示的应用程序名称
#定义SYSVIEW_APP_NAME
//目标设备名称
"Demo Application"
                                                “Cortex-M4”
```

```
//时间戳的频率。必须匹配SEGGER_SYSVIEW_Conf.h
#定义SYSVIEW_TIMESTAMP_FREQ                    (SystemCoreClock)

// System Frequency。SystemcoreClock在大多数CMSIS兼容项目中使用。
#define SYSVIEW_CPU_FREQ        (SystemCoreClock)

// The lowest RAM address used for IDs (pointers)
#define SYSVIEW_RAM_BASE        (0x10000000)

// Define as
 1 if the Cortex-M cycle counter is used as SystemView timestamp. Must match SEGGER_SYSVIEW_Conf
#ifndef   USE_CYCCNT_TIMESTAMP
  #define USE_CYCCNT_TIMESTAMP    1
#endif

// Define as
 1 if the Cortex-M cycle counter is used and there might be no debugger attached while recording
#ifndef   ENABLE_DWT_CYCCNT
  #define ENABLE_DWT_CYCCNT
  (USE_CYCCNT_TIMESTAMP & SEGGER_SYSVIEW_POST_MORTEM_MODE)
#endif

/*********************************************************************
*
*       Defines, fixed
*
**********************************************************************
*/
#define DEMCR                     (*(volatile unsigned long*) (0xE000EDFCuL))
  // Debug Exception and Monitor Control Register
#define TRACEENA_BIT              (1uL << 24)
  // Trace enable bit
#define DWT_CTRL                  (*(volatile unsigned long*) (0xE0001000uL))
  // DWT Control Register
#define NOCYCCNT_BIT              (1uL << 25)
  // Cycle counter support bit
#define CYCCNTENA_BIT             (1uL << 0)
  // Cycle counter enable bit

/*********************************************************************
*
*       _cbSendSystemDesc()
*
*  Function description
*    Sends SystemView description strings.
*/
static void _cbSendSystemDesc(void) {
  SEGGER_SYSVIEW_SendSysDesc("N="SYSVIEW_APP_NAME",D="SYSVIEW_DEVICE_NAME);
  SEGGER_SYSVIEW_SendSysDesc("I#15=SysTick");
}

/*********************************************************************
*
*       Global functions
*
**********************************************************************
*/
void SEGGER_SYSVIEW_Conf(void) {
#if USE_CYCCNT_TIMESTAMP
#if ENABLE_DWT_CYCCNT
  //
  // If no debugger is connected, the DWT must be enabled by the application
  //
  if ((DEMCR & TRACEENA_BIT) == 0) {
    DEMCR |= TRACEENA_BIT;
  }
#endif
  //
```

```
  //  The cycle counter must be activated in order
  //  to use time related functions.
  //
  if ((DWT_CTRL & NOCYCCNT_BIT) == 0) {       // Cycle counter supported?
    if ((DWT_CTRL & CYCCNTENA_BIT) == 0) {    // Cycle counter not enabled?
      DWT_CTRL |= CYCCNTENA_BIT;              // Enable Cycle counter
    }
  }
#endif
  SEGGER_SYSVIEW_Init(SYSVIEW_TIMESTAMP_FREQ, SYSVIEW_CPU_FREQ,
                      0, _cbSendSystemDesc);
  SEGGER_SYSVIEW_SetRAMBase(SYSVIEW_RAM_BASE);
}

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

4.6.2 Cortex-M7

与Cortex-M4相同的功能/设置等适用。更多信息请参考第82页的*Cortex-M3 / Cortex-M4*。

缓存

当将SystemView的RTT缓冲区放入可缓存的内存中时，通过J-Link和RTT进行连续记录模式时，性能会略微降低(性能下降< 1%)。这是因为J-Link在访问RTT缓冲区时必须执行缓存维护操作。

4.6.3 Cortex-M0 / Cortex-M0+ / Cortex-M1

Recording mode	Supported?
Continuous recording	Yes
Single-shot recording	Yes
Post-mortem analysis	Yes

4.6.3.1 Cortex-M0事件时间戳

Cortex-M0, Cortex-M0+和Cortex-M1没有循环计数寄存器。事件时间戳必须由应用程序时钟源提供，例如系统计时器SysTick。SEGGER_SYSVIEW_X_GetTimestamp()可以用来实现这个功能。

当SysTick中断在应用程序中使用时，例如由RTOS, SysTick处理程序应该增加SEGGER_SYSVIEW_TickCnt，否则必须向应用程序添加SysTick处理程序并相应地配置。

配置:

```
//
// SEGGER_SYSVIEW_TickCnt has to be defined in the module which
// handles the SysTick and must be incremented in the SysTick
// handler before any SYSVIEW event is generated.
//
// Example in embOS RTOSInit.c:
//
// unsigned int SEGGER_SYSVIEW_TickCnt; // <<-- Define SEGGER_SYSVIEW_TickCnt.
// void SysTick_Handler(void) {
// #if OS_PROFILE
//   SEGGER_SYSVIEW_TickCnt++;  // <<-- Increment SEGGER_SYSVIEW_TickCnt asap.
// #endif
//   OS_EnterNestableInterrupt();
//   OS_TICK_Handle();
//   OS_LeaveNestableInterrupt();
```



```
//}
//
extern unsigned int SEGGER_SYSVIEW_TickCnt;

/*********************************************************************
*
*       Defines, fixed
*
**********************************************************************
*/
#define SCB_ICSR
  (*(volatile U32*) (0xE000ED04uL)) // Interrupt Control State Register
#define SCB_ICSR_PENDSTSET_MASK     (1UL << 26)     // SysTick pending bit
#define SYST_RVR
  (*(volatile U32*) (0xE000E014uL)) // SysTick Reload Value Register
#define SYST_CVR
  (*(volatile U32*) (0xE000E018uL)) // SysTick Current Value Register
/*********************************************************************
*
*       SEGGER_SYSVIEW_X_GetTimestamp()
*
* Function description
*   Returns the current timestamp in ticks using the system tick
*   count and the SysTick counter.
*   All parameters of the SysTick have to be known and are set via
*   configuration defines on top of the file.
*
* Return value
*   The current timestamp.
*
* Additional information
*   SEGGER_SYSVIEW_X_GetTimestamp is always called when interrupts are
*   disabled. Therefore locking here is not required.
*/
U32 SEGGER_SYSVIEW_X_GetTimestamp(void) {
  U32 TickCount;
  U32 Cycles;
  U32 CyclesPerTick;
  //
  // Get the cycles of the current system tick.
  // SysTick is down-counting, subtract the current value from the number of cycles per tick.
  //
  CyclesPerTick = SYST_RVR + 1;
  Cycles = (CyclesPerTick - SYST_CVR);
  //
  // Get the system tick count.
  //
  TickCount = SEGGER_SYSVIEW_TickCnt;
  //
  // If a SysTick interrupt is pending, re-read timer and adjust result
  //
  if ((SCB_ICSR & SCB_ICSR_PENDSTSET_MASK) != 0) {
    Cycles = (CyclesPerTick - SYST_CVR);
    TickCount++;
  }
  Cycles += TickCount * CyclesPerTick;

  return Cycles;
}
```

4.6.3.2 Cortex-M0中断ID

当前活动的中断可以通过读取Cortex-M ICSR[5:0]直接识别，这是中断控制器状态寄存器(ICSR)中的活动向量场。

配置:

```
//
// Get the interrupt Id by reading the Cortex-M ICSR[5:0]
//
#define SEGGER_SYSVIEW_GET_INTERRUPT_ID()    ((*(U32 *)(0xE000ED04)) & 0x3F)
```

4.6.3.3 Cortex-M0 SystemView锁定和解锁

锁定和解锁SystemView以防止传输记录被中断可以通过禁用中断来完成。

SystemView和RTT的Lock和unlock可以是相同的。

配置:

```
//
// RTT locking for GCC toolchains in SEGGER_RTT_Conf.h
//
#define SEGGER_RTT_LOCK()       {                                                \
                                  unsigned int LockState;                        \
                                  __asm volatile ("mrs   %0, primask  \n\t"      \
                                                  "mov   r1, $1     \n\t"        \
                                                  "msr   primask, r1  \n\t"      \
                                                  : "=r" (LockState)             \
                                                  :                              \
                                                  : "r1"                         \
                                                  );                             \
                                }

#define SEGGER_RTT_UNLOCK()       __asm volatile ("msr   primask, %0  \n\t"      \
                                                  :                              \
                                                  : "r" (LockState)              \
                                                  :                              \
                                                  );

//
// Define SystemView locking in SEGGER_SYSVIEW_Conf.h
//
#define SEGGER_SYSVIEW_LOCK()   SEGGER_RTT_LOCK()
#define SEGGER_SYSVIEW_UNLOCK() SEGGER_RTT_UNLOCK()
```

4.6.3.4 Cortex-M0样例配置

SEGGER_SYSVIEW_Conf.h

```
/*********************************************************************
*            (c) 1995 - 2018 SEGGER Microcontroller GmbH             *
**********************************************************************
---------------------------- END-OF-HEADER -----------------------------

File        : SEGGER_SYSVIEW_Conf.h
Purpose     : SEGGER SysView configuration for Cortex-M0, Cortex-M0+,
              and Cortex-M1
*/

#ifndef SEGGER_SYSVIEW_CONF_H
#define SEGGER_SYSVIEW_CONF_H

/*********************************************************************
*
*       SysView timestamp configuration
*/
// Retrieve a system timestamp via user-defined function
```



```
#define SEGGER_SYSVIEW_GET_TIMESTAMP()      SEGGER_SYSVIEW_X_GetTimestamp()
// number of valid bits low-order delivered by SEGGER_SYSVIEW_X_GetTimestamp()
#define SEGGER_SYSVIEW_TIMESTAMP_BITS       32

/*********************************************************************
*
*       SysView Id configuration
*/
// Default value for the lowest Id reported by the application.
// Can be overridden by the application via SEGGER_SYSVIEW_SetRAMBase().
#define SEGGER_SYSVIEW_ID_BASE         0x20000000
// Number of bits to shift the Id to save bandwidth.
// (for example 2 when all reported Ids (pointers) are 4 byte aligned)
#define SEGGER_SYSVIEW_ID_SHIFT        0

/*********************************************************************
*
*       SysView interrupt configuration
*/
// Get the currently active interrupt Id. (read Cortex-M ICSR[8:0]
 = active vector)
#define SEGGER_SYSVIEW_GET_INTERRUPT_ID()    ((*(U32 *)(0xE000ED04)) & 0x3F)

/*********************************************************************
*
*       SysView locking
*/
// Lock SysView (nestable)
#define SEGGER_SYSVIEW_LOCK()              SEGGER_RTT_LOCK()
// Unlock SysView (nestable)
#define SEGGER_SYSVIEW_UNLOCK()            SEGGER_RTT_UNLOCK()

#endif

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

SEGGER_SYSVIEW_Config_embOS_CM0.c

```
/*********************************************************************
*                   (c) SEGGER Microcontroller GmbH                  *
*                        The Embedded Experts                        *
*                           www.segger.com                           *
**********************************************************************

-------------------------- END-OF-HEADER -----------------------------

File    : SEGGER_SYSVIEW_Config_embOS_CM0.c
Purpose : Sample setup configuration of SystemView with embOS
          on Cortex-M0/Cortex-M0+/Cortex-M1 systems which do not
          have a cycle counter.
Revision: $Rev: 25330 $

Additional information:
  SEGGER_SYSVIEW_TickCnt must be incremented in the SysTick
  handler before any SYSVIEW event is generated.

  Example in embOS RTOSInit.c:

  void SysTick_Handler(void) {
  #if (OS_PROFILE != 0)
    SEGGER_SYSVIEW_TickCnt++;  // Increment SEGGER_SYSVIEW_TickCnt before calling
 OS_EnterNestableInterrupt().
  #endif
    OS_EnterNestableInterrupt();
    OS_TICK_Handle();
    OS_LeaveNestableInterrupt();
  }
```



```
* /
# include “RTOS.h”
# include “SEGGER_SYSVIEW.h”
# include “SEGGER_SYSVIEW_embOS.h”

/*********************************************************************************
*
*            定义、固定
*
* *********************************************************************************
* /
#定义SCB_ICSR            (*(volatile U32*) (0xE000ED04uL))
   //中断控制状态寄存器
#定义SCB_ICSR_PENDSTSET_MASK                (1ul << 26)                // SysTick暂挂位
#定义SYST_RVR            (*(volatile U32*) (0xE000E014uL))
   // SysTick重新加载值寄存器
#定义SYST_CVR            (*(volatile U32*) (0xE000E018uL))
   // SysTick当前值寄存器

/*********************************************************************************
*
*            本地函数
*
* *********************************************************************************
* /
/*********************************************************************************
*
*            _cbSendSystemDesc()
*
*    功能描述
*       发送SystemView描述字符串.
* /
_cbSendSystemDesc(void) {
   SEGGER_SYSVIEW_SendSysDesc("N=" SEGGER_SYSVIEW_APP_NAME ",  O=embOS,D=" SEGGER_SYSVIEW_DEVICE_NA . cn "
#ifdef SEGGER_SYSVIEW_SYSDESC0
   SEGGER_SYSVIEW_SendSysDesc (SEGGER_SYSVIEW_SYSDESC0);
# endif
#ifdef SEGGER_SYSVIEW_SYSDESC1
   SEGGER_SYSVIEW_SendSysDesc(SEGGER_SYSVIEW_SYSDESC1);
# endif
#ifdef SEGGER_SYSVIEW_SYSDESC2
   (SEGGER_SYSVIEW_SYSDESC2);
# endif
}

/*********************************************************************************
*
*            全局函数
*
* *********************************************************************************
* /
/*********************************************************************************
*
*            SEGGER_SYSVIEW_Conf()
*
*功能描述
*     配置和初始化SystemView并将其注册到embOS.
*
*附加信息
*     如果启用，SEGGER_SYSVIEW_Conf()也将立即启动
*     用SystemView记录事件.
* /
void SEGGER_SYSVIEW_Conf(void) {
   SEGGER_SYSVIEW_Init (SEGGER_SYSVIEW_TIMESTAMP_FREQ SEGGER_SYSVIEW_CPU_FREQ,
                                    &SYSVIEW_X_OS_TraceAPI,   _cbSendSystemDesc);
   OS_SetTraceAPI(&embOS_TraceAPI_SYSVIEW);                  //配置embOS使用SYSVIEW.
#如果SEGGER_SYSVIEW_START_ON_INIT
```



```
  SEGGER_SYSVIEW_Start();
  // Start recording to catch system initialization.
#endif
}

/*********************************************************************
*
*       SEGGER_SYSVIEW_X_GetTimestamp()
*
* Function description
*   Returns the current timestamp in cycles using the system tick
*   count and the SysTick counter.
*   All parameters of the SysTick have to be known and are set via
*   configuration defines on top of the file.
*
* Return value
*   The current timestamp.
*
* Additional information
*   SEGGER_SYSVIEW_X_GetTimestamp is always called when interrupts are
*   disabled. Therefore locking here is not required.
*/
U32 SEGGER_SYSVIEW_X_GetTimestamp(void) {
  U32 TickCount;
  U32 Cycles;
  U32 CyclesPerTick;
  //
  // Get the cycles of the current system tick.
  // SysTick is down-counting, subtract the current value from the number of cycles per tick.
  //
  CyclesPerTick = SYST_RVR + 1;
  Cycles = (CyclesPerTick - SYST_CVR);
  //
  // Get the system tick count.
  //
  TickCount = SEGGER_SYSVIEW_TickCnt;
  //
  // If a SysTick interrupt is pending, re-read timer and adjust result
  //
  if ((SCB_ICSR & SCB_ICSR_PENDSTSET_MASK) != 0) {
    Cycles = (CyclesPerTick - SYST_CVR);
    TickCount++;
  }
  Cycles += TickCount * CyclesPerTick;

  return Cycles;
}

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

4.6.4 Cortex-A / Cortex-R

Recording mode	Supported?
Continuous recording	Yes/NO
Single-shot recording	Yes
Post-mortem analysis	Yes

连续记录仅在Cortex-A / Cortex-R设备上支持，这些设备通过AHB-AP通过后台存储器访问支持RTT。欲了解更多信息，请参考J-Link用户手册和网站。

4.6.4.1 Cortex-A/R事件时间戳

Cortex-A和Cortex-R周期计数器仅作为性能监视器扩展的一部分实现，可能并不总是可访问的。Cortex-A和Cortex-R也没有像Cortex-M SysTick那样的通用系统定时器源。

关于如何初始化Performance计数器的示例，请参考第98页的*TI AM3358 Cortex-A8样例配置*。

否则事件时间戳必须由应用程序时钟源提供。参考第95页*Renesas RZ/A1 Cortex-A9样例配置*。

对于时钟源，可以使用任何合适的计时器。如果可能的话，建议使用OS系统计时器，因为它通常可以节省额外的配置和资源使用。如果应用程序中没有使用计时器，则必须配置一个合适的计时器以与SystemView一起使用。

一些操作系统实现API函数来获取OS的周期时间。如果这样的函数是可用的，它可以直接使用或由SEGGER_SYSVIEW_X_GetTimestamp()包装。如果OS不提供以周期检索OS时间的功能，则必须实现SEGGER_SYSVIEW_X_Get-Timestamp()以从计时器获取时间戳。

