

SEGGER

SystemView

User Guide

Document: UM08027
Software Version: 2.10
Revision: 0
Date: November 6, 2015



A product of SEGGER Microcontroller GmbH & Co. KG

www.segger.com

Disclaimer

Specifications written in this document are believed to be accurate, but are not guaranteed to be entirely free of error. The information in this manual is subject to change for functional or performance improvements without notice. Please make sure your manual is the latest edition. While the information herein is assumed to be accurate, SEGGER Microcontroller GmbH & Co. KG (SEGGER) assumes no responsibility for any errors or omissions. SEGGER makes and you receive no warranties or conditions, express, implied, statutory or in any communication with you. SEGGER specifically disclaims any implied warranty of merchantability or fitness for a particular purpose.

Copyright notice

You may not extract portions of this manual or modify the PDF file in any way without the prior written permission of SEGGER. The software described in this document is furnished under a license and may only be used or copied in accordance with the terms of such a license.

© 2015 SEGGER Microcontroller GmbH & Co. KG, Hilden / Germany

Trademarks

Names mentioned in this manual may be trademarks of their respective companies.

Brand and product names are trademarks or registered trademarks of their respective holders.

Contact address

SEGGER Microcontroller GmbH & Co. KG

In den Weiden 11

D-40721 Hilden

Germany

Tel. +49 2103-2878-0

Fax. +49 2103-2878-28

E-mail: support@segger.com

Internet: www.segger.com

Manual versions

This manual describes the current software version. If you find an error in the manual or a problem in the software, please inform us and we will try to assist you as soon as possible. Contact us for further information on topics or functions that are not yet documented.

Print date: November 6, 2015

Software	Revision	Date	By	Description
2.10	0	151106	JL	Official Release.
2.09	0	151026	JL	Initial Pre-Release.

About this document

Assumptions

This document assumes that you already have a solid knowledge of the following:

- The software tools used for building your application (assembler, linker, C compiler).
- The C programming language.
- The target processor.
- DOS command line.

If you feel that your knowledge of C is not sufficient, we recommend *The C Programming Language* by Kernighan and Richie (ISBN 0-13-1103628), which describes the standard in C programming and, in newer editions, also covers the ANSI C standard.

How to use this manual

This manual explains all the functions and macros that the product offers. It assumes you have a working knowledge of the C language. Knowledge of assembly programming is not required.

Typographic conventions for syntax

This manual uses the following typographic conventions:

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.
GUI Element	Buttons, dialog boxes, menu names, menu commands.
Emphasis	Very important sections.

Table of contents

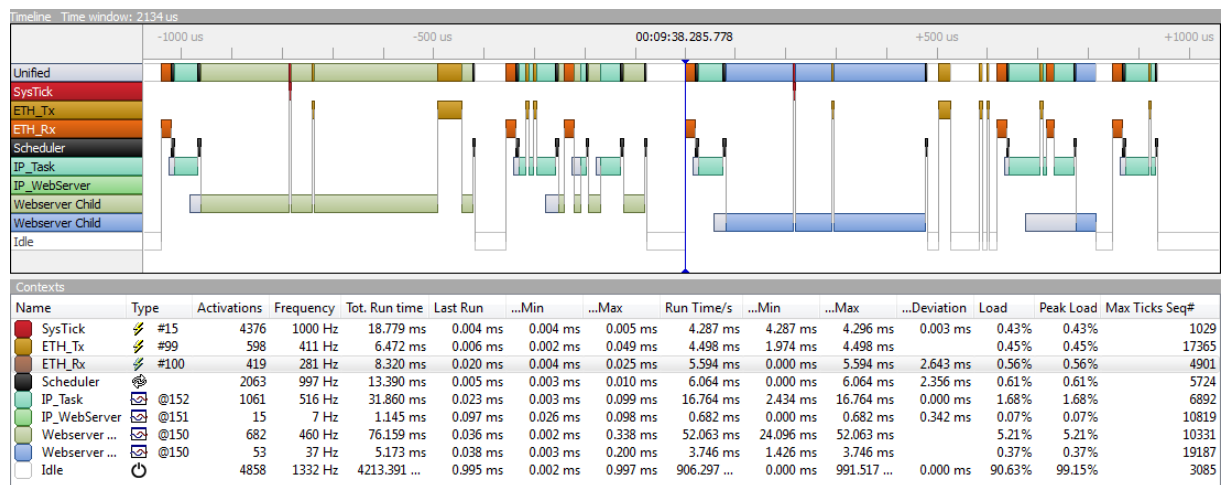
1	Overview	9
1.1	What is SEGGER SystemView?	10
1.2	The SEGGER SystemView package	11
1.2.1	Download and installation	11
1.2.2	Package content	11
2	Getting started with SEGGER SystemView	13
2.1	Starting SystemViewer and loading data	14
2.2	A first look at the system	15
2.3	Analysing system activity	17
2.4	Further analysis of the application core	19
2.4.1	Analysis conclusion	20
3	Host application - SystemViewer	21
3.1	Introduction	22
3.2	Events window	23
3.3	System information	24
3.4	Timeline window	25
3.5	CPU Load window	26
3.6	Contexts window	27
3.7	GUI controls	28
4	Target implementation - SYSVIEW modules	31
4.1	Prerequisites	32
4.1.1	Segger SystemView target implementation modules	32
4.2	Including SEGGER SystemView in the application	33
4.3	The SystemView configuration	34
4.4	SEGGER SystemView API functions	35
4.4.1	SEGGER_SYSVIEW_Init()	36
4.4.2	SEGGER_SYSVIEW_OnUserStart()	37
4.4.3	SEGGER_SYSVIEW_OnUserStop()	38
4.4.4	SEGGER_SYSVIEW_SendSysDesc()	39
4.4.5	SEGGER_SYSVIEW_SetRAMBase()	40
4.5	Configuring SEGGER SystemView	41
4.5.1	SEGGER_SYSVIEW_RTT_BUFFER_SIZE	41
4.5.2	SEGGER_SYSVIEW_RTT_CHANNEL	41
4.5.3	SEGGER_SYSVIEW_GET_TIMESTAMP()	41
4.5.4	SEGGER_SYSVIEW_TIMESTAMP_BITS	41
4.5.5	SEGGER_SYSVIEW_ID_BASE	41

4.5.6	SEGGER_SYSVIEW_ID_SHIFT	41
4.5.7	SEGGER_SYSVIEW_GET_INTERRUPT_ID()	42
4.5.8	SEGGER_SYSVIEW_LOCK()	42
4.5.9	SEGGER_SYSVIEW_UNLOCK()	42

Chapter 1

Overview

This section describes SEGGER SystemView in general.



1.1 What is SEGGER SystemView?

SystemView is a great tool to gain a deep understanding of the runtime behavior of an application, going far beyond what a debugger is offering. This is particularly advantageous when developing and working in complex systems with multiple threads and events.

