

Embedded  
Online  
Conference



[www.embeddedonlineconference.com](http://www.embeddedonlineconference.com)

# Advanced Debugging and Performance Analysis Techniques for Embedded Applications

**Axel** Wolf

# THE SPEAKER

Axel Wolf



➞ **Technical Director @ SEGGER US**

Focus: Development Tools and Software for Embedded Systems

Axel Wolf is Technical Director at SEGGER Microcontroller LLC in the US, where he is responsible for business development, key account management, partner management, as well as technical support. Axel also regularly represents SEGGER at trade shows, conferences, and partner events. He has 25+ years of experience in microcontrollers, embedded software development, and the associated development tools. Before joining SEGGER in January of 2018, Axel served in advanced technical, marketing, and management positions at Renesas Electronics, NXP Semiconductors, Philips Semiconductors, Infineon Technologies, and Siemens Semiconductors. He holds a BSEE from Baden-Wuerttemberg Cooperative State University (DHBW) in Stuttgart, Germany. Axel is located in Milpitas, California. He can be reached at [axel.wolf@segger.com](mailto:axel.wolf@segger.com).

# AGENDA

1

SEGGER Introduction

2

“Basic” vs. “Advanced”  
Embedded Debugging

3

Live Demo:  
Streaming Instruction Trace /  
Real-Time Code Coverage /  
Real-Time Code Profiling