· 计时器应该以1 MHz (1 tick/us)或更快的速度运行。
· 定时器应该在溢出或为零时产生中断
· 计时器应该在自动加载模式

虚拟配置:

```
//
// SEGGER_SYSVIEW_TickCnt has to be defined in the module which
// handles interrupts and must be incremented in the interrupt
// handler as soon as the timer interrupt is acknowledged and
// before any SYSVIEW event is generated.
//
// Example:
//
// unsigned int SEGGER_SYSVIEW_TickCnt; // <<-- Define SEGGER_SYSVIEW_TickCnt.
// void OS_irq_handler(void) {
//   U32  InterruptId;
//   InterruptId  = INTC_ICCIAR & 0x3FF;  // read and extract the interrupt ID
//   if (InterruptId == TIMER_TICK_ID) {
//     SEGGER_SYSVIEW_TickCnt++;  // <<-- Increment SEGGER_SYSVIEW_TickCnt asap.
//   }
//   SEGGER_SYSVIEW_InterruptId
 = InterruptId; // Save active interrupt for SystemView event
//   SEGGER_SYSVIEW_RecordEnterISR();
//   //
//   // Handle interrupt, call ISR
//   //
//   SEGGER_SYSVIEW_RecordExitISR();
// }
//
extern unsigned int SEGGER_SYSVIEW_TickCnt;

/*********************************************************************
*
*       Defines, fixed
*
**********************************************************************
*/
//
// Define the required timer registers here.
//
#define TIMER_RELOAD_VALUE        /* as value which is used to initialize and
 reload the timer */
#define TIMER_COUNT               /* as timer register which holds the current
 counter value */
```

#定义TIMER_INTERRUPT_PENDING() /*作为检查计时器中断是否正在等待*/


```
/*********************************************************************
*
*       SEGGER_SYSVIEW_X_GetTimestamp()
*
* Function description
*   Returns the current timestamp in ticks using the system tick
*   count and the SysTick counter.
*   All parameters of the SysTick have to be known and are set via
*   configuration defines on top of the file.
*
* Return value
*   The current timestamp.
*
* Additional information
*   SEGGER_SYSVIEW_X_GetTimestamp is always called when interrupts are
*   disabled. Therefore locking here is not required.
*/
U32 SEGGER_SYSVIEW_X_GetTimestamp(void) {
  U32 TickCount;
  U32 Cycles;
  U32 CyclesPerTick;
  //
  // Get the cycles of the current system tick.
  // Sample timer is down-counting,
  // subtract the current value from the number of cycles per tick.
  //
  CyclesPerTick = TIMER_RELOAD_VALUE + 1;
  Cycles = (CyclesPerTick - TIMER_COUNT);
  //
  // Get the system tick count.
  //
  TickCount = SEGGER_SYSVIEW_TickCnt;
  //
  // Check if a timer interrupt is pending
  //
  if (TIMER_INTERRUPT_PENDING()) {
    TickCount++;
    Cycles = (CyclesPerTick - TIMER_COUNT);
  }
  Cycles += TickCount * CyclesPerTick;

  return Cycles;
}
```

4.6.4.2 Cortex-A/R中断ID

由于Cortex-A和Cortex-R内核没有内部中断控制器，检索当前活动的中断Id取决于目标设备上使用的中断控制器。必须实现SEGGER_SYSVIEW_GET_INTERRUPT_ID()来匹配这个中断控制器。
下面的配置显示了如何在设备上获得中断Id，其中包括ARM通用中断控制器(GIC)。

对于其他中断控制器，操作可能会有所不同。参考第98页的*TI AM3358 Cortex-A8样例配置*。

由于活动中断Id只能从GIC与中断的确认连接中检索，因此只能读取一次。因此，在通用中断处理程序中确认Id时，必须将其存储在变量中。

虚拟配置:

```
//
// SEGGER_SYSVIEW_InterruptId has to be defined in the module which
// handles the interrupts and must be set to the acknowledged interrupt Id.
```

```
//
// Example:
//
// #define GIC_BASE_ADDR   /* as base address of the GIC on the device */
// #define GICC_BASE_ADDR  (GIC_BASE_ADDR + 0x2000u)
// #define GICC_IAR        (*(volatile unsigned*)(GICC_BASE_ADDR + 0x000C))
//
// unsigned int SEGGER_SYSVIEW_InterruptId; //
 <<-- Define SEGGER_SYSVIEW_InterruptId.
// void OS_irq_handler(void) {
//
//   int_id = GICC_IAR & 0x03FF;  // Read interrupt ID, acknowledge interrupt
//   SEGGER_SYSVIEW_InterruptId = iar_val;
//   OS_EnterInterrupt();         // Inform OS that interrupt handler is running
//   pISR();                      // Call interrupt service routine
//   OS_LeaveInterrupt();
  // Leave interrupt, perform task switch if required
// }
//
extern unsigned int SEGGER_SYSVIEW_InterruptId;

#define SEGGER_SYSVIEW_GET_INTERRUPT_ID()  (SEGGER_SYSVIEW_InterruptId)
```

4.6.4.3 Cortex-A/R SystemView锁定和解锁

由于Cortex-A和Cortex-R内核没有内部中断控制器，锁定和解锁SystemView以防止传输记录被中断可以通过完全禁用FIQ和IRQ来完成，或者通过使用中断控制器特定的方法来完成。下面的配置显示了如何禁用RTT和Sys-temView的所有中断。

SystemView和RTT的Lock和unlock可以是相同的。

配置:

```
//
// RTT locking for GCC toolchains in SEGGER_RTT_Conf.h
// Set and restore IRQ and FIQ mask bits.
//
#define SEGGER_RTT_LOCK()       {                                          \
                                  unsigned int LockState;                  \
                                  __asm volatile ("mrs  r1, CPSR      \n\t" \
                                                  "mov  %0, r1        \n\t" \
                                                  "orr  r1, r1, #0xC0 \n\t" \
                                                  "msr  CPSR_c, r1    \n\t" \
                                                  : "=r" (LockState)       \
                                                  :                        \
                                                  : "r1"                   \
                                                  );

#define SEGGER_RTT_UNLOCK()       __asm volatile ("mov  r0, %0        \n\t" \
                                                  "mrs  r1, CPSR      \n\t" \
                                                  "bic  r1, r1, #0xC0 \n\t" \
                                                  "and  r0, r0, #0xC0 \n\t" \
                                                  "orr  r1, r1, r0    \n\t" \
                                                  "msr  CPSR_c, r1    \n\t" \
                                                  :                        \
                                                  : "r" (LockState)        \
                                                  : "r0", "r1"             \
                                                  );                       \
                                }

//
// Define SystemView locking in SEGGER_SYSVIEW_Conf.h
//
#define SEGGER_SYSVIEW_LOCK()   SEGGER_RTT_LOCK()
```



```
#define SEGGER_SYSVIEW_UNLOCK() SEGGER_RTT_UNLOCK()
```

4.6.4.4 瑞萨RZ/A1 Cortex-A9样例配置

这个瑞萨RZ/A1 (R7S72100)的样例配置从embOS中检索当前活动的中断和系统tick计数器。

它使用OS Timer生成时间戳。RZ/A1包括一个GIC。

SEGGER_SYSVIEW_Conf.h

```
/*********************************************************************************************
*                    (c) 1995 - 2018 SEGGER微控制器有限公司                                    *
* **********************************************************************************************
------------------------- END-OF-HEADER  ------------------------------------------------------------

文件                : SEGGER_SYSVIEW_Conf.h
目的                :瑞萨RZ/A1 Cortex-A的SEGGER SysView配置9
                     与SEGGER embOS.
* /

#ifndef SEGGER_SYSVIEW_CONF_H
#定义SEGGER_SYSVIEW_CONF_H

/*********************************************************************************************
*
*             SysView缓冲区配置
* /
// SysView使用的缓冲区字节数.
//应该大到足以单次记录.
#定义SEGGER_SYSVIEW_RTT_BUFFER_SIZE                  1024 * 1024
// SysView将使用的RTT通道.
#定义SEGGER_SYSVIEW_RTT_CHANNEL                      1

/*********************************************************************************************
*
*             SysView时间戳配置
* /
//通过操作系统特定的函数获取系统时间戳
#定义SEGGER_SYSVIEW_GET_TIMESTAMP ()                 SEGGER_SYSVIEW_X_GetTimestamp()
//////////////////////////////////////////////////////////////////// SEGGER_SYSVIEW_X_GetTimestamp()发送的低阶有效位的个数
#定义SEGGER_SYSVIEW_TIMESTAMP_BITS                   32

/*********************************************************************************************
*
*             SysView中断配置
* /
//
//////////////////////////////////////////////////////////////////// SEGGER_SYSVIEW_InterruptId必须在模块中定义
////////////////////////////////////////////////////////////////////////////////////////////////////////////处理中断，必须设置为已确认的中断Id.
//
// /例如:
//
// #定义GIC_BASE_ADDR               /*作为设备上GIC的基址*/
// #定义GICC_BASE_ADDR              (GIC_BASE_ADDR + 0x2000u)
// #定义GICC_IAR                    (*(volatile unsigned*)(GICC_BASE_ADDR + 0x000C))
//
// unsigned int SEGGER_SYSVIEW_InterruptId;//
  <<——定义SEGGER_SYSVIEW_InterruptId.
// void OS_irq_handler(void) {
//
//       int_id = GICC_IAR & 0x03FF;              //读取中断ID，确认中断
//       SEGGER_SYSVIEW_InterruptId = iar_val;
//       OS_EnterInterrupt ();                    //通知操作系统中断处理程序正在运行
//       pISR ();                                 //调用中断服务例程
//       OS_LeaveInterrupt ();
    //离开中断，需要时执行任务切换
```

```
/ /}
//
extern unsigned int SEGGER_SYSVIEW_InterruptId;

#定义SEGGER_SYSVIEW_GET_INTERRUPT_ID ()                    (SEGGER_SYSVIEW_InterruptId)

/*********************************************************************
*
*                SysView锁定
* /
//锁定SysView (nestable)
#定义SEGGER_SYSVIEW_LOCK ()                                SEGGER_RTT_LOCK ()
//解锁SysView (nestable)
#定义SEGGER_SYSVIEW_UNLOCK ()                              SEGGER_RTT_UNLOCK ()

# endif

/ ************************************************************************* 文件结束 **************************/
```

SEGGER_SYSVIEW_Config_embOS_RZA1.c

```
/*********************************************************************************************************************
*                          (c) 1995 - 2018 SEGGER Microcontroller GmbH                                              *
* ******************************************************************************************************************
---------------------- END-OF-HEADER  ------------------------------------------------------------------------------

文件                    : SEGGER_SYSVIEW_Config_embOS_RZA1.c
目的                    : SystemView与embOS的示例设置配置
                          适用于瑞萨RZ/A1 Cortex-A9.
* /
# include“RTOS.h”
# include“SEGGER_SYSVIEW.h”
# include“SEGGER_SYSVIEW_embOS.h”

////////////////////////////////////////////////////////////////////////////////////////// SystemcoreClock可以在大多数CMSIS兼容项目中使用.
//在非cmsis项目中定义如下. 的SYSVIEW_CPU_FREQ
extern unsigned int SystemCoreClock;

/*********************************************************************************************************************
*
*                定义,可配置
*
* ******************************************************************************************************************
* /
//要在SystemView中显示的应用程序名称
#定义SYSVIEW_APP_NAME                                “embOS演示应用程序”

//目标设备名称
#定义SYSVIEW_DEVICE_NAME                             “R7S72100”

//////////////////////////////////////////////////////////////////////////////////////////时间戳的频率。必须匹配SEGGER_SYSVIEW_Conf.h
//和SEGGER_SYSVIEW_X_GetTimestamp().
#定义SYSVIEW_TIMESTAMP_FREQ                          (399900000u / 12)

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////系统频率。SystemcoreClock在大多数CMSIS兼容项目中使用.
#定义SYSVIEW_CPU_FREQ                                (399900000 u)

// id(指针)使用的最低RAM地址
//如果RAM不从0点开始20000000. ,  应该进行调整
#define SYSVIEW_RAM_BASE                             (0 x60020000)


#定义TIMER_INTERRUPT_PENDING() /*作为检查计时器中断是否正在等待*/


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

```
*                  _cbSendSystemDesc()
*
*    功能描述
*        发送SystemView描述字符串.
* /
_cbSendSystemDesc(void) {
    SEGGER_SYSVIEW_SendSysDesc (N = SYSVIEW_APP_NAME”,D = '' SYSVIEW_DEVICE_NAME);
}

/*********************************************************************************************************
*
*                  全局函数
*
* *********************************************************************************************************
* /
void SEGGER_SYSVIEW_Conf(void) {
    SEGGER_SYSVIEW_Init (SYSVIEW_TIMESTAMP_FREQ SYSVIEW_CPU_FREQ,
                                          &SYSVIEW_X_OS_TraceAPI _cbSendSystemDesc);
    SEGGER_SYSVIEW_SetRAMBase (SYSVIEW_RAM_BASE);
    OS_SetTraceAPI (&embOS_TraceAPI_SYSVIEW);                    //配置embOS使用SYSVIEW.
}

/*********************************************************************************************************
*
*                  SEGGER_SYSVIEW_X_GetTimestamp ()
*
*功能描述
*       使用系统tick返回当前时间戳
*       count和SysTick计数器.
*       必须知道SysTick的所有参数，并通过
*       配置定义在文件. 之上
*
*返回值
*       当前时间戳.
*
*附加信息
*       SEGGER_SYSVIEW_X_GetTimestamp总是在中断时被调用
*       禁用。因此这里不需要锁定.
* /
U32 SEGGER_SYSVIEW_X_GetTimestamp(void) {
    U32 TickCount;
    U32周期;
    U32 CyclesPerTick;
    //
    //获取当前系统tick的周期.
    //样本计时器正在向下计数,
    ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////从每个滴答的周期数中减去当前值.
    //
    CyclesPerTick = 33249 + 1;
    Cycles = (CyclesPerTick - OSTM_CNT);
    //
    //获取系统tick count.
    //
    TickCount = SEGGER_SYSVIEW_TickCnt;
    //
    //检查计时器中断是否正在等待
    //
    if (TIMER_INTERRUPT_PENDING()) {
        TickCount + +;
        Cycles = (CyclesPerTick - OSTM_CNT);
    }
    Cycles += TickCount * CyclesPerTick;

    返回Cycles +;
}

/************************** 文件结束 ***************************/
```

4.6.4.5 TI AM3358 Cortex-A8样例配置

这个TI AM3358的示例配置直接检索当前活动的中断。它初始化并使用Cortex-A性能计数器生成时间戳。

SystemView时间映射生成可以用于其他Cortex-A设备，其中包括性能计数器单元。

SEGGER_SYSVIEW_Conf.h

```
/*********************************************************************
*            (c) 1995 - 2018 SEGGER Microcontroller GmbH             *
**********************************************************************
---------------------------- END-OF-HEADER -----------------------------

File        : SEGGER_SYSVIEW_Conf.h
Purpose     : Generic SEGGER SysView configuration for non-Cortex-M
              devices.
*/

#ifndef SEGGER_SYSVIEW_CONF_H
#define SEGGER_SYSVIEW_CONF_H

/*********************************************************************
*
*       SysView timestamp configuration
*/
// Retrieve a system timestamp via user-defined function
#define SEGGER_SYSVIEW_GET_TIMESTAMP()      SEGGER_SYSVIEW_X_GetTimestamp()
// number of valid bits low-order delivered by SEGGER_SYSVIEW_X_GetTimestamp()
#define SEGGER_SYSVIEW_TIMESTAMP_BITS       32

/*********************************************************************
*
*       SysView Id configuration
*/
// Default value for the lowest Id reported by the application.
// Can be overridden by the application via SEGGER_SYSVIEW_SetRAMBase().
#define SEGGER_SYSVIEW_ID_BASE         0
// Number of bits to shift the Id to save bandwidth.
// (for example 2 when all reported Ids (pointers) are 4 byte aligned)
#define SEGGER_SYSVIEW_ID_SHIFT        0

/*********************************************************************
*
*       SysView interrupt configuration
*/
#define SEGGER_SYSVIEW_GET_INTERRUPT_ID()   SEGGER_SYSVIEW_X_GetInterruptId()

/*********************************************************************
*
*       SysView locking
*/
// Lock SysView (nestable)
#define SEGGER_SYSVIEW_LOCK()              SEGGER_RTT_LOCK()
// Unlock SysView (nestable)
#define SEGGER_SYSVIEW_UNLOCK()            SEGGER_RTT_UNLOCK()