SystemView consists of two parts:

The PC application *SystemViewer* and some code collecting information on the target system.

The SystemViewer host application allows analysis and profiling of the behavior of an embedded system. It uses SEGGER J-Link and its *Real Time Transfer* (RTT) technology to transfer data describing the behavior of the embedded system. This is not only analyzed and visualized in real time, it can also be saved to a file for analysis at a later time or to simply document the behavior of the system.

The data is transferred via the debug interface, meaning that no additional hardware (and critically, no additional pins) is required to use it and it can be used on any system that can be debugged.

SystemView makes it possible to analyze which interrupts, tasks and software timers have executed, how often, when exactly and how much time they have used. It sheds light on what exactly happened in which order, which interrupt has triggered which task switch, which interrupt and task has called which API function of the underlying RTOS.

Cycle-accurate profiling can be performed and even user functionality can be timed.

SystemView should be used to verify that the embedded system behaves as expected and can be used to find problems and inefficiencies, such as superfluous and spurious interrupts, and unexpected task changes. It can be used with any RTOS which is instrumented to call SYSVIEW event functions, but also in systems without an RTOS or without an instrumented RTOS to analyze interrupt execution.

How does it work?

The data transfer uses RTT, which needs a small software module to be included on the target side. The target system calls SYSVIEW functions in certain situations, such as interrupt start and interrupt end, which rely on RTT to store information in a small buffer on the target side. The type of event together with a high-accuracy timestamp is written (via the RTT module) into the RTT buffer. Timestamps can be as accurate as 1 CPU cycle, which equates to 5ns on a 200MHz CPU.

What resources are required on the target side?

The combined ROM size of RTT and the SYSVIEW modules is less than 2 KB. For typical systems, about 600 bytes of RAM are sufficient. No other hardware is required. The CPU needs less than 1µs per event (for a 200 MHz Cortex-M4 CPU), which results in less than 1% overhead in a system with 10,000 events per second. Since the debug interface (JTAG, SWD, FINE, ...) is used to transfer the data, no additional pins are required.

On which CPUs can SystemView be used?

SystemView can be used on any system supported by J-Link RTT technology. RTT requires the ability to reading memory via the debug interface during program execution. This especially includes ARM Cortex-M0, M0+, M1, M3, M4 processors as well as all Renesas RX devices.

How much work is it to add it to a target system?

Not very much. A small number of files need to be added to the make file or project. If the operating system supports SystemView, then only one function needs to be called. In a system without RTOS or non-instrumented RTOS, two lines of code need to be added to every interrupt function which should be monitored. That's all and should not take more than a few minutes.

1.2 The SEGGER SystemView package

The following sections describe how to install the SEGGER SystemView package and its contents.

1.2.1 Download and installation

The SEGGER SystemView package is available as an installer setup. Download the latest setup from <http://www.segger.com> and execute it. The setup wizard guides through the installation.

After installation the package content can be accessed through the Windows *Start* menu or from the file explorer.

1.2.2 Package content

The SEGGER SystemView package includes everything needed for application tracing — the target instrumentation code and the host PC visualization tool.

Additional sources to interface with SEGGER software, such as embOS, and sample trace files are included for a quick and easy start.

The following table lists the package content.

File	Description
SystemViewer.exe	SystemView analysis and visualization tool.
UM08027_SystemView.pdf	This documentation.
Description/ SYSVIEW_embOS.txt	SystemView API description file for SEGGER embOS.
SampleTraces/ OS_IP_WebServer.SVdat	SystemView sample trace file of a web server application.
SampleTraces/ OS_Start_LEDBlink.SVdat	SystemView sample trace file of a simple embOS application.
TargetSrc/Config/ SEGGER_RTT_Conf.h	SEGGER Real Time Transfer (RTT) configuration file.
TargetSrc/Config/ SEGGER_SYSVIEW_Conf.h	SEGGER SystemView configuration file.
TargetSrc/OS/ SEGGER_SYSVIEW_embOS.c	Interface between SystemView and embOS.
TargetSrc/OS/ SEGGER_SYSVIEW_embOS.h	Interface header.
TargetSrc/Sample/Config/ SEGGER_SYSVIEW_Config.h	Sample initialization header.
TargetSrc/Sample/Config/ SEGGER_SYSVIEW_Config_embOS.c	Sample initialization of SystemView with embOS.
TargetSrc/Sample/Config/ SEGGER_SYSVIEW_Config_None.c	Sample initialization of SystemView with no OS.
TargetSrc/SEGGER/RTT.c	SEGGER RTT module source.
TargetSrc/SEGGER/RTT.h	SEGGER RTT module header.
TargetSrc/SEGGER/ SEGGER_SYSVIEW.c	SEGGER SystemView module source.
TargetSrc/SEGGER/ SEGGER_SYSVIEW.h	SEGGER SystemView module header.
TargetSrc/SEGGER/ SEGGER_SYSVIEW_ConfDefault.h	SEGGER SystemView configuration fallback.

File	Description
TargetSrc/SEGGER/ SEGGER_SYSVIEW_Int.h	SEGGER SystemView internal header.

Chapter 2

Getting started with SEGGER SystemView

This section describes how to get started with SEGGER SystemView. It explains how to analyze an application based on monitored data.

This chapter refers to the sample data file `OS_IP_WebServer.SVDat` which is part of the SEGGER SystemView package.

The sample data file shows the behavior of a target system running the embOS RTOS, the embOS/IP TCP/IP stack and a web server application.