4

Other Advanced Debugging  
Features

5

Live demo: Real-Time  
Recording and Runtime  
Analysis

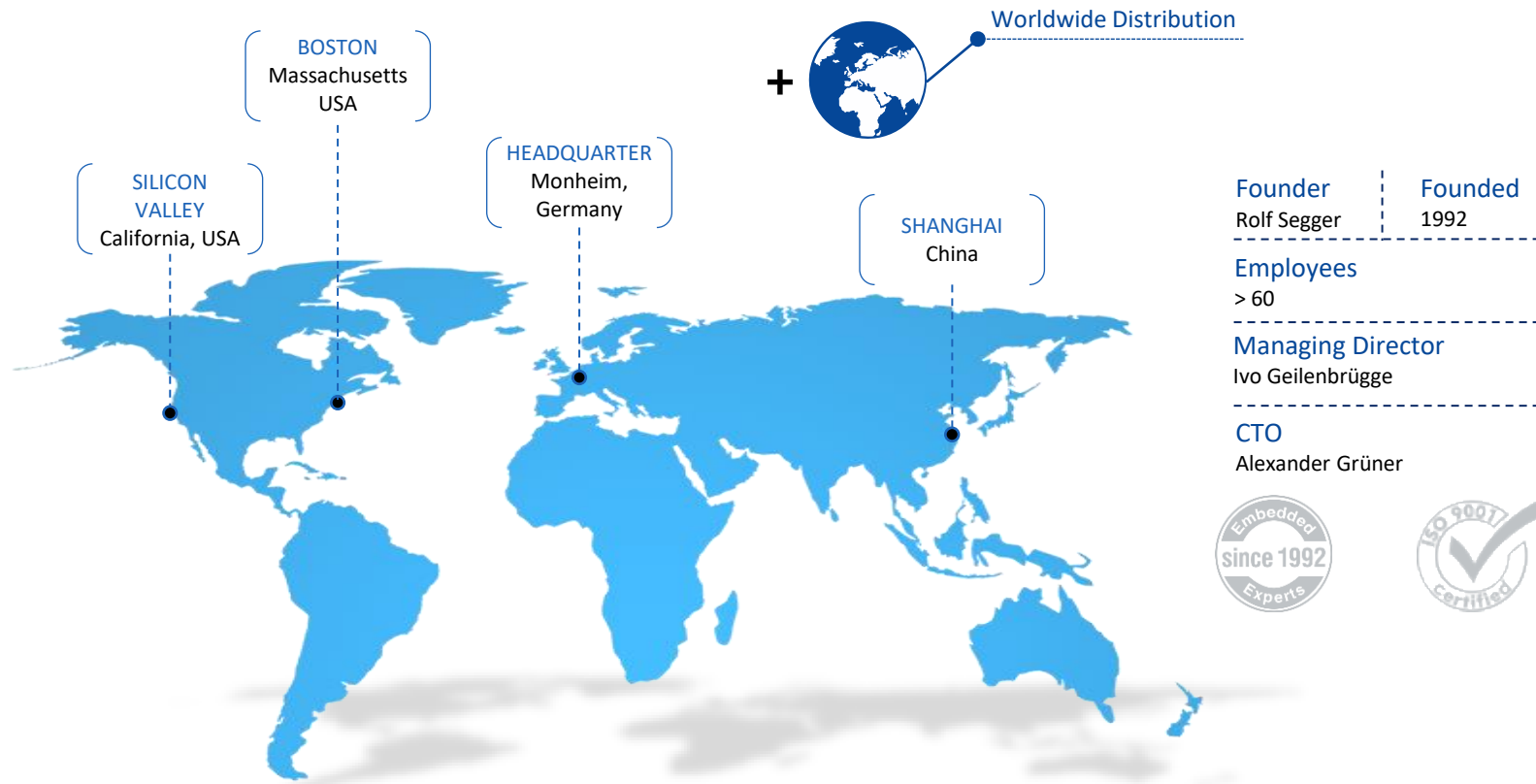
6

Summary



# **SEGGER Introduction**

# SEGGER Company Introduction



## The Embedded Experts

Your One-Stop Shop from **Development** to **Production**

# SEGGER Company Introduction



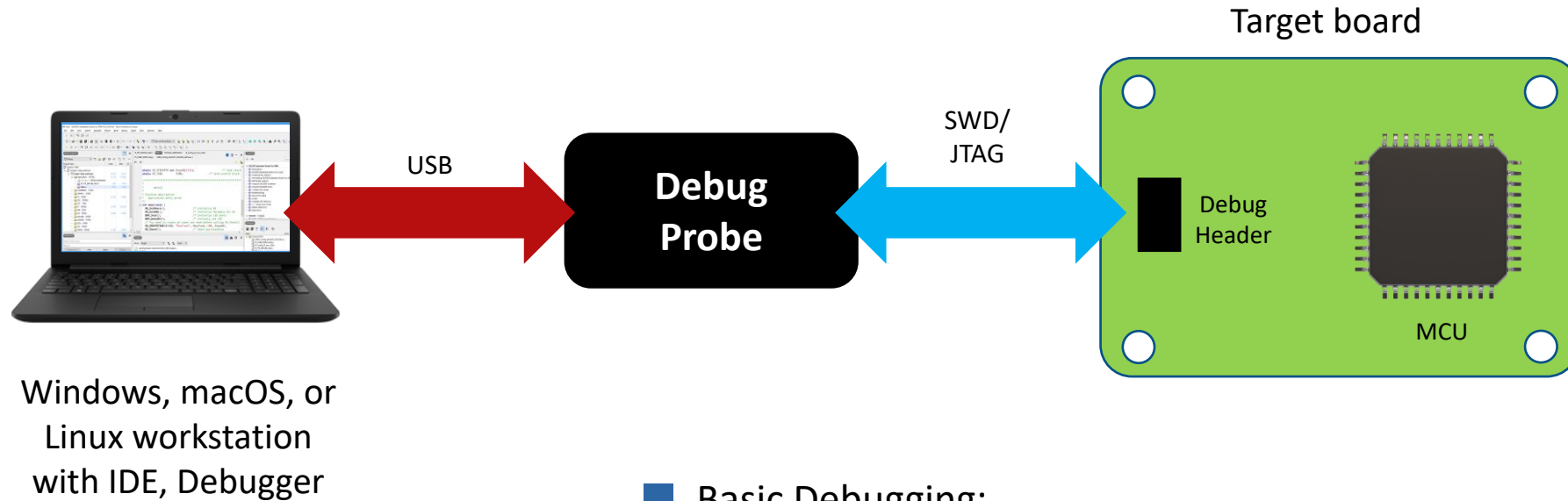
**SEGGER** has over 30 years of experience in Embedded Systems, producing state-of-the-art **middleware**, and offering a full set of hardware tools (for **development** and **production**) and **software tools**.