#endif

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

SEGGER_SYSVIEW_Config_embOS_AM3358.c

```
/*********************************************************************
*            (c) 1995 - 2018 SEGGER Microcontroller GmbH             *
```

```
* ********************************************************************************************************
---------------------------- END-OF-HEADER  -----------------------------------------------------------------------------

文件                        : SEGGER_SYSVIEW_Config_embOS_RZA1.c
目的                        : SystemView与embOS的示例设置配置
                              用于TI AM3358 Cortex-A8.
* /
# include “RTOS.h”
# include “SEGGER_SYSVIEW.h”
# include “SEGGER_SYSVIEW_embOS.h”

//
////////////////////////////////////////////////////////////////////////////////////////////////// SystemcoreClock可以在大多数CMSIS兼容项目中使用.
//////////////////////////////////////////////////////////////////////////////////在非cmsis项目中直接定义SYSVIEW_CPU_FREQ.
//
extern unsigned int SystemCoreClock;

/*********************************************************************************************************
*
*                  定义,可配置
*
* ********************************************************************************************************
* /
//要在SystemView中显示的应用程序名称
#如果未定义        sysview_app_name
    #define sysview_app_name                                “embOS启动项目”
# endif

//目标设备名称
#如果未定义        sysview_设备名称
    #define sysview_设备名称                                “AM3358”
# endif

////////////////////////////////////////////////////////////////////////////////////////////////////时间戳的频率。必须匹配SEGGER_SYSVIEW_Conf.h
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////性能计数器频率等于核心时钟频率.
#定义SYSVIEW_TIMESTAMP_FREQ                         (SystemCoreClock)

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////// System Frequency。SystemcoreClock在大多数CMSIS兼容项目中使用.
#如果未定义        SYSVIEW_CPU_FREQ
    #定义SYSVIEW_CPU_FREQ                               (SystemCoreClock)
# endif

// id(指针)使用的最低RAM地址
#如果未定义        sysview_ram_base
    #define sysview_ram_base                            (0 x80000000)
# endif

#如果未定义        SYSVIEW_SYSDESC0
    #定义SYSVIEW_SYSDESC0                               “我# 67 = SysTick # 18 = USB,我# 17 =美国标准局,我# 36 = LCDC”
# endif

#define INTC_BASE                                       (0 x48200000ul)
#定义INTC_SIR_IRQ                                       (*(volatile U32) (INTC_BASE + 0x40uL))

/*********************************************************************************************************
*
*                  本地函数
*
**********************************************************************************************************
* /
/*********************************************************************
*
*                  _cbSendSystemDesc()
*
*     功能描述
*          发送SystemView描述字符串。
* /
_cbSendSystemDesc(void) {
```

```
  SEGGER_SYSVIEW_SendSysDesc("N="SYSVIEW_APP_NAME",O=embOS,D="SYSVIEW_DEVICE_NAME);
#ifdef SYSVIEW_SYSDESC0
  SEGGER_SYSVIEW_SendSysDesc(SYSVIEW_SYSDESC0);
#endif
#ifdef SYSVIEW_SYSDESC1
  SEGGER_SYSVIEW_SendSysDesc(SYSVIEW_SYSDESC1);
#endif
#ifdef SYSVIEW_SYSDESC2
  SEGGER_SYSVIEW_SendSysDesc(SYSVIEW_SYSDESC2);
#endif
}

/*********************************************************************
*
*       _InitPerformanceCounter
*
*  Function description
*    Initialize the internal Cortex-A Performance counter.
*    The function will work for Cortex-A8, Cortex-A9.
*    Please check whether this also suites for your core.
*/
static void _InitPerformanceCounter(U32 PerformReset, I32 UseDivider) {
 //
 // in general enable all counters (including cycle counter)
 //
 I32 Value = 1;

 //
 // Peform reset:
 //
 if (PerformReset) {
   Value |= 2;     // reset all counters to zero.
   Value |= 4;     // reset cycle counter to zero.
 }

 if (UseDivider) {
   Value |= 8;     // enable "by 64" divider for CCNT.
 }
 Value |= 16;

 // program the performance-counter control-register:
 __asm volatile ("MCR p15, 0, %0, c9, c12, 0\t\n"
                 :                  // Output result
                 : "r"(Value)       // Input
                 :                  // Clobbered list
                );
 //
 // Enable all counters
 //
 __asm volatile ("MCR p15, 0, %0, c9, c12, 1\t\n"
                 :                      // Output result
                 : "r"(0x8000000f)      // Input
                 :                      // Clobbered list
                );
 //
 // Clear overflows
 //
 __asm volatile ("MCR p15, 0, %0, c9, c12, 3\t\n"
                 :                      // Output result
                 : "r"(0x8000000f)      // Input
                 :                      // Clobbered list
                );
}

/*********************************************************************
*
*       Global functions
*
```



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

/*********************************************************************
*
*       SEGGER_SYSVIEW_Conf
*
*  Function description
*    Configures SYSVIEW.
*
*    Please check whether this also suites for your core.
*/
void SEGGER_SYSVIEW_Conf(void) {
  //
  // Write USEREN Register
  //
  __asm volatile ("MCR p15, 0, %0, C9, C14, 0\n\t"
                  :              // Output result
                  : "r"(1)       // Input
                  :              // Clobbered list
                 );

  //
  // Disable counter overflow interrupts
  //
  __asm volatile ("MCR p15, 0, %0, C9, C14, 2\n\t"
                  :                   // Output result
                  : "r"(0x8000000f)   // Input
                  :                   // Clobbered list
                 );
  _InitPerformanceCounter(1, 0);
  SEGGER_SYSVIEW_Init(SYSVIEW_TIMESTAMP_FREQ, SYSVIEW_CPU_FREQ,
                      &SYSVIEW_X_OS_TraceAPI, _cbSendSystemDesc);
  SEGGER_SYSVIEW_SetRAMBase(SYSVIEW_RAM_BASE);
  OS_SetTraceAPI(&embOS_TraceAPI_SYSVIEW);    // Configure embOS to use SYSVIEW.
}

/*********************************************************************
*
*       SEGGER_SYSVIEW_X_GetTimestamp()
*
* Function description
*   Returns the current timestamp in ticks using the performance counter.
*
* Return value
*   The current timestamp.
*
* Additional information
*   SEGGER_SYSVIEW_X_GetTimestamp is always called when interrupts are
*   disabled. Therefore locking here is not required.
*/
U32  SEGGER_SYSVIEW_X_GetTimestamp(void) {
  register U32 r = 0;
  //
  // Read CCNT Register
  //
  __asm volatile ("MRC p15, 0, %0, c9, c13, 0"
                  : "+r"(r)    // Output result
                  :            // Inputs
                  : );         // Clobbered list
  return r;
}

/*********************************************************************
*
*       SEGGER_SYSVIEW_X_GetInterruptId()
*
*  Function description
```



```
*    Return the currently active IRQ interrupt number
*    from the INTC_SIR_IRQ.
*/
U32 SEGGER_SYSVIEW_X_GetInterruptId(void) {
  return (INTC_SIR_IRQ & (0x7Fu)); // INTC_SIR_IRQ[6:0]: ActiveIRQ
}
```

4.6.5 瑞萨RX

Recording mode	Supported?
Continuous recording	Yes
Single-shot recording	Yes
Post-mortem analysis	Yes

4.6.5.1 瑞萨RX事件时间戳

事件时间戳必须由应用程序时钟源计时器提供。SEG-GER_SYSVIEW_X_GetTimestamp()可以用来实现这个功能。

在计时器中断中创建任何其他事件之前，中断处理程序应该增加SEGGER_SYSVIEW_TickCnt。

配置:

```
//
// SEGGER_SYSVIEW_TickCnt has to be defined in the module which
// handles the system tick timer and must be incremented in the timer interrupt
// handler before any SYSVIEW event is generated.
//
// Example in embOS RTOSInit.c:
//
// unsigned int SEGGER_SYSVIEW_TickCnt; // <<-- Define SEGGER_SYSVIEW_TickCnt.
// void SysTick_Handler(void) {
// #if OS_PROFILE
//   SEGGER_SYSVIEW_TickCnt++;  // <<-- Increment SEGGER_SYSVIEW_TickCnt asap.
// #endif
//   OS_EnterNestableInterrupt();
//   OS_TICK_Handle();
//   OS_LeaveNestableInterrupt();
// }
//
extern unsigned int SEGGER_SYSVIEW_TickCnt;

/*********************************************************************
*
*           定义、固定
*
**********************************************************************
* /
//系统定时器配置
#定义IRR_BASE_ADDR                    x00087000u)
#define CMT0_VECT                     28你
#定义OS_TIMER_VECT                    CMT0_VECT
#定义TIMER_PRESCALE                   (8 u)
#define CMT0_BASE_ADDR                (0 x00088000u)
#定义CMT0_CMCNT (0                    (*(volatile U16*) (CMT0_BASE_ADDR + 0x04u))

/*********************************************************************
*
*             SEGGER_SYSVIEW_X_GetTimestamp ()
*
* Function description
*       Returns the current timestamp in ticks using the system tick
*       count and the system timer counter.
```

```
*   All parameters of the system timer have to be known and are set via
*   configuration defines on top of the file.
*
* Return value
*   The current timestamp.
*
* Additional information
*   SEGGER_SYSVIEW_X_GetTimestamp is always called when interrupts are
*   disabled. Therefore locking here is not required.
*/
U32 SEGGER_SYSVIEW_X_GetTimestamp(void) {
  U32 Time;
  U32 Cnt;

  Time = SEGGER_SYSVIEW_TickCnt;
  Cnt  = CMT0_CMCNT;
  //
  // Check if timer interrupt pending ...
  //
  if ((*(volatile U8*)(IRR_BASE_ADDR + OS_TIMER_VECT) & (1u << 0u)) != 0u) {
    Cnt = CMT0_CMCNT;       // Interrupt pending, re-read timer and adjust result
    Time++;
  }
  return ((SYSVIEW_TIMESTAMP_FREQ/1000) * Time) + Cnt;
}
```

4.6.5.2 瑞萨RX中断ID

当前活动的中断级别可以用作RX设备上的中断ID。在样例配置中，它由SEG-GER_SYSVIEW_Config_[System]_RX.c中的SEGGER_SYSVIEW_X_GetInterruptId()提供。

配置:

```
//
// Get the interrupt Id via user-provided function
//
#define SEGGER_SYSVIEW_GET_INTERRUPT_ID()   SEGGER_SYSVIEW_X_GetInterruptId()
```

4.6.5.3 瑞萨RX SystemView锁定和解锁

锁定和解锁SystemView以防止传输记录被中断可以通过禁用中断来完成。

SystemView和RTT的Lock和unlock可以是相同的。

配置:

```
//
// RTT locking for IAR toolchains in SEGGER_RTT_Conf.h
//
  #define SEGGER_RTT_LOCK()   {                                       \
                                unsigned long LockState;              \
                                LockState = __get_interrupt_state();  \
                                __disable_interrupt();

  #define SEGGER_RTT_UNLOCK()   __set_interrupt_state(LockState);     \
                              }

//
// Define SystemView locking in SEGGER_SYSVIEW_Conf.h
//
#define SEGGER_SYSVIEW_LOCK()   SEGGER_RTT_LOCK()
#define SEGGER_SYSVIEW_UNLOCK() SEGGER_RTT_UNLOCK()
```


4.6.5.4 瑞萨RX配置样例 - -

SEGGER_SYSVIEW_Conf.h

```
/*********************************************************************
*            (c) 1995 - 2018 SEGGER Microcontroller GmbH             *
**********************************************************************
-------------------------- END-OF-HEADER -----------------------------

File        : SEGGER_SYSVIEW_Conf.h
Purpose     : SEGGER SysView configuration for Renesas RX
*/

#ifndef SEGGER_SYSVIEW_CONF_H
#define SEGGER_SYSVIEW_CONF_H

/*********************************************************************
*
*       SysView timestamp configuration
*/
// Retrieve a system timestamp via user-defined function
#define SEGGER_SYSVIEW_GET_TIMESTAMP()      SEGGER_SYSVIEW_X_GetTimestamp()
// number of valid bits low-order delivered by SEGGER_SYSVIEW_X_GetTimestamp()
#define SEGGER_SYSVIEW_TIMESTAMP_BITS       32

/*********************************************************************
*
*       SysView Id configuration
*/
// Default value for the lowest Id reported by the application.
// Can be overridden by the application via SEGGER_SYSVIEW_SetRAMBase().
#define SEGGER_SYSVIEW_ID_BASE         0
// Number of bits to shift the Id to save bandwidth.
// (for example 2 when all reported Ids (pointers) are 4 byte aligned)
#define SEGGER_SYSVIEW_ID_SHIFT        0

/*********************************************************************
*
*       SysView interrupt configuration
*/
// Get the currently active interrupt Id. (read Cortex-M ICSR[8:0]
 = active vector)
#define SEGGER_SYSVIEW_GET_INTERRUPT_ID()   SEGGER_SYSVIEW_X_GetInterruptId()

/*********************************************************************
*
*       SysView locking
*/
// Lock SysView (nestable)
#define SEGGER_SYSVIEW_LOCK()              SEGGER_RTT_LOCK()
// Unlock SysView (nestable)
#define SEGGER_SYSVIEW_UNLOCK()            SEGGER_RTT_UNLOCK()

#endif

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

SEGGER_SYSVIEW_Config_embOS_CM0.c

```
/*********************************************************************
*            (c) 1995 - 2018 SEGGER Microcontroller GmbH             *
*                        The Embedded Experts                        *
*                           www.segger.com                           *
**********************************************************************

-------------------------- END-OF-HEADER -----------------------------
```

```
文件          : SEGGER_SYSVIEW_Config_NoOS_RX.c
目的:Renesas RX上SystemView的样例设置配置
                没有操作系统的系统.
Revision: $Rev: 18540 $
* /
# include “RTOS.h”
# include “SEGGER_SYSVIEW.h”
# include “SEGGER_SYSVIEW_embOS.h”

//
////////////////////////////////////////////////////// SystemcoreClock可以在大多数CMSIS兼容项目中使用.
//////////////////////////////////////////////在非cmsis项目中直接定义SYSVIEW_CPU_FREQ.
//
extern unsigned int SystemCoreClock;

/*********************************************************************************************
*
*            定义、固定
*
* *********************************************************************************************
* /

/*********************************************************************************************
*
*            定义,可配置
*
* *********************************************************************************************
* /
//要在SystemViewer中显示的应用程序名称
#如果未定义  SYSVIEW_APP_NAME
   #定义SYSVIEW_APP_NAME                    “演示应用程序”
# endif

//目标设备名称
#如果未定义  sysview_设备名称
   #define sysview_设备名称                 “RX64M”
# endif

////////////////////////////////////////////////////////////////////////////系统频率。SystemcoreClock在大多数CMSIS兼容项目中使用.
#如果未定义  SYSVIEW_CPU_FREQ
   #定义SYSVIEW_CPU_FREQ                    (SystemCoreClock)
# endif

////////////////////////////////////////////////////////////////////////////时间戳的频率。必须匹配SEGGER_SYSVIEW_Conf.h和RTOSInit.c
#如果未定义  sysview_timestamp_freq
   #定义sysview_timestamp_freq
   (SYSVIEW_CPU_FREQ/2u/8u) //假设系统定时器运行于
  1/ CPU主频的16位
# endif

// id(指针)使用的最低RAM地址
#如果未定义  sysview_ram_base
   #define sysview_ram_base                 (0)
# endif

#如果未定义  SYSVIEW_SYSDESC0
   #定义SYSVIEW_SYSDESC0
    “我# 0 = IntPrio0, # 1 = IntPrio1我# 2 = IntPrio2我# 3 = IntPrio3我# 4 = IntPrio4”
# endif

/ / #如果未定义   Sysview_sysdesc1
//      #define Sysview_sysdesc1
  “我# 5 = IntPrio5, # 6 = IntPrio6我# 7 = IntPrio7我# 8 = IntPrio8我# 9 = IntPrio9我# 10 = IntPrio10”
/ / # endif

/ / #如果未定义   Sysview_sysdesc2
```



```
//        #定义SYSVIEW_SYSDESC2
   “我# 11 = IntPrio11, # 12 = IntPrio12我# 13 = IntPrio13我# 14 = IntPrio14我# 15 = IntPrio15”
/ / # endif

//系统定时器配置
#定义IRR_BASE_ADDR                          (0 x00087000u)
#define CMT0_VECT                           28u
#定义OS_TIMER_VECT                          CMT0_VECT
#定义TIMER_PRESCALE                         (8 u)
#define CMT0_BASE_ADDR                      (0 x00088000u)
#定义CMT0_CMCNT                             (*(volatile U16*) (CMT0_BASE_ADDR + 0x04u))

extern unsigned SEGGER_SYSVIEW_TickCnt;
     // Tick处理器中递增的Tick计数器值.

/*********************************************************************************************************************
*
*                  _cbSendSystemDesc ()
*
*      功能描述
*           发送SystemView描述字符串.
* /
static void _cbSendSystemDesc(void) {
    SEGGER_SYSVIEW_SendSysDesc (N = SYSVIEW_APP_NAME”,D = " SYSVIEW_DEVICE_NAME);
#ifdef SYSVIEW_SYSDESC0
    SEGGER_SYSVIEW_SendSysDesc(SYSVIEW_SYSDESC0);
# endif
#ifdef SYSVIEW_SYSDESC1
    SEGGER_SYSVIEW_SendSysDesc(SYSVIEW_SYSDESC1);
# endif
#ifdef SYSVIEW_SYSDESC2
    SEGGER_SYSVIEW_SendSysDesc(SYSVIEW_SYSDESC2)
# endif
}

/*********************************************************************************************************************
*
*                  全局函数
*
* ********************************************************************************************************************
* /
void SEGGER_SYSVIEW_Conf(void) {
    SEGGER_SYSVIEW_Init (SYSVIEW_TIMESTAMP_FREQ SYSVIEW_CPU_FREQ,
                                          0, _cbSendSystemDesc);
    SEGGER_SYSVIEW_SetRAMBase (SYSVIEW_RAM_BASE);
}

/*********************************************************************************************************************
*
*                  SEGGER_SYSVIEW_X_GetTimestamp ()
*
*功能描述
*       使用系统tick返回当前时间戳
*       count和SysTick计数器.
*       必须知道SysTick的所有参数，并通过
*       配置定义在文件. 之上
*
*返回值
*       当前时间戳.
*
*附加信息
*       当中断发生时，总是调用SEGGER_SYSVIEW_X_GetTimestamp
*       禁用.
*       因此这里不需要锁，而OS_GetTime_Cycles()可能需要锁
*       被称为.
* /
U32 SEGGER_SYSVIEW_X_GetTimestamp(void) {
    U32时间;
```



```
  U32 Cnt;

  Time = SEGGER_SYSVIEW_TickCnt;
  Cnt  = CMT0_CMCNT;
  //
  // Check if timer interrupt pending ...
  //
  if ((*(volatile U8*)(IRR_BASE_ADDR + OS_TIMER_VECT) & (1u << 0u)) != 0u) {
    Cnt = CMT0_CMCNT;       // Interrupt pending, re-read timer and adjust result
    Time++;
  }
  return ((SYSVIEW_TIMESTAMP_FREQ/1000) * Time) + Cnt;
}

/*********************************************************************
*
*       SEGGER_SYSVIEW_X_GetInterruptId()
*
*  Function description
*    Return the priority of the currently active interrupt.
*/
U32 SEGGER_SYSVIEW_X_GetInterruptId(void) {
  U32 IntId;
 __asm volatile ("mvfc    PSW, %0           \t\n" // Load current PSW
                 "and     #0x0F000000, %0   \t\n" // Clear all except IPL
 ([27:24])
                 "shlr    #24, %0           \t\n" // Shift IPL to [3:0]
                 : "=r" (IntId)                   // Output result
                 :                                // Input
                 :                                // Clobbered list
                );
  return IntId;
}

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

4.6.6 其他cpu

Recording mode	Supported?
Continuous recording	No
Single-shot recording	Yes
Post-mortem analysis	Yes

在cpu上，上面的部分没有涉及到，SystemView也可以在单镜头模式下使用。

要正确运行SystemView，必须配置相同的项:
获取事件时间戳。
· 获取活动中断的中断Id。
· 锁定和解锁SystemView，以防止录制被中断。

4.7 支持的操作系统

下一章描述了哪些(RT)操作系统已经被检测使用Sys-temView以及如何配置它们。

4.7.1 embOS

SEGGER embOS (V4.12a及更高版本)可以在启用分析时为SystemView和其他记录实现生成跟踪事件。

4.7.1.1 为SystemView配置embOS

“OS_LIBMODE_SP”、“OS_LIBMODE_DP”和“OS_LIBMODE_DT”的embOS图书馆配置开启了“profile”功能(详情请参考《embOS用户手册UM01001》)。

除了SYSTEMVIEW和RTT核心模块，以下文件必须包含在应用程序中:

· 对于Cortex-M3和Cortex-M4目标包括SEGGER_SYSVIEW_Config_embOS.c。
· 对于Cortex-M0和Cortex-M1目标包括SEGGER_SYSVIEW_Config_embOS.c。

该文件为SystemView提供了额外所需的功能，并允许配置适合目标系统，如定义应用程序名称、目标设备和目标核心频率。它初始化SYSTEMVIEW模块，并配置embOS向SystemView发送跟踪事件。配置举例请参见第82页“*支持的*cpu”。

在应用程序开始时，在main中，在初始化目标之后，必须调用SEGGER_SYSVIEW_Con-f()来启用SystemView。

现在，当应用程序运行时，SystemView可以连接到目标并开始记录事件。当SystemView被连接或SEGGER_SYSVIEW_Start()被调用时，所有的任务、中断和OS Scheduler活动以及embOS API调用都会被记录下来。

4.7.2 uC/OS-III

SystemView可以与Micrium的uC/OS-III一起使用，以记录任务、中断和调度程序活动。

4.7.2.1 为SystemView配置uC/OS-III

除了SYSTEMVIEW和RTT核心模块之外，以下文件必须包含在应用项目中:

SEGGER_SYSVIEW_Config_uCOSIII.c为Sys-temView提供了额外所需的功能，并允许配置适合目标系统，如定义应用程序名称，目标设备和目标核心频率。SystemView包附带的示例配置文件被配置为用于大多数Cortex-M3、cortex - m4和Cortex-M7目标。有关配置示例，请参见第82页的“*支持的cpu*”。

SEGGER_SYSVIEW_uCOSIII.c和os_trace_events.h提供uC/ OS-III与SystemView的接口。它们通常不需要修改。

os_cfg_trace.h是SystemView所需的最小uc/OS-III Trace配置文件。如果项目已经包含此文件，请确保内容适合应用程序。该文件包括两个定义，用于配置SystemView记录中要管理和命名的最大任务数量和最大资源数量。