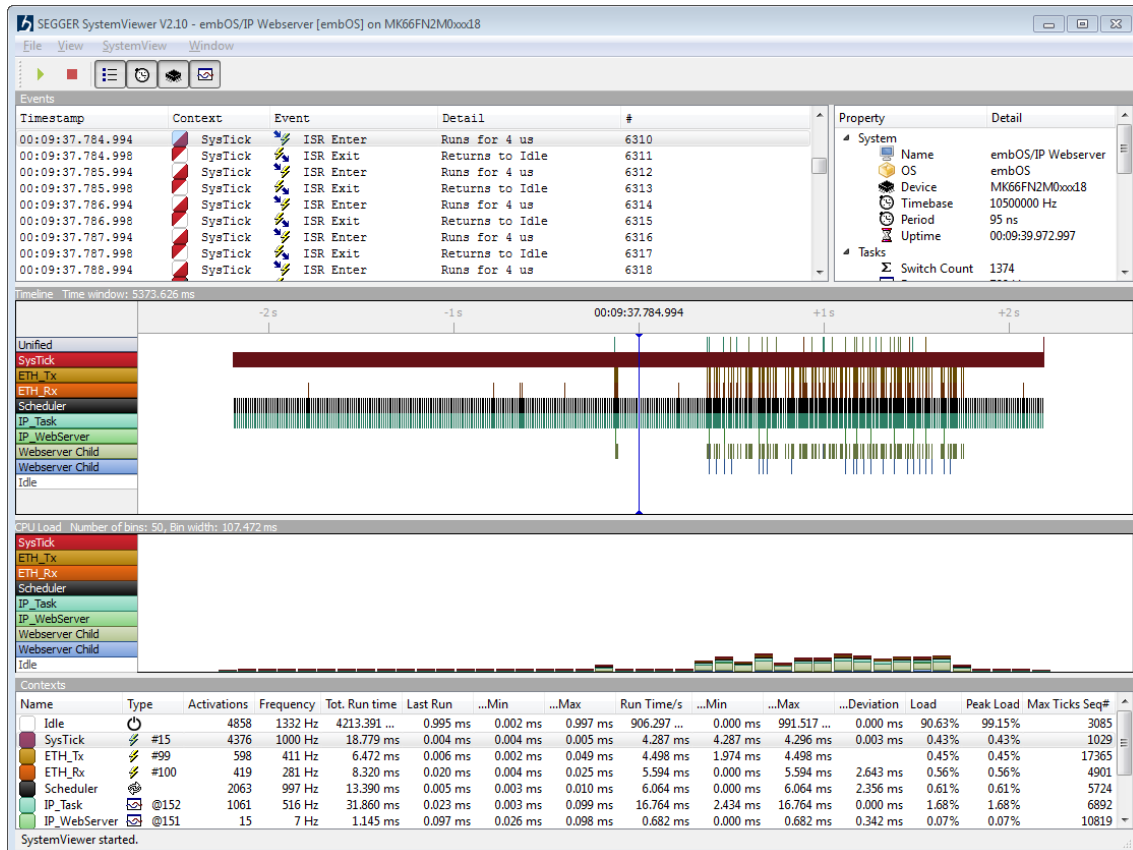
We are going to analyze what the application is doing with the information from SEGGER SystemView.

2.1 Starting SystemViewer and loading data

SystemViewer can monitor data live from the target application. The monitored data can be saved to a file for later work with it. Saved data can be analyzed without a J-Link and even without the target hardware or the target application. This allows analysis of the system by developers who do not have physical access to it.

- Start SystemViewer (SystemViewer.exe) from the Windows *Start* menu or the installation directory.
- Select File → Load Data (Keyboard shortcut: F3).
- Choose OS_IP_WebServer.SVdat from \$PackageInstallationDir\$/SampleTraces/ and click Open.

SystemViewer loads and analyzes the data and should now look like this:



2.2 A first look at the system

We will take a first look at the data to get some information about the monitored system.

The *Timeline* window shows the complete monitored data. In the *Events* list, scroll to the first item to get started.

System information

The window on the top right displays information about the system. This information has been sent by the application, therefore SystemViewer does not require any setup configuration.

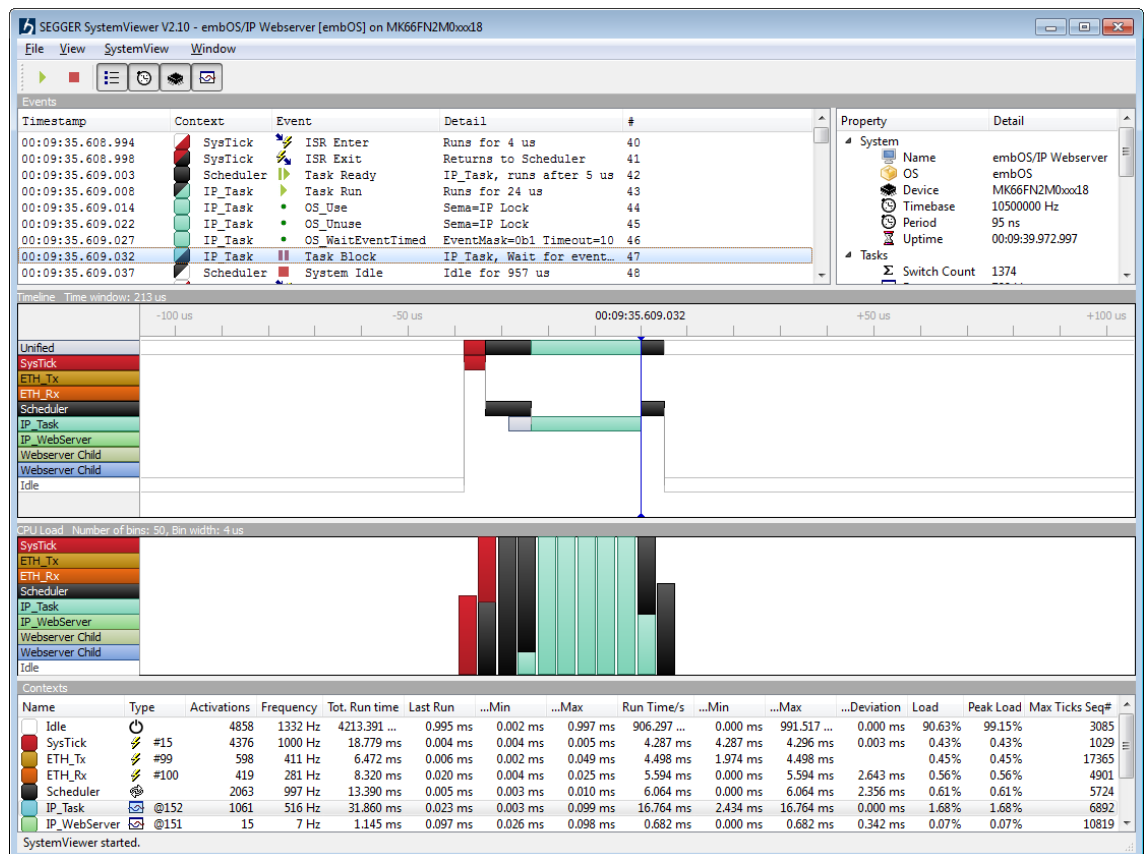
The System information includes the application name, the used OS, the target device, and the timing information. Additional information about task switches and interrupt frequency provide a quick overview of the system.

Timeline

The *Timeline* window shows the system activity by context (task, interrupt, scheduler and idle) over the system time. Each row refers to one context item and we can see all items which have been used in the application while it has been monitored. In the initial view we can see system activity every millisecond.

Click on the vertical line below '+1 ms' to center the item and zoom in to see the active contexts. (Use [Ctrl] + the mouse wheel, [Ctrl] + [left] and [right] keys or View→Zoom In, View→Zoom Out to zoom.) We see that the activity every millisecond is the 1 kHz SysTick interrupt.

Jump to the next interrupt with View→Forward (Keyboard shortcut: F) until we see more activity than the SysTick interrupt. Zoom in or out to see all activity on the screen. We can see the SysTick interrupt returned to the OS Scheduler, which makes the IP_Task ready, indicated by the grey bar in the IP_Task's row, and lets it run. When the IP_Task has done its current job it goes back to idle.