```
#define  TRACE_CFG_MAX_TASK                  16u
#define  TRACE_CFG_MAX_RESOURCES             16u
```

启用记录

uC/OS-III事件记录可在os_cfg.h中配置。

定义“OS_CFG_TRACE_EN”为“1u”，开启基本记录。
当OS_CFG_TRACE_API_ENTER_EN定义为1u时，API函数调用也会被记录。

为了记录API函数退出的时间，也可以将OS_CFG_TRACE_API_EXIT_EN定义为1u。

在系统初始化后，在应用程序开始时调用TRACE_INIT()：

```
  [...]
    BSP_Init(); /* Initialize BSP functions       */
    CPU_Init(); /* Initialize the uC/CPU services */

#if (defined(OS_CFG_TRACE_EN) && (OS_CFG_TRACE_EN > 0u))
    /* Initialize uC/OS-III Trace. Should be called after initializing the
 system. */
    TRACE_INIT();
#endif
  [...]
```

4.7.3 uC/OS-II

SystemView可以与Micrium的uC/OS-II一起使用，以记录任务、中断和调度程序活动。SystemView支持已在v2.92.13中添加
关于如何为SystemView配置uC/OS-II的信息，请参见https:// doc.micrium.com/display/osiidoc/SEGGER+SystemView的指南

4.7.3.1 为SystemView配置uC/OS-II

除了SYSTEMVIEW和RTT核心模块之外，以下文件必须包含在应用项目中:

SEGGER_SYSVIEW_Config_uCOSII.c为Sys-temView提供了额外所需的功能，并允许配置适合目标系统，如定义应用程序名称，目标设备和目标核心频率。SystemView包附带的示例配置文件被配置为用于大多数Cortex-M3、cortex - m4和Cortex-M7目标。有关配置示例，请参见第82页的“*支持的cpu*”。

SEGGER_SYSVIEW_uCOSII.c和os_trace_events.h提供uC/OS-II与SystemView的接口。它们通常不需要修改。

os_cfg_trace.h是SystemView所需的最小uc/OS-II Trace配置文件。如果项目已经包含此文件，请确保内容适合应用程序。该文件包括两个定义，用于配置SystemView记录中要管理和命名的最大任务数量和最大资源数量。

```
#define  TRACE_CFG_MAX_TASK                  16u
#define  TRACE_CFG_MAX_RESOURCES             16u
```

启用记录

uC/OS-II事件记录可在os_cfg.h中配置。
定义OS_CFG_TRACE_EN为1u，开启基本记录。
当OS_CFG_TRACE_API_ENTER_EN定义为1u时，API函数调用也会被记录。

为了记录API函数退出的时间，也可以将OS_CFG_TRACE_API_EXIT_EN定义为1u。

在系统初始化后，在应用程序开始时调用TRACE_INIT()：

```
[…]
   BSP_Init ();/*初始化BSP函数                                    * /
   CPU_Init ();/*初始化uC/CPU服务*/
```



```
#if (defined(OS_CFG_TRACE_EN) && (OS_CFG_TRACE_EN > 0u))
    /* Initialize uC/OS-II Trace. Should be called after initializing the system.
 */
    TRACE_INIT();
#endif
  [...]
```

4.7.4 Micrium OS Kernel

SystemView可以与Micrium OS Kernel一起使用，以记录任务、中断和调度程序活动。

4.7.4.1 配置SystemView的Micrium OS Kernel

除了SYSTEMVIEW和RTT核心模块之外，以下文件必须包含在应用项目中:

segger_sysview_config_micriumokernel .c为SystemView提供了额外所需的功能，并允许配置适合目标系统，如定义应用程序名称，目标设备和目标核心频率。SystemView包附带的示例配置文件被配置为用于大多数Cortex-M3、Cortex-M4和Cortex-M7目标器。有关配置示例，请参见第82页的“*支持的cpu*”。

segger_sysview_micriumokernel .c和os_trace_events.h提供了Micrium OS Kernel和SystemView之间的接口。它们通常不需要修改。

os_cfg_trace.h是Sys-temView所需的最小Micrium OS Kernel Trace配置文件。如果项目已经包含此文件，请确保内容适合应用程序。该文件包括两个定义，用于配置SystemView记录中要管理和命名的最大任务数量和最大资源数量。

```
#define  TRACE_CFG_MAX_TASK                  16
#define  TRACE_CFG_MAX_RESOURCES             u  16
```

启用记录

Micrium OS Kernel事件记录可在os_cfg.h中配置。
定义OS_CFG_TRACE_EN为1u，开启基本记录功能。

当OS_CFG_TRACE_API_ENTER_EN定义为1u时，API函数调用也会被记录。

为了记录API函数退出的时间，也可以将OS_CFG_TRACE_API_EXIT_EN定义为1u。

在系统初始化后，在应用程序开始时调用TRACE_INIT():

```
  [...]
    BSP_Init(); /* Initialize BSP functions       */
    CPU_Init(); /* Initialize the uC/CPU services */

#if (defined(OS_CFG_TRACE_EN) && (OS_CFG_TRACE_EN > 0u))
    /* Initialize Micrium OS Kernel Trace. Should be called after initializing
 the system. */
    TRACE_INIT();
#endif
  [...]
```

4.7.5 FreeRTOS

FreeRTOS还可以为SystemView生成跟踪事件，并允许基本但有用的分析而无需修改。

对于更详细的分析，如Scheduler活动和中断，FreeRTOS源代码和使用的端口必须稍微修改。

4.7.5.1 为SystemView配置FreeRTOS

除了SYSTEMVIEW和RTT核心模块之外，segger_sysview_config_free - tos .c必须包含在应用程序中。该文件为SystemView提供了额外所需的功能，并允许配置适合目标系统，如定义应用程序名称、目标设备和目标核心频率。有关配置示例，请参阅第82页的“*支持*的cpu”。

SEGGER_SYSVIEW_FreeRTOS.h头文件必须包含在FreeRTOS-Config.h的末尾或在每次包含FreeRTOS.h的上面。它定义了跟踪宏来创建SYSTEMVIEW事件。

为了获得最好的结果INCLUDE_xTaskGetIdleTaskHandle和INCLUDE_pxTaskGetStack-Start应该在FreeRTOSConfig.h中定义为1。

补丁文件Sample/FreeRTOSV8/ patch /FreeRTOSV8.2.3_Core。补丁显示了FreeRTOS 8.2.3源代码和GCC/ARM_CM4F端口所需的修改。当使用FreeRTOS的其他版本或端口时，它可以作为参考。例如，如果使用GCC/ARM_CM4F以外的其他端口，则必须相应地添加traceISR_ENTER(), traceISR_EXIT()和traceISR_EXIT_TO_SCHEDULER()调用。

补丁文件Sample/FreeRTOSV9/ patch /FreeRTOSV9_Core。patch可用于FreeRTOS V9补丁。

补丁文件Sample/FreeRTOSV10/ patch /FreeRTOSV10_Core. exe补丁可用于FreeRTOS V10.0.0版本的补丁，也可作为FreeRTOS其他版本补丁的参考。

注意:由于FreeRTOS的某些限制，SystemView必须手动存储任务名称。默认情况下，8个任务名将被缓冲。要增加这个值，请转到SEG-GER_SYSVIEW_FreeRTOS.h，并将SYSVIEW_FREERTOS_MAX_NOF_TASKS条目编辑为应用程序中使用的任务数。

4.7.6 nutx

SystemView可以与nutx RTOS一起使用来记录系统活动。
SystemView已被添加到nutx主线中，可以在其安装工具中启用和配置。

更多信息可在nutx项目文档中获得。

4.7.7 Zephyr

SystemView可以用作Zephyr中的记录器来记录事件。更多信息可在Zephyr项目文档中获得。

4.7.8 其他操作系统

其他操作系统还没有正式的检测。
如果您想在其他OS上使用SystemView，请与SEGGER或OS供应商联系。OS插装也可以通过下一章的指南来完成。

4.7.8.1 无OS

SystemView可以在没有任何仪器化OS的情况下使用，以记录中断活动和用户事件。

为SystemView配置系统

除了SYSTEMVIEW和RTT核心模块外，应用程序中还必须包含SEGGER_SYSVIEW_Config_NoOS.c。这个文件提供了所需的基本配置

SystemView的功能，并可以修改以适应系统。有关配置示例，请参见第82页的“*支持*的cpu”。一个额外的SEGGER_SYSVIEW_OS_API指针可以在SEGGER_SYSVIEW_Init中传递，以提供有关系统时间或系统“任务”的信息。

关于如何在系统中记录中断的描述，请参考第121页的*记录中断*。

关于如何在NoOS系统上实现SystemView的通用指南可以在我们的Wiki上找到:https://wiki.segger.com/Use_SystemView_without_RTOS。

第五章

目标上的系统视图

本节描述如何在用户应用程序代码中使用系统视图来扩展仪表化OS生成的分析信息。

5.1 性能标记

性能标记可用于测量系统中的时间，例如例程的执行时间，或者从接收以太网数据包到对其进行分析和处理的延迟。

性能标记显示在带有运行时间和计数的事件列表中，并在时间轴窗口中显示。

要在应用程序中添加性能标记，请使用SEGGER_SYSVIEW_MarkStart()， SEG-GER_SYSVIEW_Mark()和SEGGER_SYSVIEW_MarkStop()。

性能标记由MarkerId标识和关联。为了在系统视图应用中更容易识别，可以使用SEGGER_SYSVIEW_NameMark-er()为每个Id设置一个名称，该名称可以从系统描述回调中调用。

5.2 终端输出

系统视图支持格式化输出到系统视图应用程序。这使得调试日志消息可以显示在事件列表中，这可以为更好的系统分析添加更多信息。

5.3 资源名

资源，如信号量、互斥锁或邮箱，通常作为指针传递给API函数，并作为(压缩)地址记录在事件中。

为了在系统视图应用程序中更容易识别，可以使用SEGGER_SYSVIEW_NameResource()为每个资源地址设置一个名称。

当名称已知，并且OS描述将参数标识为资源时，将显示名称而不是资源地址。

第六章

I配置操作系统和软件模块

本节描述如何将SEGGER SystemView集成到OS或中间件模块中，以便能够记录其执行情况。

6.1 将SEGGER SystemView集成到OS中

SEGGER SystemView可以集成到任何(RT)OS中，以获取有关任务执行、OS内部活动(如调度程序)和OS API调用的信息。对于以下rtos，这种集成已经完成，可以开箱即用。

- SEGGER embOS (V4.12a或更高版本)
- 微晶uC/OS-II
- 微晶uC/OS-III (V3.06或更高版本)
- FreeRTOS (V8.2.3或更高版本)

要集成到其他操作系统中，请联系OS发行商或按照本节中的说明进行集成。

本节中的示例是伪代码，用于说明何时调用特定的SystemView函数。为了允许跟踪检测工具的通用集成，还可以将对这些函数的调用集成为函数宏或通过可配置的跟踪API。

I配置OS核心

为了能够记录任务执行和上下文切换，必须对OS核心进行检测，以便在适当的核心功能上生成SystemView事件。
中断执行在大多数情况下也由OS处理。这允许检测在进入和退出中断时调用的相应OS函数，否则必须为应用程序中的每个ISR执行此操作。

检测OS核心的第三个方面是为更详细的分析提供运行时信息。这些信息包括系统时间，允许SystemView显示相对于应用程序开始的时间戳，而不是记录开始的时间戳，以及任务列表，SystemView使用它来显示任务名称、堆栈信息和按优先级排序任务。

6.1.1 记录任务活动

SystemView可以记录一组预定义的系统事件，用于系统和OS活动的主要信息，如任务执行。这些事件应该由OS在相应的函数中生成。

预定义的事件有:

Event	Description	SystemView API
Task Create	A new task is created.	*SEGGER_SYSVIEW_OnTaskCreate* on page 172
Task Start Ready	A task is marked as ready to start or resume execution.	*SEGGER_SYSVIEW_OnTaskStartReady* on page 174
Task Start Exec	A task is activated, it starts or resumes execution.	*SEGGER_SYSVIEW_OnTaskStartExec* on page 173
Task Stop Ready	A task is blocked or suspended.	*SEGGER_SYSVIEW_OnTaskStopReady* on page 176
Task Stop Exec	A task terminates.	*SEGGER_SYSVIEW_OnTaskStopExec* on page 175
System Idle	No task is executing, the system goes into Idle state.	*SEGGER_SYSVIEW_OnIdle* on page 171

6.1.1.1 任务创建

完成新任务的创建。
任务创建事件发生在系统创建任务时。

在任务创建事件中，使用新任务的Id调用SEGGER_SYSVIEW_OnTaskCreate()。另外，建议使用SEGGER_SYSVIEW_SendTaskInfo()记录新任务的任务信息。

例子

```
void OS_CreateTask(TaskFunc* pF, unsigned Prio, const char* sName, void* pStack) {
  SEGGER_SYSVIEW_TASKINFO Info;
  OS_TASK*                pTask;  // Pseudo struct to be replaced

  [OS specific code ...]

  SEGGER_SYSVIEW_OnTaskCreate((unsigned)pTask);
  memset(&Info, 0, sizeof(Info));
  //
  // Fill elements with current task information
  //
  Info.TaskID     = (U32)pTask;
  Info.sName      = pTask->Name;
  Info.Prio       = pTask->Priority;
  Info.StackBase  = (U32)pTask->pStack;
  Info.StackSize  = pTask->StackSize;
  SEGGER_SYSVIEW_SendTaskInfo(&Info);
}
```

6.1.1.2 任务启动准备就绪

任务被标记为准备开始或恢复执行。

例如，当任务的延迟时间过期时，或者当任务等待的资源可用时，或者当事件被触发时，就会发生任务启动就绪事件。

在任务启动就绪事件调用SEGGER_SYSVIEW_OnTaskStartReady()与任务的Id已经准备好。

例子

```
int OS_HandleTick(void) {
  int TaskReady = 0;    // Pseudo variable indicating a task is ready

  [OS specific code ...]

  if (TaskReady) {
    SEGGER_SYSVIEW_OnTaskStartReady((unsigned)pTask);
  }
}
```

6.1.1.3 任务启动执行

一个任务被激活，它开始或恢复执行。

当上下文即将切换到激活的任务时，将发生任务启动执行事件。这通常由Scheduler在有就绪任务时完成。

在任务启动执行事件调用SEGGER_SYSVIEW_OnTaskStartExec()与将要执行的任务的Id。

例子

```
void OS_Switch(void) {

  [OS specific code ...]

  //
  // If a task is activated
```



```
  //
  SEGGER_SYSVIEW_OnTaskStartExec((unsigned)pTask);
  //
  // Else no task activated, go into idle state
  //
  SEGGER_SYSVIEW_OnIdle()
}
```

6.1.1.4 任务停止准备

任务被阻塞或挂起。
任务停止就绪事件发生在任务被挂起或阻塞时，例如，因为它延迟了特定的时间，或者当它试图声明另一个任务正在使用的资源时，或者当它等待事件发生时。当一个任务被挂起或阻塞时，调度程序将激活另一个任务或进入空闲状态。

在任务停止就绪事件中，调用SEGGER_SYSVIEW_OnTaskStopReady()，其中包含被阻塞的任务Id和一个ReasonId，可以指示任务被阻塞的原因。

例子

```
void OS_Delay(unsigned NumTicks) {

  [OS specific code ...]