Conclusion

We have got some information about the monitored system, the application name, the target device and the system frequency. From the Timeline we know which tasks and interrupts are used by the application, that it is controlled by the 1 kHz SysTick interrupt, and the IP_Task is activated at least every 10 ms.

2.3 Analysing system activity

After getting some information of the system we will analyze what the system does when it is active.

Events list

The Events list shows all events as they are sent from the system and displays their information, including timestamp of the event, active context, type of event and event details.

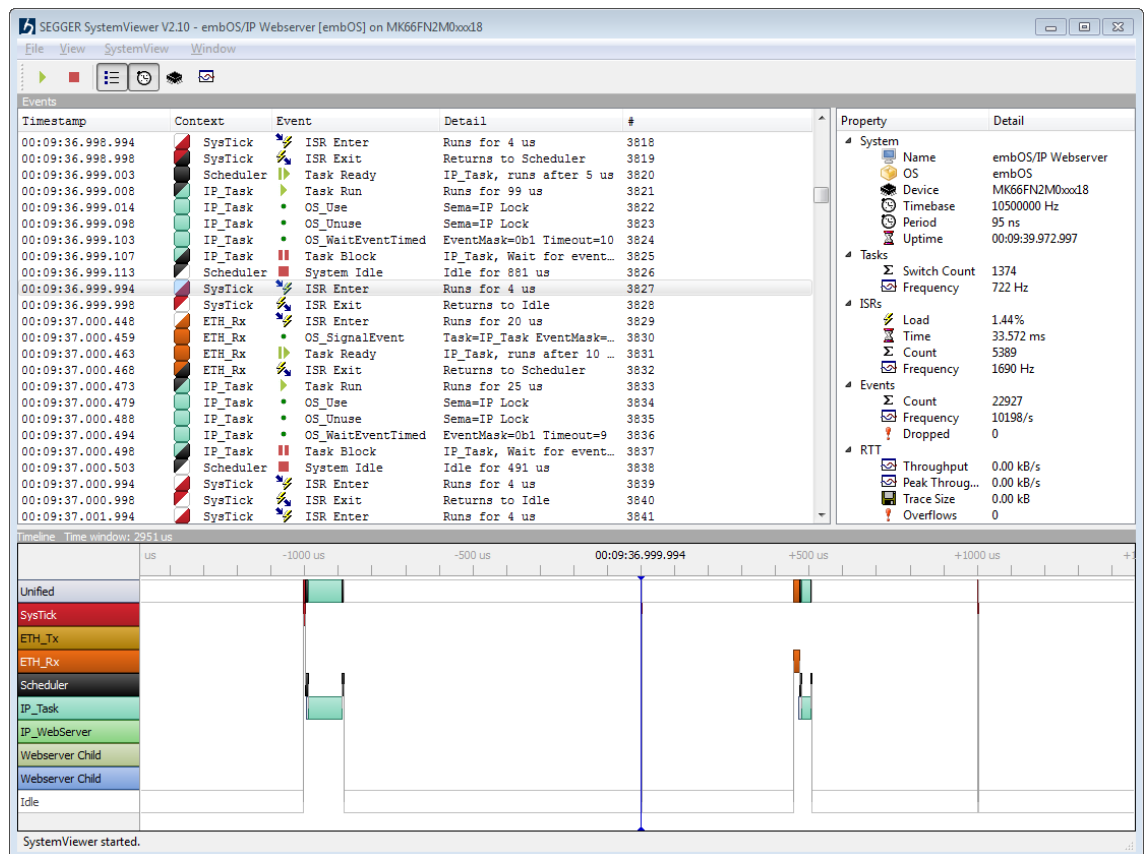
We have seen that every millisecond the SysTick ISR enters and exits and that it activates the IP_Task every 10 ms because its timeout occurred.

Go to event #478 with View→Go to Event... (Keyboard shortcut: Ctrl+G). It is a call of OS_WaitEventTimed with a timeout out 10 ms from the IP_Task at 00:01:32.759.006. The timeout would happen at 00:01:32.769.006.

Set a time reference on the event with View→Events→Toggle Reference (Keyboard shortcut R). All following timestamps in the events list are measured from the latest reference.

To now see whether the IP_Task runs because of the timeout or because of the event, move the mouse over the label of the IP_Task row in the timeline and click on the right arrow button to jump to the next activity of IP_Task. Alternatively select the Task Run event (#475) and select View→Next Sibling (Keyboard shortcut: Shift+N) to jump to the next Task Run event of IP_Task.

The timestamp is 00:00:00.003.703, clearly before the timeout, so the task was activated by the event it waited for. We can see the ETH_Rx interrupt happened before and called OS_SignalEvent, which marked the task as ready as indicated in the timeline. The ETH_Rx interrupt returns to the Scheduler and IP_Task runs. We have received something via Ethernet.