  SEGGER_SYSVIEW_OnTaskStopReady(OS_Global.pCurrentTask, OS_CAUSE_WAITING);
}
```

6.1.1.5 任务停止执行

任务终止。
任务停止执行事件发生在任务最终停止执行时，例如当它完成任务并终止时。

在任务停止就绪事件中，调用SEGGER_SYSVIEW_OnTaskStopExec()将当前任务记录为已停止。

例子

```
void OS_TerminateTask(void) {

  [OS specific code ...]

  SEGGER_SYSVIEW_OnTaskStopExec();
}
```

6.1.1.6 系统空闲

没有任务正在执行，系统进入Idle状态。
系统空闲事件发生时，一个任务挂起或停止，没有其他任务准备好。系统可以切换到空闲状态以节省电力，等待中断或任务准备就绪。

在一些操作系统中，空闲是由额外的任务来处理的。在这种情况下，当Idle任务也被激活时，建议记录系统空闲事件。

在SystemView中，Idle状态下花费的时间不会显示为CPU Load。

在系统空闲事件中调用SEGGER_SYSVIEW_OnIdle()。

例子

```
void OS_Switch(void) {

  [OS specific code ...]

  //
  // If a task is activated
  //
  SEGGER_SYSVIEW_OnTaskStartExec((unsigned)pTask);
  //
  // Else no task activated, go into idle state
  //
  SEGGER_SYSVIEW_OnIdle()
}
```

6.1.2 记录中断

SystemView可以记录进入和离开中断服务程序(isr)。Sys-temView API为这些事件提供了函数，当它提供标记中断执行的函数时，应该将这些函数添加到OS中。

当OS调度器由中断(例如SysTick中断)控制时，退出中断事件应该区分恢复正常执行或切换到调度器，并调用适当的SystemView函数。

6.1.2.1 进入中断

当OS提供一个函数来通知OS中断代码正在执行时，在中断服务例程(ISR)开始时调用，OS函数应该调用SEGGER_SYSVIEW_RecordEnterISR()来记录进入中断事件。

当OS不提供进入中断函数，或者ISR不调用它时，用户有责任调用SEGGER_SYSVIEW_RecordEnterISR()来记录中断执行。

SEGGER_SYSVIEW_RecordEnterISR()通过segger_sysview_get_interrupt_id()函数宏自动检索中断ID，该宏定义在SEGGER_SYSVIEW_Conf.h中。

例子

```
void OS_EnterInterrupt(void) {

  [OS specific code ...]

  SEGGER_SYSVIEW_RecordEnterISR();
}
```

6.1.2.2 退出中断

当OS提供一个函数来通知OS中断代码已经执行，在中断服务例程(ISR)结束时调用，OS函数应该调用:

- SEGGER_SYSVIEW_RecordExitISR()当系统将恢复正常执行时。
- 当中断导致上下文切换时，SEGGER_SYSVIEW_RecordExitISRToScheduler()。

例子

```
void OS_ExitInterrupt(void) {

  [OS specific code ...]
  //
  // If the interrupt will switch to the Scheduler
```



```
  //
  SEGGER_SYSVIEW_RecordExitISRToScheduler();
  //
  // Otherwise
  //
  SEGGER_SYSVIEW_RecordExitISR();
}
```

6.1.2.3 示例isr

下面的两个例子展示了如何在SystemView中记录中断执行，有OS中断处理和没有中断处理。

OS处理示例

```
void Timer_Handler(void) {
  //
  // Inform OS about start of interrupt execution
  // (records SystemView Enter Interrupt event).
  //
  OS_EnterInterrupt();
  //
  // Interrupt functionality could be here
  //
  APP_TimerCnt++;
  //
  // Inform OS about end of interrupt execution
  // (records SystemView Exit Interrupt event).
  //
  OS_ExitInterrupt();
}
```

没有OS处理的示例

```
void ADC_Handler(void) {
  //
  // Explicitly record SystemView Enter Interrupt event.
  // Should not be called in high-frequency interrupts.
  //
  SEGGER_SYSVIEW_RecordEnterISR();
  //
  // Interrupt functionality could be here
  //
  APP_ADCValue = ADC.Value;
  //
  // Explicitly record SystemView Exit Interrupt event.
  // Should not be called in high-frequency interrupts.
  //
  SEGGER_SYSVIEW_RecordExitISR();
}
```

6.1.3 记录运行时信息

系统视图可以记录更详细的运行时信息，如系统时间和任务信息。这些信息在录音开始时记录下来，并在系统视图运行时定期请求。

为了请求信息，SEGGER_SYSVIEW_OS_API结构可以在初始化时传递给系统视图，其中包含操作系统特定的回调函数。

设置SEGGER_SYSVIEW_OS_API是可选的，但建议使用该参数，以便系统视图能够显示更详细的信息。

SEGGER_SYSVIEW_OS_API

```
typedef struct {
  U64  (*pfGetTime)      (void);
  void (*pfSendTaskList) (void);
} SEGGER_SYSVIEW_OS_API;
```

参数

Parameter	Description
pfGetTime	Pointer to a function returning the system time.
pfSendTaskList	Pointer to a function recording the entire task list.

6.1.3.1 pfGetTime

描述

获取系统时间，即系统启动后的时间，以微秒为单位。
如果pfGetTime为NULL，系统视图只能显示相对于记录开始的时间戳。

原型

```
U64 (*pfGetTime) (void);
```

6.1.3.2 pfSendTaskList

描述

通过SEGGER_SYSVIEW_SendTaskInfo()记录整个任务列表。
如果pfSendTaskList为NULL，SystemView可能只获得记录时新创建的任务的任务信息。当SystemView应用程序连接以获取当前任务列表的所有信息时，在记录开始时调用pfSendTaskList。

原型

```
void (*pfSendTaskList) (void);
```

例子

```
void cbSendTaskList(void) {
  SEGGER_SYSVIEW_TASKINFO Info;
  OS_TASK*                pTask;

  OS_EnterRegion(); // Disable scheduling to make sure the task list does not change.
  for (pTask = OS_Global.pTask; pTask; pTask = pTask->pNext) {
    //
    // Fill all elements with 0 to allow extending the structure
    // in future version without breaking the code.
    //
    memset(&Info, 0, sizeof(Info));
    //
    // Fill elements with current task information
    //
    Info.TaskID     = (U32)pTask;
    Info.sName      = pTask->Name;
    Info.Prio       = pTask->Priority;
    Info.StackBase  = (U32)pTask->pStackBot;
    Info.StackSize  = pTask->StackSize;
    //
```



```
    // Record current task information
    //
    SEGGER_SYSVIEW_SendTaskInfo(&Info);
  }
  OS_LeaveRegion(); // Enable scheduling again.
}
```

6.1.4 记录OS API调用

除了OS核心工具之外，SystemView还可以记录从应用程序执行的OS API调用。API函数可以像OS核心一样进行检测。

使用SystemView记录API事件可以在传递简单参数时使用SEG-GER_SYSVIEW_RecordXXX()函数来完成，或者通过使用适当的SEGGER_SYSVIEW_EncodeXXX()函数来创建SystemView事件并调用SEGGER_SYSVIEW_SendPacket()来记录它。

例子

```
/*********************************************************************
*
*       OS_malloc()
*
*  Function description
*    API function to allocate memory on the heap.
*/
void OS_malloc(unsigned Size) {
  SEGGER_SYSVIEW_RecordU32(ID_OS_MALLOC,  // Id of OS_malloc (>= 32)
                           Size           // First parameter
                          );

  [OS specific code...]
}
```

为了记录API函数执行的时间并记录其返回值，也可以通过调用SEGGER_SYSVIEW_RecordEnd-Call来记录返回值或调用SEGGER_SYSVIEW_RecordEndCallReturnValue来记录返回值及其返回值来检测API函数的返回值。

6.1.5 OS描述文件

为了使SystemView正确解码API调用，它需要在SystemView的/description/目录中存在一个描述文件。文件名必须为SYSVIEW_<OSName>.txt，其中<OSName>为系统描述中发送的名称。

6.1.5.1 API功能描述

描述文件包含OS可以记录的所有API函数。文件中的每一行都是一个函数，格式如下:

```
<EventID> <FunctionName> <ParameterDescription> | <ReturnValueDescription>
```

<EventId>是为API函数记录的Id。取值范围为32 ~ 511。

<FunctionName> is the name of the API function, displayed in the Event column of Sys-temView. It may not contain spaces.

<ParameterDescription> is the description string of the parameters which are recorded with the API function.

<ReturnValueDescription> is the description string of the return value which can be recorded with SystemView. The ReturnValueDescription is optional.

参数显示可以通过一组修饰符进行设置:

- %b -将参数显示为二进制。
- %B -将参数显示为十六进制字符串(例如00 AA FF…)。
- %d -将参数显示为有符号十进制整数。
- %D -将参数显示为时间值。
- %I -如果系统视图知道资源id，则将参数显示为资源名。%p -将参数显示为4字节十六进制整数(例如0xAABBCCDD)。
- %s -将参数显示为字符串。
- %t -如果系统视图知道任务id，则将参数显示为任务名称。%u -将参数显示为无符号十进制整数。
- %x -将参数显示为十六进制整数。

例子

下面的示例显示了sysview_embs .txt的一部分

```
35        OS_CheckTimer             pGlobal=%p
42        OS_Delay                  Delay=%u
43        OS_DelayUntil             Time=%u
44        OS_setPriority            Task=%t Pri=%u
45        OS_WakeTask               Task=%t
46        OS_CreateTask             Task=%t Pri=%u Stack=%p Size=%u
```

除了默认修饰符之外，描述文件还可以定义NamedTypes来将数值映射到字符串，例如，这对于显示枚举或错误代码的文本值很有用。

NamedTypes有以下格式:

```
NamedType < TypeName > <关键> = <价值> (< Key1 > = < Value1 >…]
```

NamedTypes可以在ParameterDescription和ReturnValueDescription中使用。

例子

```
#
# Types for parameter formatters
#
NamedType OSErr 0=OS_ERR_NONE
NamedType OSErr 10000=OS_ERR_A 10001=OS_ERR_ACCEPT_ISR
NamedType OSErr 12000=OS_ERR_C 12001=OS_ERR_CREATE_ISR
NamedType OSErr 13000=OS_ERR_D 13001=OS_ERR_DEL_ISR

NamedType OSFlag 0=FLAG_NONE 1=FLAG_READ 2=FLAG_WRITE 3=FLAG_READ_WRITE
#
# API Functions
#
34     OSFunc Param=%OSFlag | Returns %OSErr
```

6.1.5.2 任务状态说明

当任务暂停执行时，它的状态记录在SystemView事件中。
该任务状态可以在SystemView中通过TaskS-tate描述转换为文本表示。

任务状态有如下格式:

```
TaskState <Mask> <Key>=<Value>, [<Key1>=<Value1>, ...]
```

例子

```
#
```


Task States

任务状态 0xFF 0=Ready, 1=Delayed or Timeout, 2=Pending, 3=Pending with Timeout, 4=Suspended, 5=Suspended with Timeout, 6=Suspended and Pending, 7=Suspended and Pending with Timeout, 255=Deleted

6.1.5.3 选项描述

也可以在描述文件中设置特定于操作系统的选项来配置SystemView。目前可以在描述文件中插入的选项有:Option ReversePriority:任务优先级高的值等于任务优先级低的值。

6.1.6 OS集成示例

下面的代码显示了基于伪代码片段在OS中集成SystemView的位置，可以作为参考。

```
/*********************************************************************
*                  (c) 1995 - 2018 SEGGER Microcontroller GmbH                  *
*                              嵌入式专家                                        *
*                                  www.segger.com                               *
**********************************************************************
---------------------------- END-OF-HEADER -----------------------------

Purpose : Pseudo-code OS with SEGGER SystemView integration.
*/

/*********************************************************************
*
*       OS_CreateTask()
*
*  Function description
*    Create a new task and add it to the system.
*/
void OS_CreateTask(TaskFunc* pF, unsigned Prio, const char* sName, void* pStack) {
  SEGGER_SYSVIEW_TASKINFO Info;
  OS_TASK*                pTask;  // Pseudo struct to be replaced

  [OS specific code ...]

  SEGGER_SYSVIEW_OnTaskCreate((unsigned)pTask);
  memset(&Info, 0, sizeof(Info));
  //
  // Fill elements with current task information
  //
  Info.TaskID     = (U32)pTask;
  Info.sName      = pTask->Name;
  Info.Prio       = pTask->Priority;
  Info.StackBase  = (U32)pTask->pStack;
  Info.StackSize  = pTask->StackSize;
  SEGGER_SYSVIEW_SendTaskInfo(&Info);
}

/*********************************************************************
*
*       OS_TerminateTask()
*
*  Function description
*    Terminate a task and remove it from the system.
*/
void OS_TerminateTask(void) {

  [OS specific code ...]

  SEGGER_SYSVIEW_OnTaskStopExec();
```

```
}

/*********************************************************************
*
*       OS_Delay()
*
*  Function description
*    Delay and suspend a task for the given time.
*/
void OS_Delay(unsigned NumTicks) {

  [OS specific code ...]

  SEGGER_SYSVIEW_OnTaskStopReady(OS_Global.pCurrentTask, OS_CAUSE_WAITING);
}

/*********************************************************************
*
*       OS_HandleTick()
*
*  Function description
*    OS System Tick handler.
*/
int OS_HandleTick(void) {
  int TaskReady = 0;   // Pseudo variable indicating a task is ready

  [OS specific code ...]

  if (TaskReady) {
    SEGGER_SYSVIEW_OnTaskStartReady((unsigned)pTask);
  }
}

/*********************************************************************
*
*       OS_Switch()
*
*  Function description
*    Switch to the next ready task or go to idle.
*/
void OS_Switch(void) {

  [OS specific code ...]

  //
  // If a task is activated
  //
  SEGGER_SYSVIEW_OnTaskStartExec((unsigned)pTask);
  //
  // Else no task activated, go into idle state
  //
  SEGGER_SYSVIEW_OnIdle()
}

/*********************************************************************
*
*       OS_EnterInterrupt()
*
*  Function description
*    Inform the OS about start of interrupt execution.
*/
void OS_EnterInterrupt(void) {

  [OS specific code ...]

  SEGGER_SYSVIEW_RecordEnterISR();
}
```



```
/*********************************************************************
*
*       OS_ExitInterrupt()
*
*  Function description
*    Inform the OS about end of interrupt execution and switch to
*    Scheduler if necessary.
*/
void OS_ExitInterrupt(void) {

  [OS specific code ...]
  //
  // If the interrupt will switch to the Scheduler
  //
  SEGGER_SYSVIEW_RecordExitISRToScheduler();
  //
  // Otherwise
  //
  SEGGER_SYSVIEW_RecordExitISR();
}
```


6.2 将SEGGER SystemView集成到中间件模块中

SEGGER SystemView还可以集成到中间件模块甚至应用程序模块中，以获取有关这些模块执行的信息，如API调用或中断触发事件。例如，这种集成在SEGGER emos /IP中用于监控通过IP发送和接收数据包，SEGGER emFile用于记录API调用。
对于集成到其他模块，请联系您的经销商或按照本节中的说明进行集成。

6.2.1 注册模块

为了能够记录中间件模块事件，模块必须通过SEGGER_SYSVIEW_RegisterModule()在SystemView注册。
模块传递一个SEGGER_SYSVIEW_MODULE结构指针，它包含关于模块的信息，并接收模块可以生成的事件id的事件偏移量。
注册时必须在SEGGER_SYSVIEW_MODULE结构中设置sDescription和NumEvents。可选的pfSendModuleDesc也可以设置。
在返回SEGGER_SYSVIEW_RegisterModule()时，SEG-GER_SYSVIEW_MODULE结构体的EventOffset被设置为模块可能生成的最低事件Id, pNext被设置为指向下一个注册模块以创建链表。因此，SEGGER_SYSVIEW_MODULE结构体必须是可写的，并且不能在堆栈上分配。

SEGGER_SYSVIEW_MODULE结构体

```
struct SEGGER_SYSVIEW_MODULE {
  const char*                   sModule;
        U32                     NumEvents;
        U32                     EventOffset;
        void                    (*pfSendModuleDesc)(void);
        SEGGER_SYSVIEW_MODULE*  pNext;
};
```

参数

Parameter	Description
sModule	Pointer to a string containing the module name and optionally the module event description.
NumEvents	Number of events the module wants to register.
EventOffset	Offset to be added to the event Ids. Out parameter, set by this function. Do not modify after calling this function.
pfSendModuleDesc	Callback function pointer to send more detailed module description to SystemView.
pNext	Pointer to next registered module. Out parameter, set by this function. Do not modify after calling this function.

例子

```
SEGGER_SYSVIEW_MODULE IPModule = {
  "M=embOSIP, " \
  "0 SendPacket IFace=%u NumBytes=%u, " \
  "1 ReceivePacket Iface=%d NumBytes=%u", // sModule
  2,                                      // NumEvents
  0,
  // EventOffset, Set by SEGGER_SYSVIEW_RegisterModule()
  NULL,
  // pfSendModuleDesc, NULL: No additional module description
```



```
  NULL,
  // pNext, Set by SEGGER_SYSVIEW_RegisterModule()
};

static void _IPTraceConfig(void) {
  //
  // Register embOS/IP at SystemView.
  // SystemView has to be initialized before.
  //
  SEGGER_SYSVIEW_RegisterModule(&IPModule);
}
```

6.2.2 记录模块活动

为了能够记录模块活动，必须对模块进行检测，以便在适当的函数中生成SystemView事件。

通过直接集成SystemView函数，通过可配置的宏函数或一个API结构(可以由SystemView填充和设置)，可以对模块进行检测。

记录事件与SystemView可以完成与准备使用SEG-GER_SYSVIEW_RecordXXX()函数传递简单的参数，或通过使用适当的SEGGER_SYSVIEW_EncodeXXX()函数创建一个SystemView事件，并调用SEGGER_SYSVIEW_SendPacket()记录它。

例子

```
int SendPacket(IP_PACKET *pPacket) {
  //
  // The IP stack sends a packet.
  // Record it according to the module description of SendPacket.
  //
  SEGGER_SYSVIEW_RecordU32x2(
  // Id of SendPacket (0) + Offset for the registered module
                            ID_SENDPACKET + IPModule.EventOffset,
  // First parameter  (displayed as event parameter IFace)
                            pPacket->Interface,
  // Second parameter (displayed as event parameter NumBytes)
                            pPacket->NumBytes
                           );

  [Module specific code...]
}
```

有关更多信息，请参阅第124页的*记录OS API调用*和第132页的API参考。

与操作系统一样，中间件模块描述可以在带有模块名称(M的值=)的描述文件中可用。参考第124页的*OS描述文件*。

6.2.3 提供模块描述

Segger_sysview_module。sModule指向一个字符串，该字符串包含注册模块的基本信息，它是一个逗号分隔的列表，可以包含以下项目:

Item	Identifier	Example
Module name	M	"M=embOSIP"
Module token	T	"T=IP"
Description	S	"S='embOS/IP V12.09'"
Module event	<ID> <Event> <Parameter>	"0 SendPacket IFace=%u NumBytes=%u"

字符串长度不能超过默认值为128的SEGGER_SYSVIEW_MAX_STRING_LEN。

发送额外的描述字符串，并发送由模块使用和记录的资源的名称，Segger_sysview_module。pfSendModuleDesc可以在注册模块时设置。

Segger_sysview_module。pfSendModuleDesc在SystemView连接时定期调用。它可以调用SEGGER_SYSVIEW_RecordModuleDescription()和SEG-GER_SYSVIEW_NameResource()。

例子

```
static void _cbSendIPModuleDesc(void) {
  SEGGER_SYSVIEW_NameResource((U32)&(RxPacketFifo), "Rx FIFO");
  SEGGER_SYSVIEW_NameResource((U32)&(TxPacketFifo), "Tx FIFO");
  SEGGER_SYSVIEW_RecordModuleDescription(&IPModule, "T=IP, S='embOS/IP V12.09'");
}

SEGGER_SYSVIEW_MODULE IPModule = {
  "M=embOSIP, " \
  "0 SendPacket IFace=%u NumBytes=%u, " \
  "1 ReceivePacket Iface=%d NumBytes=%u", // sModule
  2,                                      // NumEvents
  0,                                      // EventOffset, Set by RegisterModule()
  _cbSendIPModuleDesc,                    // pfSendModuleDesc
  NULL,                                   // pNext, Set by RegisterModule()
};
```


第七章 API参考

本节介绍SEGGER SystemView的公共API。

7.1 格式化的输出控制字符串

本节中接受格式化输出控制字符串的函数根据*目标格式化的规范来*执行。

7.1.1 组成

该格式由零个或多个指令组成:普通字符(不是%，它被不加修改地复制到输出流中;以及转换规范，每一个都导致获取零个或多个后续参数，如果适用的话，根据相应的转换说明符对其进行转换，然后将结果写入输出流。

每个转换规范都是由字符%引入的。在%之后依次出现以下内容:

- 修改转换规范含义的零个或多个*标志*(以任何顺序)。
- 可选*最小字段宽度*。如果转换值的字符少于字段宽度，则在字段宽度的左侧(或右侧，如果已给出左侧调整标志)填充空格(默认情况下)。字段宽度采用星号*或十进制整数的形式。
- 可选精度，给出d、u、x和x转换的最小位数。精度采用句点的形式。后面跟一个可选的十进制整数;如果只指定句点，则精度取零。如果精度与任何其他转换说明符一起出现，则行为未定义。
- 指定要应用的转换类型的转换说明符字符。

7.1.2 标记字符

旗帜字符及其含义是:

Flag	Description
-	The result of the conversion is left-justified within the field. The default, if this flag is not specified, is that the result of the conversion is left-justified within the field.
+	The result of a signed conversion *always* begins with a plus or minus sign. The default, if this flag is not specified, is that it begins with a sign only when a negative value is converted.
0	For d, u, x, and X, leading zeros (following any indication of sign or base) are used to pad to the field width rather than performing space padding. If the 0 and - flags both appear, the 0 flag is ignored. For d, u, x, and X conversions, if a precision is specified, the 0 flag is ignored. For other conversions, the behavior is undefined.

7.1.3 长度修饰符

长度修饰符h和l都被忽略。

7.1.4 转换说明符

转换说明符及其含义是:

Flag	Description
d	The argument is converted to signed decimal in the style [-]*dddd*. The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it is expanded with leading spaces. The default precision is one. The result of converting a zero value with a precision of zero is no characters.

Flag	Description
u, x, X	The unsigned argument is converted to unsigned octal for o, unsigned decimal for u, or unsigned hexadecimal notation for x or X in the style *dddd* the letters abcdef are used for x conversion and the letters ABCDEF for X conversion. The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it is expanded with leading spaces. The default precision is one. The result of converting a zero value with a precision of zero is no characters.
c	The argument is converted to an unsigned char, and the resulting character is written.
s	The s specifier is not supported.
p	The argument is a pointer to void. The value of the pointer is converted in the same format as the x conversion specifier with a fixed precision of 2*sizeof(void *).
%	A % character is written. No argument is converted.

7.2 控制功能

应用程序要调用的控制函数。

Function	Description
SEGGER_SYSVIEW_Init()	Initializes the SYSVIEW module.
SEGGER_SYSVIEW_Start()	Start recording SystemView events.
SEGGER_SYSVIEW_Stop()	Stop recording SystemView events.
SEGGER_SYSVIEW_IsStarted()	Handle incoming packets if any and check if recording is started.
SEGGER_SYSVIEW_EnableEvents()	Enable standard SystemView events to be generated.
SEGGER_SYSVIEW_DisableEvents()	Disable standard SystemView events to not be generated.

7.2.1 SEGGER_SYSVIEW_Init()

描述

初始化SYSVIEW模块。必须在SystemView应用程序连接到系统之前调用。

原型

```
无效SEGGER_SYSVIEW_Init(
                          U32                                  SysFreq,
                          U32                                  CPUFreq,
    const SEGGER_SYSVIEW_OS_API * pOSAPI, SEGGER_SYSVIEW_SEND_SYS_DESC_FUNC
      pfSendSysDesc);
```

参数

Parameter	Description
SysFreq	Frequency of timestamp, usually CPU core clock frequency.
CPUFreq	CPU core clock frequency.
pOSAPI	Pointer to the API structure for OS-specific functions.
pfSendSysDesc	Pointer to record system description callback function.

额外的信息

这个函数初始化用于传输SEGGER系统视图数据包的RTT通道。该通道被指定为“SysView”标签，以便客户端软件识别系统视图通道。

该通道用宏SEGGER_SYSVIEW_RTT_CHANNEL配置。

7.2.2 SEGGER_SYSVIEW_Start()

描述

开始记录SystemView事件。
这个函数是由SystemView应用程序在连接时触发的。对于单镜头或事后模式录制，需要由应用程序调用。

原型

void seger_sysview_start (void);

额外的信息

该函数允许传输后续跟踪调用记录的SystemView数据包，并记录SystemView启动事件。

作为start的一部分，发送一个包含系统频率的SystemView初始化包。同时还会发送当前任务列表、当前系统时间和系统描述字符串。

7.2.3 SEGGER_SYSVIEW_Stop()

描述

停止记录SystemView事件。
此函数在断开连接时由SystemView应用程序触发。对于单镜头或事后模式录制，可以由应用程序调用。

原型

```
void SEGGER_SYSVIEW_Stop(void);
```

额外的信息

此功能禁止传输后续跟踪调用记录的SystemView数据包。如果在调用此函数时启用传输，则将单个SystemView Stop事件记录到跟踪，发送，然后停止跟踪传输。

7.2.4 SEGGER_SYSVIEW_IsStarted()

描述

处理传入的数据包(如果有)，并检查是否开始录音。

原型

```
int SEGGER_SYSVIEW_IsStarted(void);
```

返回值

0:录音未开始。
>0录音已开始。

7.2.5 SEGGER_SYSVIEW_EnableEvents()

描述

启用生成标准SystemView事件。

原型

```
void SEGGER_SYSVIEW_EnableEvents(U32 EnableMask);
```

参数

Parameter	Description
EnableMask	Events to be enabled.

7.2.6 SEGGER_SYSVIEW_DisableEvents ()

描述

禁用不生成标准SystemView事件。

原型

void SEGGER_SYSVIEW_DisableEvents(U32);

参数

Parameter	Description
DisableMask	Events to be disabled.

7.3 配置功能

应用系统调用的配置函数。通常包含在系统回调函数中。

Function	Description
SEGGER_SYSVIEW_SetRAMBase()	Sets the RAM base address, which is subtracted from IDs in order to save bandwidth.
SEGGER_SYSVIEW_SendTaskList()	Send all tasks descriptors to the host.
SEGGER_SYSVIEW_SendTaskInfo()	Send a Task Info Packet, containing TaskId for identification, task priority and task name.
SEGGER_SYSVIEW_SendSysDesc()	Send the system description string to the host.
SEGGER_SYSVIEW_SendPacket()	Send an event packet.

7.3.1 SEGGER_SYSVIEW_SetRAMBase()

描述

设置RAM基址，从id中减去，以节省带宽。

原型

void SEGGER_SYSVIEW_SetRAMBase(U32 RAMBaseAddress);

参数

Parameter	Description
RAMBaseAddress	Lowest RAM Address. (i.e. 0x20000000 on most Cortex-M)

7.3.2 SEGGER_SYSVIEW_SendTaskInfo ()

描述

发送一个任务信息包，包含用于标识、任务优先级和任务名称的TaskId。

原型

```
void SEGGER_SYSVIEW_SendTaskInfo(const SEGGER_SYSVIEW_TASKINFO * pInfo);
```

参数

Parameter	Description
pInfo	Pointer to task information to send.

7.3.3 SEGGER_SYSVIEW_SendTaskList()

描述

向主机发送所有任务描述符。

原型

```
void SEGGER_SYSVIEW_SendTaskList(void);
```


7.3.4 SEGGER_SYSVIEW_SendSysDesc()

描述

向主机发送系统描述字符串。SystemView应用程序使用系统描述信息来识别当前应用程序并相应地处理事件。

系统描述通常由系统描述回调调用，以确保它只在SystemView应用程序连接时发送。

原型

```
void SEGGER_SYSVIEW_SendSysDesc(const char * sSysDesc);
```

参数

Parameter	Description
sSysDesc	Pointer to the 0-terminated system description string.

额外的信息

一个系统描述字符串不能超过SEGGER_SYSVIEW_MAX_STRING_LEN字符。可以记录多个描述字符串。

在一个系统描述字符串中可以描述以下项目。每个项目由其标识符标识，后面跟着' = '和值。项目之间用‘,’分隔。

Item	Identifier	Example
Application name	N	“N=Test Application”
Operating system	O	“O=embOS”
Additional module	M	“M=embOS/IP”
Target device	D	“D=MK66FN2M0xxx18”
Target core	C	“C=Cortex-M4”
Interrupt	I#<InterruptID>	“I#15=SysTick”

字符串示例

- N=Test Application,O=embOS,D=MK66FN2M0xxx18
- I#15=SysTick,I#99=ETH_Tx,I#100=ETH_Rx

7.3.5 SEGGER_SYSVIEW_SendModule()

描述

向主机发送已注册模块的信息。

原型

```
void SEGGER_SYSVIEW_SendModule(U8 ModuleId);
```

参数

Parameter	Description
ModuleId	Id of the requested module.

7.3.6 SEGGER_SYSVIEW_SendModuleDescription()

描述

触发已注册模块描述的发送。

原型

SEGGER_SYSVIEW_SendModuleDescription(void);

7.3.7 SEGGER_SYSVIEW_SendNumModules()

描述

向主机发送已注册模块数。

原型

SEGGER_SYSVIEW_SendNumModules(void);

7.4 应用级事件记录功能

用户应用程序中要调用的用户事件记录函数。

Function	Description
Markers	
SEGGER_SYSVIEW_MarkStart()	Record a Performance Marker Start event to start measuring runtime.
SEGGER_SYSVIEW_Mark()	Record a Performance Marker intermediate event.
SEGGER_SYSVIEW_MarkStop()	Record a Performance Marker Stop event to stop measuring runtime.
Resources	
SEGGER_SYSVIEW_NameResource()	Send the name of a resource to be displayed in SystemView.
Plain output functions	
SEGGER_SYSVIEW_Print()	Print a string to the host.
SEGGER_SYSVIEW_Warn()	Print a warning string to the host.
SEGGER_SYSVIEW_Error()	Print an error string to the host.
Host-based formatting	
SEGGER_SYSVIEW_PrintfHost()	Print a string which is formatted on the host by the SystemView Application.
SEGGER_SYSVIEW_PrintfHostEx()	Print a string which is formatted on the host by the SystemView Application with Additional information.
SEGGER_SYSVIEW_WarnfHost()	Print a warning string which is formatted on the host by the SystemView Application.
SEGGER_SYSVIEW_ErrorfHost()	Print an error string which is formatted on the host by the SystemView Application.
Target-based formatting	
SEGGER_SYSVIEW_PrintfTarget()	Print a string which is formatted on the target before sent to the host.
SEGGER_SYSVIEW_PrintfTargetEx()	Print a string which is formatted on the target before sent to the host with Additional information.
SEGGER_SYSVIEW_WarnfTarget()	Print a warning string which is formatted on the target before sent to the host.
SEGGER_SYSVIEW_ErrorfTarget()	Print an error string which is formatted on the target before sent to the host.

7.4.1 SEGGER_SYSVIEW_MarkStart()

描述

记录性能标记启动事件以开始测量运行时。

原型

void SEGGER_SYSVIEW_MarkStart(unsigned markid);

参数

Parameter	Description
MarkerId	User defined ID for the marker.

7.4.2 SEGGER_SYSVIEW_Mark()

描述

记录性能标记中间事件。

原型

```
void SEGGER_SYSVIEW_Mark(unsigned int markid);
```

参数

Parameter	Description
MarkerId	User defined ID for the marker.

7.4.3 SEGGER_SYSVIEW_MarkStop()

描述

记录性能标记停止事件以停止测量运行时。

原型

无效SEGGER_SYSVIEW_MarkStop(unsigned markid);

参数

Parameter	Description
MarkerId	User defined ID for the marker.

7.4.4 SEGGER_SYSVIEW_NameMarker ()

描述

发送要在SystemView中显示的性能标记的名称。
标记名称通常在系统描述回调中设置，以确保仅在连接SystemView应用程序时才发送。

原型

```
void SEGGER_SYSVIEW_NameMarker(      unsigned int   MarkerId,
                               const char         * sName);
```

参数

Parameter	Description
MarkerId	User defined ID for the marker.
sName	Pointer to the marker name. (Max. SEGGER_SYSVIEW_MAX_STRING_LEN Bytes)

7.4.5 SEGGER_SYSVIEW_NameResource ()

描述

发送要在SystemView中显示的资源的名称。
标记名称通常在系统描述回调中设置，以确保仅在连接SystemView应用程序时才发送。

原型

```
void SEGGER_SYSVIEW_NameResource(U32 ResourceId,
                                 const char * sName);
```

参数

Parameter	Description
ResourceId	Id of the resource to be named. i.e. its address.
sName	Pointer to the resource name. (Max. SEGGER_SYSVIEW_MAX_STRING_LEN Bytes)

7.4.6 SEGGER_SYSVIEW_Print()

描述

打印一个字符串到主机。

原型

void SEGGER_SYSVIEW_Print (const char * s);

参数

Parameter	Description
s	String to sent.

7.4.7 SEGGER_SYSVIEW_PrintfHost()

描述

打印一个由SystemView应用程序在主机上格式化的字符串。

原型

```
void SEGGER_SYSVIEW_PrintfHost(const char * s,
                               ⋯);
```

参数

Parameter	Description
s	String to be formatted.

额外的信息

所有格式参数都被视为32位标量值。

7.4.8 SEGGER_SYSVIEW_PrintfHostEx()

描述

打印一个由SystemView应用程序在主机上格式化的字符串，其中包含附加信息。

原型

```
void SEGGER_SYSVIEW_PrintfHostEx(const char * s,
                                 U32选项,
                                 …);
```

参数

Parameter	Description
s	String to be formatted.
Options	Options for the string. i.e. Log level.

额外的信息

所有格式参数都被视为32位标量值。

7.4.9 SEGGER_SYSVIEW_PrintfTarget ()

描述

打印一个字符串，该字符串在发送到主机之前在目标器上进行了格式化。

原型

```
void SEGGER_SYSVIEW_PrintfTarget(const char * s,
                                 ...);
```

参数

Parameter	Description
s	String to be formatted.

7.4.10 SEGGER_SYSVIEW_PrintfTargetEx()

描述

打印一个字符串，该字符串在发送到主机之前在目标器上进行了格式化，并带有附加信息。

原型

```
void SEGGER_SYSVIEW_PrintfTargetEx(const char * s,
                                   U32选项,
                                   …);
```

参数

Parameter	Description
s	String to be formatted.
Options	Options for the string. i.e. Log level.

7.4.11 SEGGER_SYSVIEW_Warn()

描述

向主机输出警告字符串。

原型

```
void SEGGER_SYSVIEW_Warn(const char * s);
```

参数

Parameter	Description
s	String to sent.

7.4.12 SEGGER_SYSVIEW_WarnfHost()

描述

打印由SystemView应用程序在主机上格式化的警告字符串。

原型