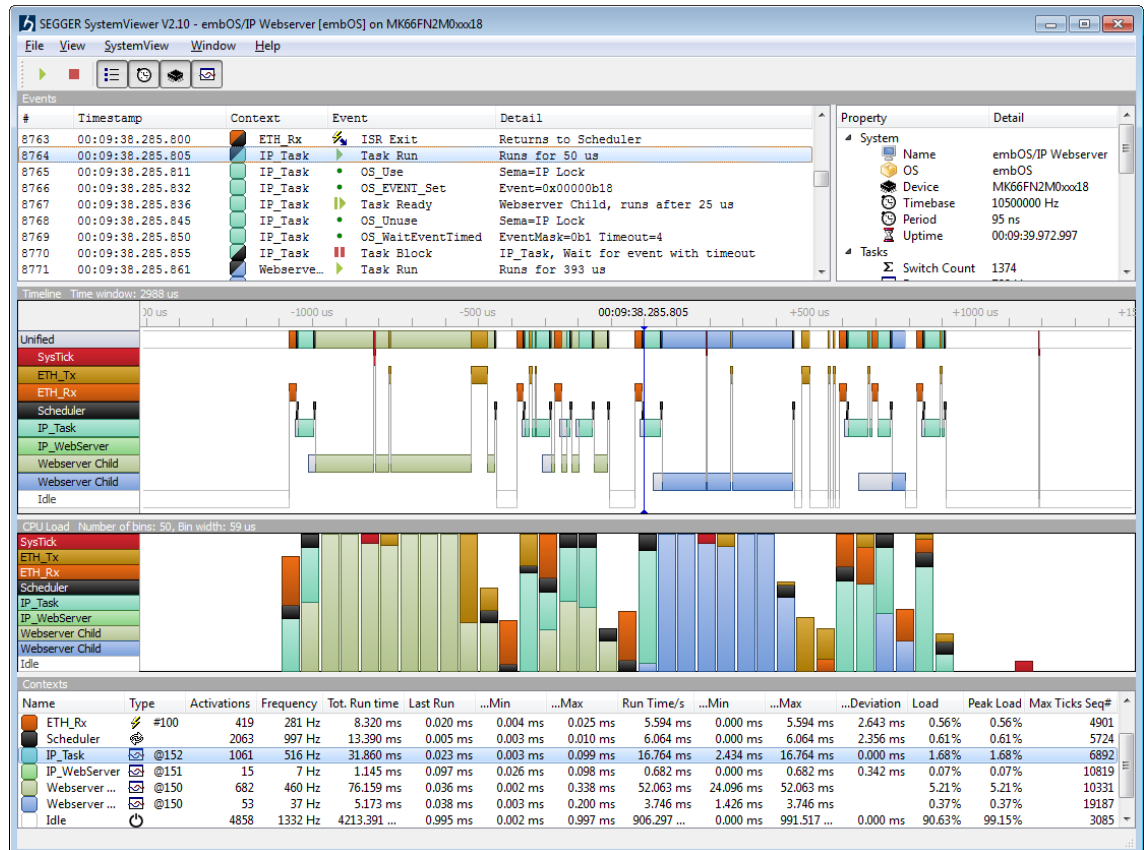
Conclusion

Going further through the events, we can see that the IP_Task is activated after the 10 ms timeout occurred or after we received something and the ETH_Rx interrupt occurred.

2.4 Further analysis of the application core

We now know that the system is mainly controlled by the ETH_Rx interrupt. The next step is to see what the system does when it is active.

The application creates a web server which can be accessed by the browser to view the embOS/IP demo web page. The sample data has been gathered while the web server was running and the browser loaded the web page multiple times.



Timeline, Events list and Contexts window

The windows of SystemViewer are synchronized and provide the best possibilities for system analysis when used together.

Search through the Events list to find some activity in the system. Go to event #809.

Here we can see that the ETH_Rx interrupt activates the IP_Task, calls OS_EVENT_Set and makes the Webserver Child task ready. After the IP_Task has done its work the Scheduler activates Webserver Child.

The Webserver Child is interrupted by another ETH_Rx interrupt which causes a preemptive task switch to the IP_Task in the Scheduler, because the IP_Task has a higher priority than the Webserver Child. After IP_Task has run, the Scheduler goes back to the Webserver Child task.

Note: Tasks are ordered by priority in the Timeline, the exact task priority can be seen in the Contexts window.

The Webserver Child runs and handles the received request. It sends back data, which can be seen by the following ETH_Tx interrupts. After it has done its work it waits for an event and goes to idle.

While recording this trace we have now seen the web page in the browser.

2.4.1 Analysis conclusion

We analyzed what a system does without insight into the application code. With the application source we can check with SEGGER SystemView that the system does what it is supposed to do.

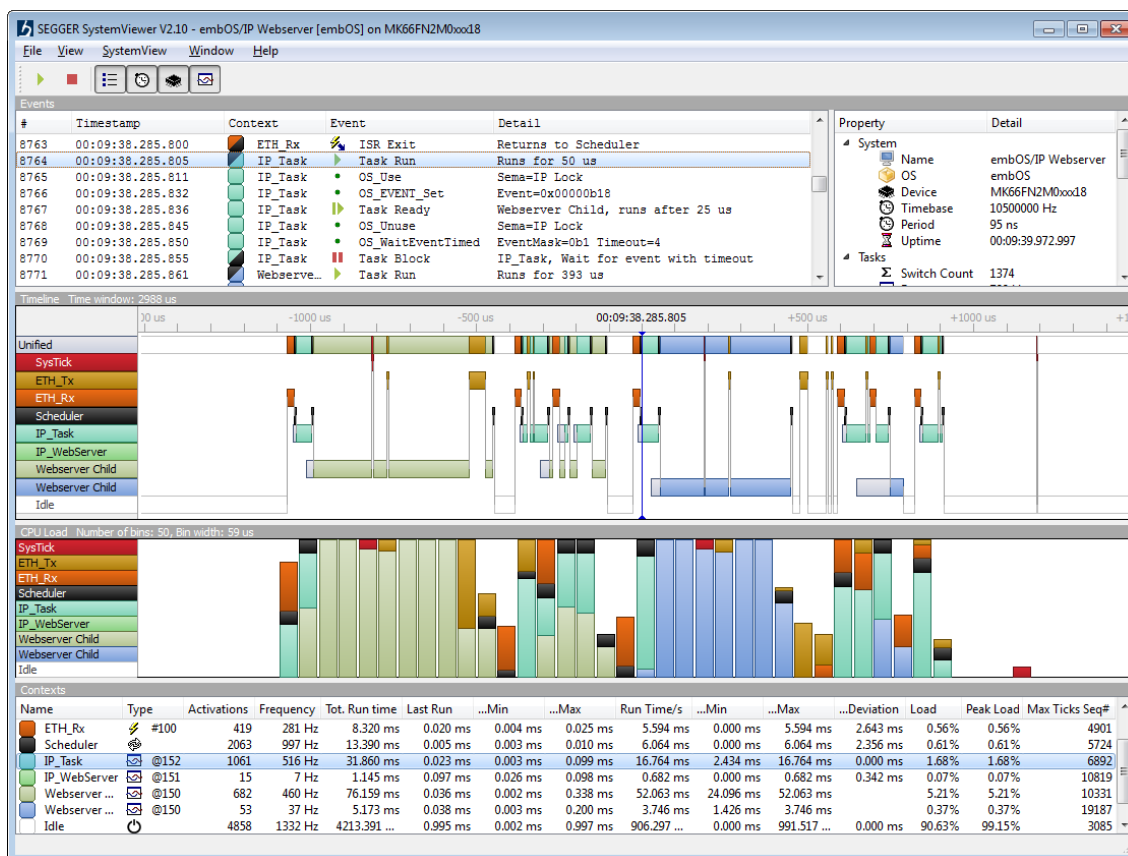
SEGGER SystemView can actively help developing applications, since it not only shows what the system does, but also allows exact time measurement and visualizes the influence of interrupts and events on the application flow. This provides advanced possibilities to find problems and to improve the system.

Chapter 3

Host application - SystemViewer

This section describes SystemViewer, the SystemView analysis and visualization tool.

3.1 Introduction



SystemViewer is the host PC visualization tool for SEGGER SystemView. It connects via a J-Link to the target application, controls the application tracing and reads its data. The monitored data is analyzed on runtime and visualized in the different windows of SystemViewer. After tracing has stopped, the data can be saved to a file which allows later analysis of the application trace.

To get started with SystemViewer please refer to the previous chapter.

SystemViewer provides different windows to visualize the execution in the system, measure timing and analyze the CPU load. All windows are synchronized to always get all information of the currently selected state.

For a description of the application windows please refer to the following sections.

SystemViewer allows going through the monitored data and keeping track of what happened in the system at any time.