```
void SEGGER_SYSVIEW_WarnfHost(const char * s,
                              ...);
```

参数

Parameter	Description
s	String to be formatted.

额外的信息

所有格式参数都被视为32位标量值。

7.4.13 SEGGER_SYSVIEW_WarnfTarget ()

描述

打印一个警告字符串，该字符串在发送到主机之前在目标器上进行了格式化。

原型

```
void SEGGER_SYSVIEW_WarnfTarget(const char * s,
                                ...);
```

参数

Parameter	Description
s	String to be formatted.

7.4.14 SEGGER_SYSVIEW_Error()

描述

向主机输出错误字符串。

原型

```
void SEGGER_SYSVIEW_Error(const char * s);
```

参数

Parameter	Description
s	String to sent.

7.4.15 SEGGER_SYSVIEW_ErrorfHost ()

描述

打印SystemView应用程序在主机上格式化的错误字符串。

原型

```
void SEGGER_SYSVIEW_ErrorfHost(const char * s)
                                                                    …);
```

参数

Parameter	Description
s	String to be formatted.

额外的信息

所有格式参数都被视为32位标量值。

7.4.16 SEGGER_SYSVIEW_ErrorfTarget()

描述

打印一个错误字符串，该字符串在发送到主机之前在目标端进行了格式化。

原型

```
void SEGGER_SYSVIEW_ErrorfTarget(const char * s)
                                                                    …);
```

参数

Parameter	Description
s	String to be formatted.

7.5 模块和RTOS对象函数

中间件模块注册和配置功能。

Function	Description
SEGGER_SYSVIEW_RegisterModule()	Register a middleware module for recording its events.
SEGGER_SYSVIEW_RecordModuleDescription()	Sends detailed information of a registered module to the host.

7.5.1 SEGGER_SYSVIEW_RegisterModule ()

描述

注册一个中间件模块以记录其事件。

原型

void SEGGER_SYSVIEW_RegisterModule(SEGGER_SYSVIEW_MODULE * pModule)

参数

Parameter	Description
pModule	The middleware module information.

额外的信息

SEGGER_SYSVIEW_MODULE元素:sDescription -指向包含模块名称和可选模块事件描述的字符串的指针。NumEvents -模块想要注册的事件数。EventOffset -要添加到事件id的偏移量。Out参数，由本函数设置。调用此函数后不要修改。pfSendModuleDesc -回调函数指针，发送更详细的模块描述给SystemView Application。pNext -指向下一个注册模块的指针。Out参数，由本函数设置。调用此函数后不要修改。

7.5.2 SEGGER_SYSVIEW_RecordModuleDescription()

描述

向主机发送已注册模块的详细信息。

原型

```
void SEGGER_SYSVIEW_RecordModuleDescription
                                        (const SEGGER_SYSVIEW_MODULE * pModule,
  const char                             * sDescription);
```

参数

Parameter	Description
pModule	Pointer to the described module.
sDescription	Pointer to a description string.

7.6 实时事件记录功能

操作系统相关的事件记录函数，由操作系统仪表调用。

Function	Description
High-level RTOS state recording	
SEGGER_SYSVIEW_OnIdle()	Record an Idle event.
SEGGER_SYSVIEW_OnTaskCreate()	Record a Task Create event.
SEGGER_SYSVIEW_OnTaskStartExec()	Record a Task Start Execution event.
SEGGER_SYSVIEW_OnTaskStartReady()	Record a Task Start Ready event.
SEGGER_SYSVIEW_OnTaskStopExec()	Record a Task Stop Execution event.
SEGGER_SYSVIEW_OnTaskStopReady()	Record a Task Stop Ready event.
SEGGER_SYSVIEW_OnTaskTerminate()	Record a Task termination event.
Low-level realtime recording	
SEGGER_SYSVIEW_RecordEnterISR()	Format and send an ISR entry event.
SEGGER_SYSVIEW_RecordExitISR()	Format and send an ISR exit event.
SEGGER_SYSVIEW_RecordExitISRToScheduler()	Format and send an ISR exit into scheduler event.
SEGGER_SYSVIEW_RecordEnterTimer()	Format and send a Timer entry event.
SEGGER_SYSVIEW_RecordExitTimer()	Format and send a Timer exit event.
SEGGER_SYSVIEW_RecordSystime()	Formats and sends a SystemView Systime containing a single U64 or U32 parameter payload.

7.6.1 SEGGER_SYSVIEW_OnIdle()

描述

记录一个Idle事件。

原型

```
void seger_sysview_onidle (void);
```


7.6.2 SEGGER_SYSVIEW_OnTaskCreate()

描述

记录“任务创建”事件。“任务创建”事件对应于在操作系统中创建任务。

原型

```
void SEGGER_SYSVIEW_OnTaskCreate(U32 TaskId);
```

参数

Parameter	Description
TaskId	Task ID of created task.

7.6.3 SEGGER_SYSVIEW_OnTaskStartExec()

描述

记录任务开始执行事件。任务开始事件对应于任务开始执行的时间，而不是准备执行的时间。

原型

void SEGGER_SYSVIEW_OnTaskStartExec(U32 TaskId);

参数

Parameter	Description
TaskId	Task ID of task that started to execute.

7.6.4 SEGGER_SYSVIEW_OnTaskStartReady ()

描述

记录“任务开始准备就绪”事件。

原型

```
void SEGGER_SYSVIEW_OnTaskStartReady(U32 TaskId);
```

参数

Parameter	Description
TaskId	Task ID of task that started to execute.

7.6.5 SEGGER_SYSVIEW_OnTaskStopExec()

描述

记录任务停止执行事件。任务停止事件对应于任务停止执行和终止的时间。

原型

void seger_sysview_ontaskstopexec (void);

7.6.6 SEGGER_SYSVIEW_OnTaskStopReady()

描述

记录任务停止准备事件。

原型

```
SEGGER_SYSVIEW_OnTaskStopReady(U32 TaskId)
                               unsigned int Cause);
```

参数

Parameter	Description
TaskId	Task ID of task that completed execution.
Cause	Reason for task to stop (i.e. Idle/Sleep)

7.6.7 SEGGER_SYSVIEW_OnTaskTerminate()

描述

记录一个Task终止事件。Task终止事件对应于在操作系统中终止一个任务。如果TaskId是当前活动的任务，也可以使用SEGGER_SYSVIEW_OnTaskStopExec。

原型

void SEGGER_SYSVIEW_OnTaskTerminate(U32 TaskId);

参数

Parameter	Description
TaskId	Task ID of terminated task.

7.6.8 SEGGER_SYSVIEW_RecordEnterISR()

描述
格式化并发送一个ISR条目事件。

原型

```
void SEGGER_SYSVIEW_RecordEnterISR(void);
```

额外的信息
发送报文数示例02 0F 50 // ISR(15)输入。时间戳为80 (0x50)

7.6.9 SEGGER_SYSVIEW_RecordExitISR ()

描述

格式化并发送一个ISR退出事件。

原型

空白SEGGER_SYSVIEW_RecordExitISR(无效);

额外的信息

格式如下:03 <时间戳> //最大时间。packet len = 6发送报文示例03 20 // ISR
退出。时间戳为32 (0x20)

7.6.10 SEGGER_SYSVIEW_RecordExitISRToScheduler()

描述

格式化并将ISR出口发送到调度程序事件。

原型

seger_sysview_recordexitisrtoscheduler (void);

额外的信息

格式如下:18 <时间戳> //最大时间。包len为6

发送的示例包18 20 // ISR退出到调度程序。时间戳为32 (0x20)

7.6.11 SEGGER_SYSVIEW_RecordEnterTimer ()

描述

格式化并发送一个Timer条目事件。

原型

void SEGGER_SYSVIEW_RecordEnterTimer(U32定时器);

参数

Parameter	Description
TimerId	Id of the timer which starts.

7.6.12 SEGGER_SYSVIEW_RecordExitTimer()

描述

格式化并发送Timer退出事件。

原型

```
void SEGGER_SYSVIEW_RecordExitTimer(void);
```


7.6.13 SEGGER_SYSVIEW_RecordSystime()

描述

格式化并发送包含单个U64或U32参数payload的SystemView Systime。

原型

SEGGER_SYSVIEW_RecordSystime(void);

7.7 动态内存监控功能

这些函数提供对系统堆和其他动态分配对象的监控。

Function	Description
SEGGER_SYSVIEW_HeapDefine()	Define heap.
SEGGER_SYSVIEW_HeapAlloc()	Record a system-heap allocation event.
SEGGER_SYSVIEW_HeapAllocEx()	Record a per-heap allocation event.
SEGGER_SYSVIEW_HeapFree()	Record a heap deallocation event.

7.7.1 SEGGER_SYSVIEW_HeapDefine ()

描述

定义堆。

原型

```
void SEGGER_SYSVIEW_HeapDefine(void          * pHeap,
                               void          * pBase,
                               unsigned int    HeapSize,
                               unsigned int    MetadataSize);
```

参数

Parameter	Description
pHeap	Pointer to heap control structure.
pBase	Pointer to managed heap memory.
HeapSize	Size of managed heap memory in bytes.
MetadataSize	Size of metadata associated with each heap allocation.

额外的信息

SystemView可以跨多个堆跟踪分配。

HeapSize必须是目标的自然对齐单位的倍数。这个大小受制于压缩，由SEGGER_SYSVIEW_ID_SHIFT的特定设置控制。

MetadataSize定义每个分配的元数据的大小。对于许多堆实现，元数据大小是机器字长的倍数，通常包含分配块的大小(在释放时使用)，指向前面和/或后面块的可选指针，以及标识块所有者的可选标记。请注意，MetadataSize不是在SystemView数据包中压缩的，也不需要是1<<SEGGER_SYSVIEW_ID_SHIFT的倍数。

7.7.2 SEGGER_SYSVIEW_HeapAlloc()

描述

记录一个系统堆分配事件。

原型

```
SEGGER_SYSVIEW_HeapAlloc      无效          * pHeap,
                              unsigned int  * pUserData;
                                              UserDataLen);
```

参数

Parameter	Description
pHeap	Pointer to heap where allocation was made.
pUserData	Pointer to allocated user data.
UserDataLen	Size of block allocated to hold user data, excluding any metadata.

额外的信息

用户数据必须为架构正确对齐，这通常要求对齐至少是double或long long的对齐。因此，pUserData通过压缩id来压缩，由SEG-GER_SYSVIEW_ID_SHIFT的特定设置控制。

以同样的方式，UserDataLen必须反映分配块的大小，而不是应用程序请求的分配大小。这个大小也会受到压缩，由SEGGER_SYSVIEW_ID_SHIFT的具体设置控制。

例如，假设分配器运行在一个Cortex-M设备上，SEG-GER_SYSVIEW_ID_SHIFT设置为2(设备的字对齐方式)。如果用户请求5字节的分配，假设的堆分配器可以为此分配大小为32字节的块。发送到SystemView进行记录的UserDataLen的值应该是32，而不是5，并且32是通过移动两个比特来压缩的，这是SEG-GER_SYSVIEW_ID_SHIFT的配置值，它描述了从托管内存中消耗的字节数，SystemView可以从这些字节数中计算精确的堆度量。

7.7.3 SEGGER_SYSVIEW_HeapAllocEx()

描述
记录每堆分配事件。

原型
SEGGER_SYSVIEW_HeapAllocEx

```
无效                * pHeap,
无符号整型          * pUserData;
unsigned int          UserDataLen;
                      标签);
```

参数

Parameter	Description
pHeap	Pointer to heap where allocation was made.
pUserData	Pointer to allocated user data.
UserDataLen	Size of block allocated to hold user data, excluding any metadata.
Tag	Block tag, typically used to identify the owner of the block.

额外的信息
用户数据必须为架构正确对齐，这通常要求对齐至少是double或long long的对齐。因此，userdata通过压缩id来压缩，由SEG-GER_SYSVIEW_ID_SHIFT的特定设置控制。

以同样的方式，UserDataLen必须反映分配块的大小，而不是应用程序请求的分配大小。这个大小也会受到压缩，由SEGGER_SYSVIEW_ID_SHIFT的具体设置控制。

例如，假设分配器运行在一个Cortex-M设备上，SEG-GER_SYSVIEW_ID_SHIFT设置为2(设备的字对齐方式)。如果用户请求5字节的分配，假设的堆分配器可以为此分配大小为32字节的块。发送到SystemView进行记录的UserDataLen的值应该是32，而不是5，并且32是通过移动两个比特来压缩的，这是SEG-GER_SYSVIEW_ID_SHIFT的配置值，它描述了从托管内存中消耗的字节数，SystemView可以从这些字节数中计算精确的堆度量。

7.7.4 SEGGER_SYSVIEW_HeapFree()

描述

记录一个堆释放事件。

原型

```
void SEGGER_SYSVIEW_HeapFree(void * phap,
                             void * userdata);
```

参数

Parameter	Description
pHeap	Pointer to heap where allocation was made.
pUserData	Pointer to allocated user data.

额外的信息

SystemViews跟踪分配并知道分配数据的大小。

7.8 高级API仪表功能

OS和模块插装调用的事件记录函数。

Function	Description
API Function Call	
SEGGER_SYSVIEW_RecordVoid()	Formats and sends a SystemView packet with an empty payload.
SEGGER_SYSVIEW_RecordU32()	Formats and sends a SystemView packet containing a single U32 parameter payload.
SEGGER_SYSVIEW_RecordU32x2()	Formats and sends a SystemView packet containing 2 U32 parameter payload.
SEGGER_SYSVIEW_RecordU32x3()	Formats and sends a SystemView packet containing 3 U32 parameter payload.
SEGGER_SYSVIEW_RecordU32x4()	Formats and sends a SystemView packet containing 4 U32 parameter payload.
SEGGER_SYSVIEW_RecordU32x5()	Formats and sends a SystemView packet containing 5 U32 parameter payload.
SEGGER_SYSVIEW_RecordU32x6()	Formats and sends a SystemView packet containing 6 U32 parameter payload.
SEGGER_SYSVIEW_RecordU32x7()	Formats and sends a SystemView packet containing 7 U32 parameter payload.
SEGGER_SYSVIEW_RecordU32x8()	Formats and sends a SystemView packet containing 8 U32 parameter payload.
SEGGER_SYSVIEW_RecordU32x9()	Formats and sends a SystemView packet containing 9 U32 parameter payload.
SEGGER_SYSVIEW_RecordU32x10()	Formats and sends a SystemView packet containing 10 U32 parameter payload.
SEGGER_SYSVIEW_RecordString()	Formats and sends a SystemView packet containing a string.
API Function Return	
SEGGER_SYSVIEW_RecordEndCall()	Format and send an End API Call event without return value.
SEGGER_SYSVIEW_RecordEndCallU32()	Format and send an End API Call event with return value.

7.8.1 SEGGER_SYSVIEW_RecordVoid()

描述

格式化并发送一个带有空负载的SystemView数据包。

原型

```
void SEGGER_SYSVIEW_RecordVoid(unsigned int EventID);
```

参数

Parameter	Description
EventID	SystemView event ID.

7.8.2 SEGGER_SYSVIEW_RecordU32()

描述

格式化并发送一个包含单个U32参数负载的SystemView数据包。

原型

无效SEGGER_SYSVIEW_RecordU32(unsigned int EventID,
U32值);

参数

Parameter	Description
EventID	SystemView event ID.
Value	The 32-bit parameter encoded to SystemView packet payload.

7.8.3 SEGGER_SYSVIEW_RecordU32x2 ()

描述

格式化并发送包含2个U32参数负载的SystemView报文。

原型

```
void SEGGER_SYSVIEW_RecordU32x2(unsigned int EventID,
                                U32 para a0, U32 para a1);
```

参数

Parameter	Description
EventID	SystemView event ID.
Para0	The 32-bit parameter encoded to SystemView packet payload.
Para1	The 32-bit parameter encoded to SystemView packet payload.

7.8.4 SEGGER_SYSVIEW_RecordU32x3()

描述

格式化并发送一个包含3个U32参数负载的SystemView报文。

原型

```
无效SEGGER_SYSVIEW_RecordU32x3(unsigned int EventID,
                               U32 Para0, U32 Para1, U32
                               Para2);
```

参数

Parameter	Description
EventID	SystemView event ID.
Para0	The 32-bit parameter encoded to SystemView packet payload.
Para1	The 32-bit parameter encoded to SystemView packet payload.
Para2	The 32-bit parameter encoded to SystemView packet payload.

7.8.5 SEGGER_SYSVIEW_RecordU32x4()

描述

格式化并发送一个包含4个U32参数负载的SystemView报文。

原型

```
无效SEGGER_SYSVIEW_RecordU32x4(unsigned int EventID,
                               U32          a0,
                               U32          虽然这些,
                               U32          Para2,
                               Par          Para3);
```

参数

Parameter	Description
EventID	SystemView event ID.
Para0	The 32-bit parameter encoded to SystemView packet payload.
Para1	The 32-bit parameter encoded to SystemView packet payload.
Para2	The 32-bit parameter encoded to SystemView packet payload.
Para3	The 32-bit parameter encoded to SystemView packet payload.

7.8.6 SEGGER_SYSVIEW_RecordU32x5()

描述

格式化并发送包含5个U32参数负载的SystemView报文。

原型

```
void SEGGER_SYSVIEW_RecordU32x5(unsigned int EventID)
                                U32  a0,
                                U32  虽然这些,
                                U32  Para2,
                                U32  Para3,
                                Par  Para4);
```

参数

Parameter	Description
EventID	SystemView event ID.
Para0	The 32-bit parameter encoded to SystemView packet payload.
Para1	The 32-bit parameter encoded to SystemView packet payload.
Para2	The 32-bit parameter encoded to SystemView packet payload.
Para3	The 32-bit parameter encoded to SystemView packet payload.
Para4	The 32-bit parameter encoded to SystemView packet payload.

7.8.7 SEGGER_SYSVIEW_RecordU32x6()

描述

格式化并发送一个包含6个U32参数负载的SystemView报文。

原型