3.2 Events window

Timestamp	Context	Event	Detail	#
00:09:38.285.800	ETH Rx	ISR Exit	Returns to Scheduler	8763
00:09:38.285.805	IP_Task	Task Run	Runs for 50 us	8764
00:09:38.285.811	IP_Task	OS_Use	Sema=IP Lock	8765
00:09:38.285.832	IP_Task	OS_EVENT_Set	Event=0x00000b18	8766
00:09:38.285.836	IP_Task	Task Ready	Webserver Child, runs a...	8767
00:09:38.285.845	IP_Task	OS_Unuse	Sema=IP Lock	8768
00:09:38.285.850	IP_Task	OS_WaitEventTimed	EventMask=0b1 Timeout=4	8769
00:09:38.285.855	IP_Task	Task Block	IP_Task, Wait for event...	8770
00:09:38.285.861	Webserve...	Task Run	Runs for 393 us	8771




















The Events window shows all events as they are sent by the system and displays their information. Every event has the following items:

- A timestamp in target time, which can be displayed with microsecond or nanosecond resolution.
- A context from which it has been created, i.e. the task which is running.
- An event description, displayed with the type of event (ISR enter and exit, task activity, API call).
- Event details describing the parameters of the event, i.e. the API call parameters.
- An ID to locate events in the list.

The Events window allows going through the list, jumping to the next or previous context, or to the next or previous similar event. The Timeline and CPU Load windows can be synchronized to show the currently selected event.

The timestamp in the events list can be relative to the start of recording or the target system time if it has been sent by the system. Events can be set as time reference for following events to allow easy measurement of when an event occurred after another one.

3.3 System information

Property	Detail
System	
 Name	embOS/IP Webserver
 OS	embOS
 Device	MK66FN2M0xxx18
 Timebase	10500000 Hz
 Period	95 ns
 Uptime	00:09:39.972.997
Tasks	
 Switch Count	1374
 Frequency	722 Hz
ISRs	
 Load	1.44 %
 Time	33.572 ms
 Count	5389
 Frequency	1690 Hz
Events	
 Count	22927
 Frequency	10198/s
 Dropped	0
RTT	
 Throughput	0.00 kB/s
 Peak Throug...	0.00 kB/s
 Trace Size	0.00 kB
 Overflows	0

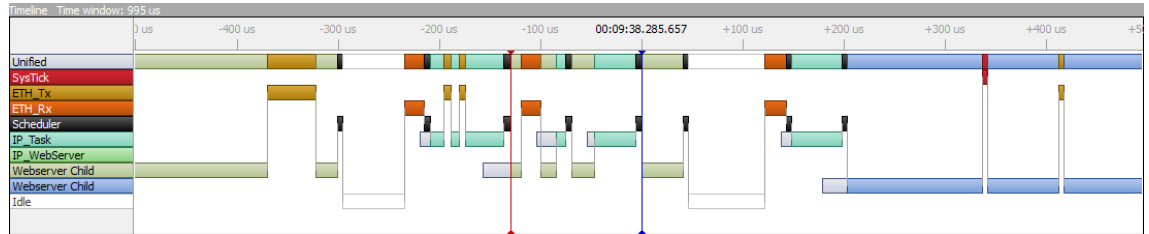
The window on the top right displays:

- Information about the system, which has been sent by the application to identify it.
- Statistics about tasks, interrupts, SystemView events and recording throughput.
- Record properties, which can be set by the user.

The System information includes the application name, the used OS, the target device, and the timing information. Additional information about task switches and interrupt frequency provides a quick overview of the system.

The record properties can be set by the user to be stored with the record when it is saved and allow identifying a record when it is loaded for later analysis.

3.4 Timeline window



The Timeline window shows the system activity by context (task, interrupt, scheduler, timer and idle) over the system time. Each row refers to one context item to show all context items which have been used in the application while it has been monitored.

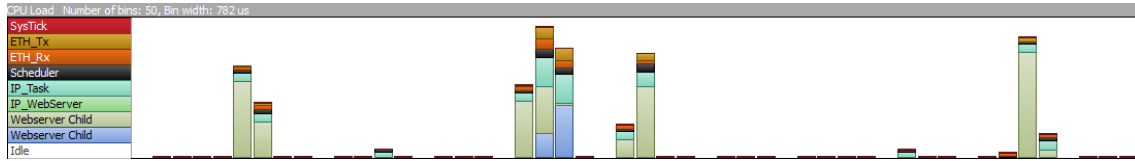
Switches between contexts are displayed as connection lines to easily identify which events cause context switches and when they occurred. It also indicates when tasks are marked ready before they start to run. Contexts are ordered by priority. It starts with interrupts on top ordered by Id. Followed by the Scheduler and software timers, if they are used in the system. Below the Scheduler (and timer) the tasks are ordered from high priority to low priority. The bottom context displays idle time, when no other context is active.

The Timeline can be synchronized with the Events list. The event under the cursor (the blue line) is the selected event in the list. The cursor can be fixed at 10%, 50%, and 90% of the window and update the selection in the list when scrolling through the timeline, or it is fixed on the selected event and moves with the timeline while scrolling. The cursor can be set on a context item in the timeline to select the corresponding event in the events list and vice-versa.

To get an overview of the whole system or to see the exact duration of an event the Timeline view can be zoomed in or out.

To jump to the next or previous activity of a context, the labels include left and right buttons on mouse-over.

3.5 CPU Load window



The CPU Load window displays the used CPU time of contexts for a period. The CPU Load is measured over the width of a bin with the current resolution of the Timeline and is therefore synchronized with the zoom level.

The number of bins can be selected to measure the load over a shorter or longer period. With a single bin the CPU load is measured over the whole visible Timeline.

3.6 Contexts window

Contexts														
Name	Type	Activations	Frequency	Tot. Run time	Last Run	...Min	...Max	Run Time/s	...Min	...Max	...Deviation	Load	Peak Load	Max Ticks Seq#
ETH_Rx	#100	419	281 Hz	8.320 ms	0.020 ms	0.004 ms	0.025 ms	5.594 ms	0.000 ms	5.594 ms	2.643 ms	0.56%	0.56%	4901
ETH_Tx	#99	598	411 Hz	6.472 ms	0.006 ms	0.002 ms	0.049 ms	4.498 ms	1.974 ms	4.498 ms		0.45%	0.45%	17365
SysTick	#15	4376	1000 Hz	18.779 ms	0.004 ms	0.004 ms	0.005 ms	4.287 ms	4.287 ms	4.296 ms	0.003 ms	0.43%	0.43%	1029
Idle		4858	1332 Hz	4213.391 ...	0.995 ms	0.002 ms	0.997 ms	906.297 ...	0.000 ms	991.517 ...	0.000 ms	90.63%	99.15%	3085
Scheduler		2063	997 Hz	13.390 ms	0.005 ms	0.003 ms	0.010 ms	6.064 ms	0.000 ms	6.064 ms	2.356 ms	0.61%	0.61%	5724
IP_Task	@152	1061	516 Hz	31.860 ms	0.023 ms	0.003 ms	0.099 ms	16.764 ms	2.434 ms	16.764 ms	0.000 ms	1.68%	1.68%	6892
IP_WebServer	@151	15	7 Hz	1.145 ms	0.097 ms	0.026 ms	0.098 ms	0.682 ms	0.000 ms	0.682 ms	0.342 ms	0.07%	0.07%	10819

The *Contexts* window shows statistical information of the contexts (Tasks, Interrupts, Scheduler, and Idle). Each context item can be identified by its Name and Type. The Type includes the priority for tasks and the ID for interrupts. (i.e. The Cortex-M SysTick is interrupt ID #15.)

The Contexts window information include following items:

- Activation count of the context.
- Activation frequency.
- The total run-time.
- The last, minimum and maximum run time.
- The average, minimum and maximum run time per second.
- The current CPU load within the last second.
- The peak CPU load over one second.
- The maximum number of ticks within one run.

The Contexts window is updated while the recording, the current context is indicated by selection of the row.

3.7 GUI controls

SystemViewer can be controlled with mouse and keyboard, and via menus. The most important controls are also accessible in the toolbar.

The following table describes the controls of SystemViewer.

Action	Menu	Shortcut
Monitoring data		
Start monitoring the system.	SystemView → Start Recording	F5
Stop monitoring.	SystemView → Stop Recording	F9
Save monitored data to a file.	File → Save Data	F2
Load a monitor data file.	File → Load Data	F3
View/Edit recording properties.	File → Recording Properties...	Ctrl+Shift+R
Events		
Display timestamps in application system time.	View → Display System Time	None
Display timestamps in time since record start.	View → Display Record Time	None
Set the timestamp precision to 1 us.	View → Precision: 1 us	None
Set the timestamp precision to 1 ns.	View → Precision: 1 ns	None
Show/Hide API calls in the events list.	View → Show APIs	Shift+A
Show/Hide ISR Enter/Exit in the events list.	View → Show ISRs	Shift+I
Show/Hide Task activity in events list.	View → Show Tasks	Shift+T
Open dialog to go to an event by Id.	View → Go to Event...	Ctrl+G
Open dialog to go to an event by timestamp.	View → Go to Timestamp...	Ctrl+Shift+G
Set/clear the current event as time reference.	View → Events → Toggle Reference	R, Double-click
Remove all time references.	View → Events → Clear References	Ctrl+Shift+R
Jump to the next context switch.	View → Events → Forward	F
Jump to the previous context switch.	View → Events → Back	B
Jump to the next similar event.	View → Events → Next	N
Jump to the previous similar event.	View → Events → Previous	P
Jump to the next similar event with the same context.	View → Events → Next Sibling	Shift+N
Jump to the previous similar event with the same context.	View → Events → Previous Sibling	Shift+P
Toggle automatic scroll to last item on new events.	View → Events → Auto Scroll	None
Timeline		
Zoom in.	View → Timeline → Zoom In	Ctrl++, Ctrl+Scroll up

Action	Menu	Shortcut
Zoom out.	View→Timeline→Zoom Out	Ctrl+-, Ctrl+Scroll down
Scroll forward.	View→Timeline→Scroll Forward	Left, Scroll up
Scroll back.	View→Timeline→Scroll Back	Right, Scroll down
Show 1 ms across the timeline.	View→Timeline→1ms Window	1
Show 5 ms across the timeline.	View→Timeline→5ms Window	5
Show 10 ms across the timeline.	View→Timeline→10ms Window	0
CPU Load		
Measure the visible CPU load in a single bin.	View→CPU Load→Single Bin	None
Measure the visible CPU load in 10 bins.	View→CPU Load→10 Bins	None
Measure the visible CPU load in 50 bins.	View→CPU Load→50 Bins	None
Measure the visible CPU load in 100 bins.	View→CPU Load→100 Bins	None
Measure the visible CPU load in 200 bins.	View→CPU Load→200 Bins	None
Windows		
Show/hide the Events window.	Window→Events View	E
Show/hide the Timeline window.	Window→Time View	T
Show/hide the CPU Load window.	Window→CPU Load View	L
Show/hide the Contexts window.	Window→Context View	C
Open application preferences dialog.	Window→Preferences	Alt+,
Help		
Open this SystemView Manual.	Help→SystemView Manual	F11
Show SystemViewer information.	Help→About SystemViewer	F12

Chapter 4

Target implementation - SYSVIEW modules

This section describes the instrumentation of a target application to use SEGGER SystemView.

4.1 Prerequisites

To use SEGGER SystemView the following components are required:

- The SEGGER SystemView package, available at <https://www.segger.com>.
- An ARM Cortex-M or Renesas RX target.
- A J-Link Debug Probe.
- A target application project.

For an easy start it is recommended to use SEGGER embOS V4.12 or later, which already includes the SystemView integration.

4.1.1 Segger SystemView target implementation modules

The following files are part of the SEGGER SystemView target implementation. We recommend to copy all files into the application project and keep the given directory structure.

File	Description
/Config/SEGGER_RTT_Conf.h	SEGGER Real Time Transfer (RTT) configuration file.
/Config/ SEGGER_SYSVIEW_Conf.h	SEGGER SystemView configuration file.
/OS/SEGGER_SYSVIEW_embOS.c	Interface between SystemView and embOS.
/OS/SEGGER_SYSVIEW_embOS.h	Interface header.
/SEGGER/RTT.c	SEGGER RTT module source.
/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_ConfDefault.h	SEGGER SystemView configuration fallback.
/SEGGER/SEGGER_SYSVIEW_Int.h	SEGGER SystemView internal header.

4.2 Including SEGGER SystemView in the application

To use SEGGER SystemView, the target implementation modules must be added to the target application. Copy the source modules from the SEGGER SystemView package to the application source and include `SEGGER_SYSVIEW.c` and `SEGGER_RTT.c` in the project.

All information about the target system is provided to SystemView by the application.

The included file `SEGGER_SYSVIEW_Config_embOS.c` can be used as a quick start to set up the SystemView configuration in conjunction with embOS. `SEGGER_SYSVIEW_Config_None.c` is a generic setup when not using an OS. Include one of the files in the project and call `SEGGER_SYSVIEW_Conf()` in the main function.

```

/*
   Include and compile
   SEGGER_SYSVIEW_Config_embOS.c, SEGGER_SYSVIEW.c, and SEGGER_RTT.c
   in your project.
*/
#include "SEGGER_SYSVIEW_Config.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;
}

```

The generic part of SEGGER SystemView is now ready to monitor the application.

When using embOS V4.12 or later with profiling enabled, SystemView events for ISRs, Task, and API calls are generated. When not using embOS, appropriate events must be generated by the application.

Download the application to the target and let it run. As long as SystemViewer, the host application, is not connected the application will not generate SystemView events. When SystemViewer is connected it will activate monitoring in the application and read the SystemView events.