```
无效SEGGER_SYSVIEW_RecordU32x6(unsigned int EventID,
                               U32          Para0,
                               U32          Para1,
                               U32          Para2,
                               U32          Para3,
                               U32          Para4,
                               U32          Para5);
```

参数

Parameter	Description
EventID	SystemView event ID.
Para0	The 32-bit parameter encoded to SystemView packet payload.
Para1	The 32-bit parameter encoded to SystemView packet payload.
Para2	The 32-bit parameter encoded to SystemView packet payload.
Para3	The 32-bit parameter encoded to SystemView packet payload.
Para4	The 32-bit parameter encoded to SystemView packet payload.
Para5	The 32-bit parameter encoded to SystemView packet payload.

7.8.8 SEGGER_SYSVIEW_RecordU32x7()

描述

格式化并发送一个包含7个U32参数负载的SystemView报文。

原型

```
无效SEGGER_SYSVIEW_RecordU32x7(unsigned int EventID,
                               U32          Para0,
                               U32          Para1,
                               U32          Para2,
                               U32          Para3,
                               U32          Para4,
                               U32          Para5,
                               U32          Para6);
```

参数

Parameter	Description
EventID	SystemView event ID.
Para0	The 32-bit parameter encoded to SystemView packet payload.
Para1	The 32-bit parameter encoded to SystemView packet payload.
Para2	The 32-bit parameter encoded to SystemView packet payload.
Para3	The 32-bit parameter encoded to SystemView packet payload.
Para4	The 32-bit parameter encoded to SystemView packet payload.
Para5	The 32-bit parameter encoded to SystemView packet payload.
Para6	The 32-bit parameter encoded to SystemView packet payload.

7.8.9 SEGGER_SYSVIEW_RecordU32x8 ()

描述

格式化并发送一个包含8个U32参数负载的SystemView报文。

原型

(无符号int EventID,

```
U32          Para0,
U32          Para1,
U32          Para2,
U32          Para3,
U32          Para4,
U32          Para5,
U32          Para6,
U32          Para7);
```

参数

Parameter	Description
EventID	SystemView event ID.
Para0	The 32-bit parameter encoded to SystemView packet payload.
Para1	The 32-bit parameter encoded to SystemView packet payload.
Para2	The 32-bit parameter encoded to SystemView packet payload.
Para3	The 32-bit parameter encoded to SystemView packet payload.
Para4	The 32-bit parameter encoded to SystemView packet payload.
Para5	The 32-bit parameter encoded to SystemView packet payload.
Para6	The 32-bit parameter encoded to SystemView packet payload.
Para7	The 32-bit parameter encoded to SystemView packet payload.

7.8.10 SEGGER_SYSVIEW_RecordU32x9()

描述

格式化并发送一个包含9个U32参数负载的SystemView报文。

原型

```
无效SEGGER_SYSVIEW_RecordU32x9(unsigned int EventID,
                               U32          a0,
                               U32          虽然这些,
                               U32          Para2,
                               U32          Para3,
                               U32          Para4,
                               U32          运输,
                               U32          Para6,
                               U32          Para7,
                               Par          Para8);
```

参数

Parameter	Description
EventID	SystemView event ID.
Para0	The 32-bit parameter encoded to SystemView packet payload.
Para1	The 32-bit parameter encoded to SystemView packet payload.
Para2	The 32-bit parameter encoded to SystemView packet payload.
Para3	The 32-bit parameter encoded to SystemView packet payload.
Para4	The 32-bit parameter encoded to SystemView packet payload.
Para5	The 32-bit parameter encoded to SystemView packet payload.
Para6	The 32-bit parameter encoded to SystemView packet payload.
Para7	The 32-bit parameter encoded to SystemView packet payload.
Para8	The 32-bit parameter encoded to SystemView packet payload.

7.8.11 SEGGER_SYSVIEW_RecordU32x10 ()

描述

格式化并发送包含10个U32参数负载的SystemView报文。

原型

```
void SEGGER_SYSVIEW_RecordU32x10(unsigned int EventID)
                                 U32          Para0,
                                 U32          Para1,
                                 U32          Para2,
                                 U32          Para3,
                                 U32          Para4,
                                 U32          Para5,
                                 U32          Para6,
                                 U32          Para7,
                                 U32          Para8,
                                 U32          Para9);
```

参数

Parameter	Description
EventID	SystemView event ID.
Para0	The 32-bit parameter encoded to SystemView packet payload.
Para1	The 32-bit parameter encoded to SystemView packet payload.
Para2	The 32-bit parameter encoded to SystemView packet payload.
Para3	The 32-bit parameter encoded to SystemView packet payload.
Para4	The 32-bit parameter encoded to SystemView packet payload.
Para5	The 32-bit parameter encoded to SystemView packet payload.
Para6	The 32-bit parameter encoded to SystemView packet payload.
Para7	The 32-bit parameter encoded to SystemView packet payload.
Para8	The 32-bit parameter encoded to SystemView packet payload.
Para9	The 32-bit parameter encoded to SystemView packet payload.

7.8.12 SEGGER_SYSVIEW_RecordString()

描述

格式化并发送一个包含字符串的SystemView数据包。

原型

```
无效SEGGER_SYSVIEW_RecordString(unsigned int EventID,
                                const char * pString);
```

参数

Parameter	Description
EventID	SystemView event ID.
pString	The string to be sent in the SystemView packet payload.

额外的信息

该字符串被编码为后跟字符串内容的计数字节。不超过SEGGER_SYSVIEW_MAX_STRING_LEN字节将被编码到有效负载。

7.8.13 SEGGER_SYSVIEW_RecordEndCall()

描述

格式化并发送一个没有返回值的End API Call事件。

原型

```
void SEGGER_SYSVIEW_RecordEndCall(unsigned int EventID);
```

参数

Parameter	Description
EventID	Id of API function which ends.

7.8.14 SEGGER_SYSVIEW_RecordEndCallU32 ()

描述

格式化并发送一个带有返回值的End API Call事件。

原型

```
void SEGGER_SYSVIEW_RecordEndCallU32(unsigned int EventID,
                                     U32 para);
```

参数

Parameter	Description
EventID	Id of API function which ends.
Para0	Return value which will be returned by the API function.

7.9 低级事件编码功能

事件记录构建功能。

Function	Description
SEGGER_SYSVIEW_EncodeU32()	Encode a U32 in variable-length format.
SEGGER_SYSVIEW_EncodeData()	Encode a byte buffer in variable-length format.
SEGGER_SYSVIEW_EncodeString()	Encode a string in variable-length format.
SEGGER_SYSVIEW_EncodeId()	Encode a 32-bit Id in shrunken variable-length format.
SEGGER_SYSVIEW_ShrinkId()	Get the shrunken value of an Id for further processing like in SEGGER_SYSVIEW_NameResource().

7.9.1 SEGGER_SYSVIEW_EncodeU32()

描述

以可变长度格式编码U32。

原型

U8 *SEGGER_SYSVIEW_EncodeU32;
U32值);

参数

Parameter	Description
pPayload	Pointer to where U32 will be encoded.
Value	The 32-bit value to be encoded.

返回值

指向值后面字节的指针，即负载中的第一个空闲字节和下一个存储负载内容的位置。

7.9.2 SEGGER_SYSVIEW_EncodeData ()

描述

以可变长度格式编码字节缓冲区。

原型

与* SEGGER_SYSVIEW_EncodeData (与 * pPayload,
const char 实腹式预应力型钢混凝土,*
无符号整型 NumBytes);

参数

Parameter	Description
pPayload	Pointer to where string will be encoded.
pSrc	Pointer to data buffer to be encoded.
NumBytes	Number of bytes in the buffer to be encoded.

返回值

指向值后面字节的指针，即负载中的第一个空闲字节和下一个存储负载内容的位置。

额外的信息

数据被编码为后跟数据缓冲区内容的计数字节。确保NumBytes + 1字节对于有效负载是空闲的。

7.9.3 SEGGER_SYSVIEW_EncodeString ()

描述

以可变长度格式编码字符串。

原型

与* SEGGER_SYSVIEW_EncodeString (与 * pPayload,
const char *年代,
无符号整型 MaxLen);

参数

Parameter	Description
pPayload	Pointer to where string will be encoded.
s	String to encode.
MaxLen	Maximum number of characters to encode from string.

返回值

指向值后面字节的指针，即负载中的第一个空闲字节和下一个存储负载内容的位置。

额外的信息

该字符串被编码为后跟字符串内容的计数字节。不超过1 + MaxLen字节将被编码为有效载荷。

7.9.4 SEGGER_SYSVIEW_EncodeId()

描述

以缩小的可变长度格式编码32位Id。

原型

U8 *SEGGER_SYSVIEW_EncodeId;
U32 Id);

参数

Parameter	Description
pPayload	Pointer to where the Id will be encoded.
Id	The 32-bit value to be encoded.

返回值

指向值后面字节的指针，即负载中的第一个空闲字节和下一个存储负载内容的位置。

额外的信息

收缩Id的参数可以在SEGGER_SYSVIEW_Conf.h和SEGGER_SYSVIEW_SetRAMBase()中配置。SEGGER_SYSVIEW_ID_BASE:应用程序报告的最低Id。(即0x20000000当所有Id都是这个RAM中的一个地址时)SEG-GER_SYSVIEW_ID_SHIFT:为了节省带宽而移动Id的位数。(当Id为4字节对齐时为2)

7.9.5 SEGGER_SYSVIEW_ShrinkId ()

描述

获取Id的缩小值，以便像segger_sysview_name -source()那样进行进一步处理。

原型

U32 segger_sysview_shrinkkid (U32 Id);

参数

Parameter	Description
Id	The 32-bit value to be shrunken.

返回值

萎缩的Id。

额外的信息

收缩Id的参数可以在SEGGER_SYSVIEW_Conf.h和SEGGER_SYSVIEW_SetRAMBase()中配置。SEGGER_SYSVIEW_ID_BASE:应用程序报告的最低Id。(即0x20000000当所有Id都是这个RAM中的一个地址时)SEG-GER_SYSVIEW_ID_SHIFT:为了节省带宽而移动Id的位数。(当Id为4字节对齐时为2)

7.9.6 SEGGER_SYSVIEW_SendPacket()

描述
发送事件报文。

原型

```
int SEGGER_SYSVIEW_SendPacket
                                  与
                                  无符号整型
                                                  * pPacket,
                                                  * pPayloadEnd, EventId);
```

参数

Parameter	Description
pPacket	Pointer to the start of the packet.
pPayloadEnd	Pointer to the end of the payload. Make sure there are at least 5 bytes free after the payload.
EventId	Id of the event packet.

返回值
≠0 Success，消息已发送。=0 Buffer已满，Message *NOT* sent。

7.10 内部函数

应用程序提供的函数。

Function	Description
SEGGER_SYSVIEW_Conf()	Initialize and configures SystemView.
SEGGER_SYSVIEW_X_GetTimestamp()	Callback called by SystemView to get the timestamp in cycles.

7.10.1 SEGGER_SYSVIEW_Conf()

描述

可以与OS集成一起使用，以便更容易地初始化系统视图和OS系统视图接口。

此功能通常在所使用OS的SEGGER_SYSVIEW_Config_<OS>.c配置文件中提供。

原型

```
void SEGGER_SYSVIEW_Conf(void);
```

示例实现

```
void SEGGER_SYSVIEW_Conf(void) {
  //
  // Initialize SystemView
  //
  SEGGER_SYSVIEW_Init(SYSVIEW_TIMESTAMP_FREQ,   // Frequency of the timestamp.
                      SYSVIEW_CPU_FREQ,         // Frequency of the system.
                      &SYSVIEW_X_OS_TraceAPI,
  // OS-specific SEGGER_SYSVIEW_OS_API
                      _cbSendSystemDesc
  // Callback for application-specific description
                     );
  SEGGER_SYSVIEW_SetRAMBase(SYSVIEW_RAM_BASE);
  // Explicitly set the RAM base address.
  OS_SetTraceAPI(&embOS_TraceAPI_SYSVIEW);
  // Configure embOS to use SystemView via the Trace-API.
}
```


7.10.2 SEGGER_SYSVIEW_X_GetTimestamp ()

描述

该函数必须在配置SEGGER_SYSVIEW_GET_TIMESTAMP()调用时实现。默认情况下，这是在所有非cortex - m3 /4目标上完成的。

原型

```
U32 SEGGER_SYSVIEW_X_GetTimestamp(无效);
```

返回值

以时间戳周期返回当前系统时间戳。在Cortex-M3和Cortex-M4上，这是周期计数器。

示例实现

```
U32 SEGGER_SYSVIEW_X_GetTimestamp(void) {
  U32 TickCount;
  U32 Cycles;
  U32 CyclesPerTick;
  //
  // Get the cycles of the current system tick.
  // SysTick is down-counting, subtract the current value from the number of cycles per tick.
  //
  CyclesPerTick = SYST_RVR + 1;
  Cycles = (CyclesPerTick - SYST_CVR);
  //
  // Get the system tick count.
  //
  TickCount = SEGGER_SYSVIEW_TickCnt; // SEGGER_SYSVIEW_TickCnt is incremented by the system tic
  //
  // If a SysTick interrupt is pending increment the TickCount
  //
  if ((SCB_ICSR & SCB_ICSR_PENDSTSET_MASK) != 0) {
    TickCount++;
  }
  Cycles += TickCount * CyclesPerTick;

  return Cycles;
}
```


第八章

性能和资源使用

介绍SystemView的性能和资源使用情况。它包含了关于典型系统的内存需求的信息，这些信息可以用来获得大多数目标系统的充分估计。

8.1 内存需求

根据所使用的操作系统集成、目标配置和编译器优化，内存需求可能会有所不同。

为了获得性能和内存使用的平衡结果，建议为SystemView和RTT模块设置相应的编译器优化级别。编译器优化应该始终为SystemView和RTT模块打开——即使在调试配置构建中也是如此。

8.1.1 ROM使用情况

下表按组件列出SystemView的ROM使用情况。使用智能链接器，只有使用过的功能才会包含在应用程序中。

Description	ROM
Minimum core code required when using SystemView	~920 Byte
Basic SystemView recording functions for application, OS and module events	~380 Byte
OS-related SystemView recording functions	~360 Byte
Middleware module-related recording functions	~120 Byte
Complete SystemView Module	*~1.8 KByte*

下表列出了不同配置下SystemView的ROM使用情况。

Description	Configuration	ROM
SystemView Module	Balanced optimization, no static buffer	~1.8 KByte
SystemView Module	Balanced optimization, static buffer	~2.1 KByte
SystemView Module	Balanced optimization, no static buffer, post-mortem mode	~1.4 KByte
SystemView Module	Balanced optimization, static buffer, post-mortem mode	~1.7 KByte
RTT Module	Balanced optimization	~0.5 KByte

8.1.2 静态RAM使用情况

下表列出了不同配置下SystemView的静态RAM使用情况。

Description	Configuration	RAM
SystemView Module	No static buffer	~70 Byte + Channel Buffer
SystemView Module	Static buffer	~280 Byte + Channel Buffer
SystemView Module	No static buffer, post-mortem mode	~60 Byte + Channel Buffer
SystemView Module	Static buffer, post-mortem mode	~180 Byte + Channel Buffer
RTT Module		~30 Byte + Channel Buffer

8.1.3 堆栈RAM使用情况

SystemView要求堆栈在每个上下文中记录事件，这可能会记录应用程序中的事件。这通常包括调度器使用的系统堆栈、中断堆栈和任务堆栈。

由于SystemView处理系统描述和任务信息的传入请求，因此堆栈上必须有足够的空闲空间来记录事件并发送正在记录另一个事件的系统描述。

可以将SystemView配置为在较低的堆栈使用或较少的静态RAM使用之间进行选择。

Description	Maximum Stack
Static buffer for event generation and encoding	~230 Bytes
Stack buffer for event generation and encoding	~510 Bytes
Static buffer for event generation and encoding, post-mortem mode	~150 Bytes
Stack buffer for event generation and encoding, post-mortem mode	~280 Bytes

第九章

常见问题

问:*我能使用系统视图应用程序,而我正在调试我的应用程序?*
是的。系统视图可以与调试器并行运行，并进行连续记录。为了确保能够足够快地读取数据,请将调试器连接配置为高接口速度(≥4 MHz)。

问:*我能在没有J-Link的情况下连续录音吗?*
答:不是。连续记录需要J-Link实时传输(RTT)技术自动从目标读取数据。单次射击和死后记录可以用任何调试探头完成。

问:*我我能在Cortex-A、Cortex-R或ARM7、ARM9上连续录音吗?*
答:不是。RTT需要在目标运行时访问目标上的内存。如果你有一个这样的设备，只能做一次记录。

问:*我连续录制时会出现溢出事件。如何防止这种情况发生?*
答:SystemVIew RTT缓冲区满时发生溢出事件。这可能发生在以下原因:

- J-Link被调试器保持忙碌，不能足够快地读取数据。
- 目标接口速度太低，读取数据不够快。
- 应用程序生成的事件太多，无法装入缓冲区。

为了防止这种情况发生:

- 在目标运行时，尽量减少调试器与J-Link的交互。(例如禁用实时手表)
- 在所有连接到J-Link的实例中选择更高的接口速率。(例如调试器和系统视图)
- 为SystemView选择一个更大的缓冲区。(1 - 4kbyte)
- 运行SystemView独立没有调试器。

问:*SystemView找不到RTT控制块,如何配置?*
答:RTT控制块的自动检测只能在初始化后的已知RAM地址范围内完成。开始记录时要确保应用程序启动已经运行。如果RTT控制块在所选设备的已知范围之外，选择“地址”并输入RTT控制块的确切地址，或选择“地址范围”并输入RTT控制块将在其中的地址范围。

Q: *Do I have to select a Target Device to start recording?*

是的。J-Link必须知道连接的是哪个目标设备。下拉列表中列出了最近使用的设备。要选择其他设备，只需输入其名称。支持的设备列表可以在这里找到。

问:*我的问题没有在上面列出。我可以从哪里获得更多的信息?*
答:更多信息和帮助请在SEGGER论坛https://forum.segger.com提问