4.3 The SystemView configuration

The included files `SEGGER_SYSVIEW_Config_*.c` provide the configuration of SystemView and can in most cases be used without modification.

```

/*****
 *
 *      (c) SEGGER Microcontroller GmbH & Co. KG
 *
 *****/
----- END-OF-HEADER -----

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

extern unsigned int SystemCoreClock;

/*****
 *
 *      Defines, configurable
 *
 *****/
*/
// The application name to be displayed in SystemViewer
#define SYSVIEW_APP_NAME      "Demo Application"

// The target device name
#define SYSVIEW_DEVICE_NAME   "Cortex-M4"

// Frequency of the timestamp. Must match SEGGER_SYSVIEW_Conf.h
#define SYSVIEW_TIMESTAMP_FREQ (SystemCoreClock >> 4)

// System Frequency. SystemCoreClock is used in most CMSIS compatible projects.
#define SYSVIEW_CPU_FREQ      (SystemCoreClock)

// The lowest RAM address used for IDs (pointers)
#define SYSVIEW_RAM_BASE      (0x20000000)

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

/*****
 *
 *      Global functions
 *
 *****/
*/
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);    // Configure embOS to use SYSVIEW.
}

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

```

4.4 SEGGER SystemView API functions

The following functions can be used to implement SEGGER SystemView into an application. Any other functions that are exposed through `SEGGER_SYSVIEW.h` are used for integration of SEGGER SystemView into OSes and middleware modules.

Function	Description
<code>SEGGER_SYSVIEW_Init()</code>	Initializes the SYSVIEW module.
<code>SEGGER_SYSVIEW_OnUserStart()</code>	Send a user event start, such as start of a subroutine for profiling.
<code>SEGGER_SYSVIEW_OnUserStop()</code>	Send a user event stop, such as return of a subroutine for profiling.
<code>SEGGER_SYSVIEW_SendSysDesc()</code>	Send the system description string to the host.
<code>SEGGER_SYSVIEW_SetRAMBase()</code>	Sets the RAM base address, which is subtracted from IDs in order to save bandwidth.

4.4.1 SEGGER_SYSVIEW_Init()

Description

Initializes the SYSVIEW module. Must be called before SystemViewer attaches to the system.

Prototype

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

Parameters

Parameter	Description
SysFreq	Frequency of timestamp, i.e. CPU core clock frequency.
CPUFreq	CPU core clock frequency.
pOSAPI	Pointer to the API structure for OS-specific functions.
pfSendSysDesc	Pointer to SendSysDesc callback function.

Additional information

This function initializes the RTT channel used to transport SEGGER SystemView packets. The channel is assigned the label "SysView" for client software to identify the SystemView channel.

4.4.2 SEGGER_SYSVIEW_OnUserStart()

Description

Send a user event start, such as start of a subroutine for profiling.

Prototype

```
void SEGGER_SYSVIEW_OnUserStart(unsigned UserId);
```

Parameters

Parameter	Description
UserId	User defined ID for the event.

4.4.3 SEGGER_SYSVIEW_OnUserStop()

Description

Send a user event stop, such as return of a subroutine for profiling.

Prototype

```
void SEGGER_SYSVIEW_OnUserStop(unsigned UserId);
```

Parameters

Parameter	Description
UserId	User defined ID for the event.

4.4.4 SEGGER_SYSVIEW_SendSysDesc()

Description

Send the system description string to the host. The system description is used by SysViewer to identify the current application and handle events accordingly.

Prototype

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

Parameters

Parameter	Description
<code>sSysDesc</code>	Pointer to the 0-terminated system description string.

Additional information

One system description string may not exceed 128 characters.

The Following items can be described in a system description string. Each item is identified by its identifier, followed by '=' and the value. Items are separated by ','.

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"

Example strings

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

4.4.5 SEGGER_SYSVIEW_SetRAMBase()

Description

Sets the RAM base address, which is subtracted from IDs in order to save bandwidth.

Prototype

```
void SEGGER_SYSVIEW_SetRAMBase(U32 RAMBaseAddress);
```

Parameters

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

4.5 Configuring SEGGER SystemView

SEGGER SystemView is configurable to match the target device and application. The default configuration is set to match most cases and normally does not require modification.

To configure SystemView edit the macro defines in `SEGGER_SYSVIEW_Conf.h`:

- Select a timestamp source and the timestamp size, i.e. the Cortex-M cycle counter shifted by four, resulting in 28 bits.
- Define `SEGGER_SYSVIEW_GET_INTERRUPT_ID()`, i.e. read the Cortex-M ICSR bits [8:0].
- Optionally define the base ID and a shift for pointers, i.e. base ID as the lowest RAM address and a shift of 2 if all pointers are 4 byte aligned.
- Define the locking functions used when sending SystemView events, i.e. disable all interrupts.

The following sections describe the configuration defines in detail.

4.5.1 SEGGER_SYSVIEW_RTT_BUFFER_SIZE

Number of bytes that SEGGER SystemView uses for the communication buffer.

Default: 2048

4.5.2 SEGGER_SYSVIEW_RTT_CHANNEL

The RTT Channel used for SEGGER SystemView communication. 0: Auto selection

Default: 1

4.5.3 SEGGER_SYSVIEW_GET_TIMESTAMP()

Function macro to retrieve the system timestamp.

Default: `((*(U32 *) (0xE0001004)) >> 4)`

This is the Cortex-M cycle counter with the least significant four bits removed in order to save bandwidth over RTT.

4.5.4 SEGGER_SYSVIEW_TIMESTAMP_BITS

Number of valid bits low-order delivered by clock source. I.e. 28 when the 32-bit cycle counter has four low-order bits removed (as above).

Default: 28

4.5.5 SEGGER_SYSVIEW_ID_BASE

Base to be subtracted from IDs (pointers) in order to save bandwidth. I.e. the lowest RAM address.

Can be overridden by the application via `SEGGER_SYSVIEW_SetRAMBase()` on initialization.

Default: `0x10000000`

4.5.6 SEGGER_SYSVIEW_ID_SHIFT

Number of bits to shift IDs (pointers) in order to save bandwidth. I.e. 2 when all IDs are at least 4 byte aligned.

Default: 2

4.5.7 SEGGER_SYSVIEW_GET_INTERRUPT_ID()

Function macro to get the currently active interrupt ID: I.e. read the Cortex-M ICSR[8:0] = active vector.

Default: `((*(U32 *) (0xE000ED04)) & 0x1FF)`

4.5.8 SEGGER_SYSVIEW_LOCK()

Function macro to (nestable) lock SystemView transfers from being interrupted. I.e. disable interrupts.

Default: disable interrupts on Cortex-M

4.5.9 SEGGER_SYSVIEW_UNLOCK()

Function macro to (nestable) unlock SystemView transfers from being interrupted. I.e. restore previous interrupt state.

Default: restore interrupt state on Cortex-M