2

## **“Basic” vs “Advanced” Embedded Debugging**

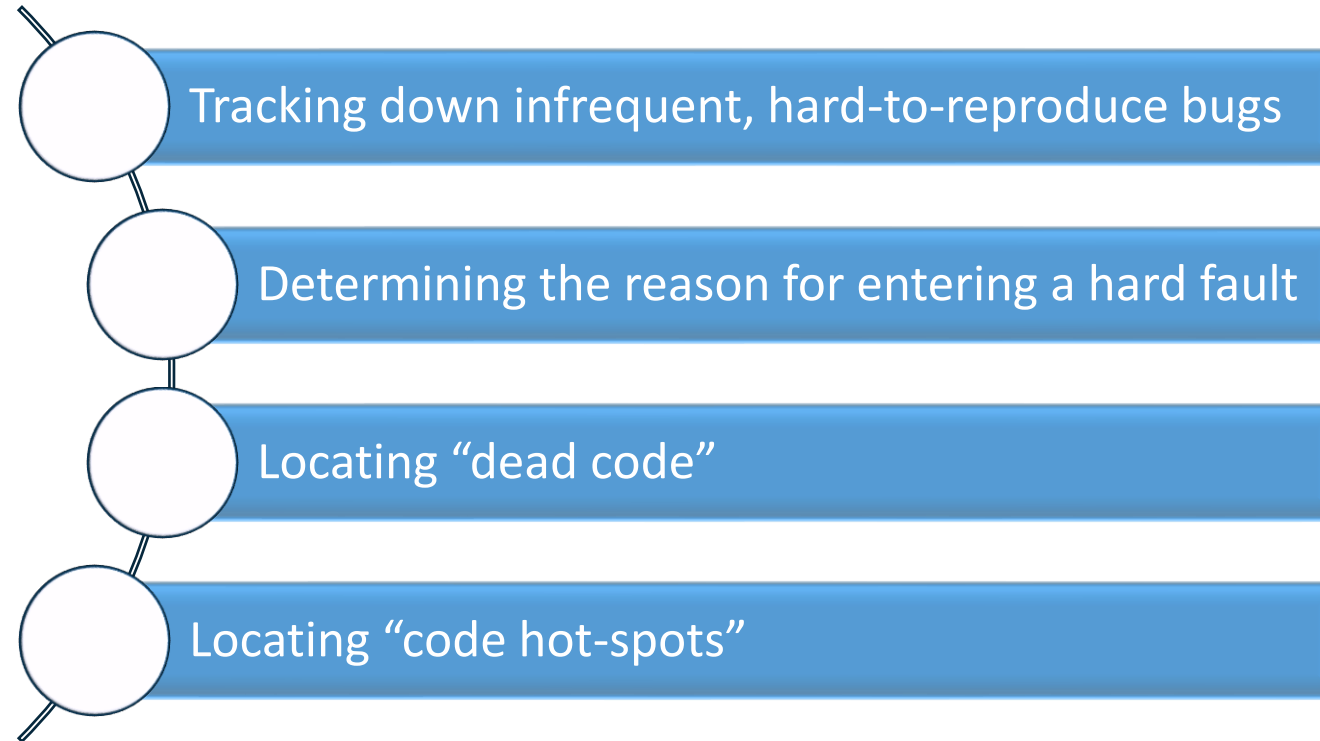
# Basic Debug Setup



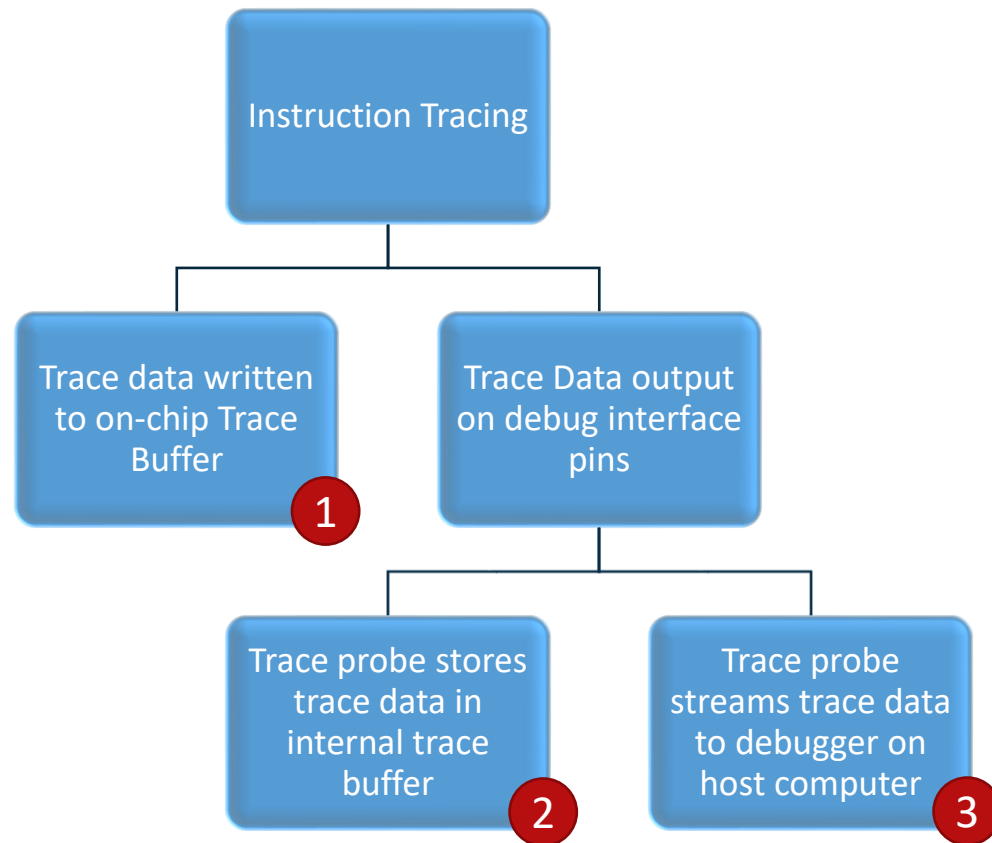
## ■ Basic Debugging:

- Download / Run code
- Halt / Resume program execution
- Single-stepping
- Breakpoints

# When Basic Debugging is not enough...

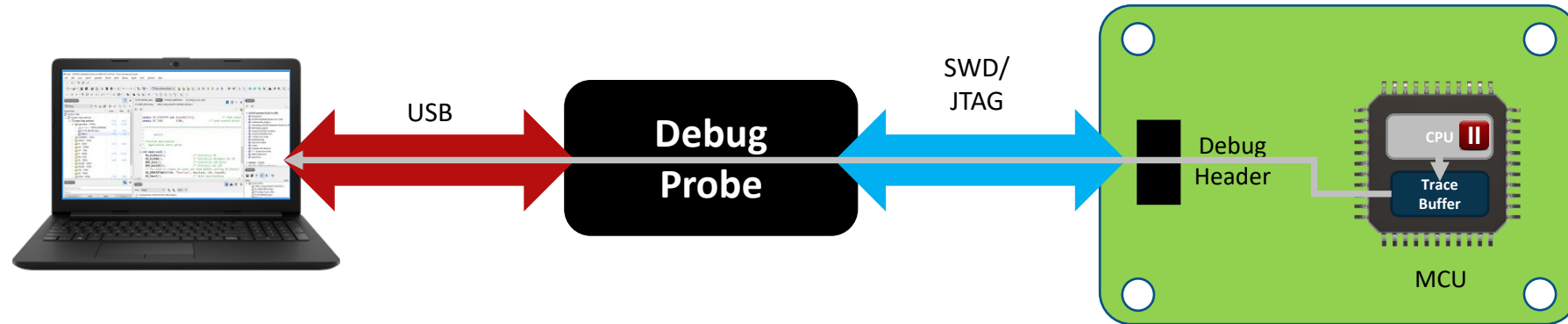


# Key to Advanced Debugging: Instruction Tracing



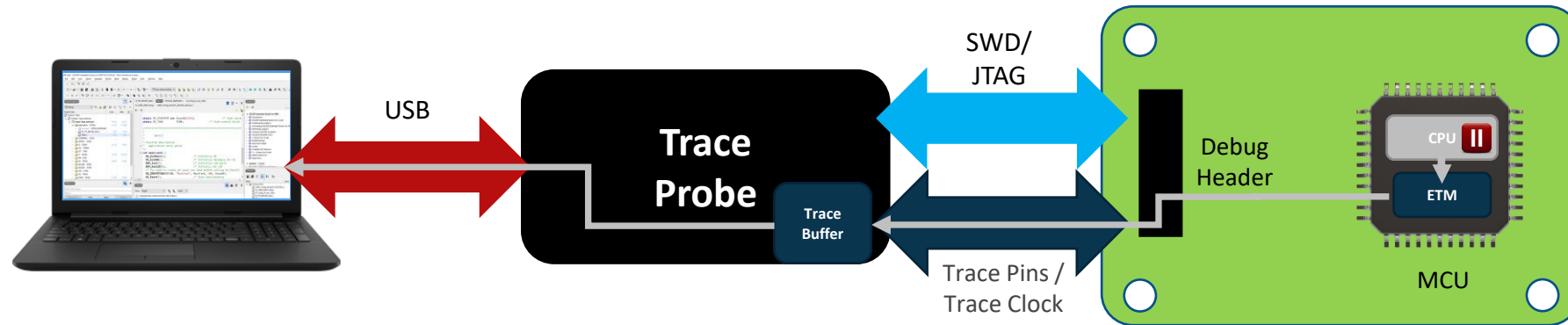
Instruction Trace			
+ _start			15
+ memory_copy			5
+ _start			4
+ memory_copy			3
+ _start			4
+ memory_copy			5
+ _start			4
+ memory_copy			3
+ _start			4
+ memory_copy			3
+ _start			4
+ memory_copy			3
+ _start			4
+ memory_copy			5
+ _start			4
+ memory_set			3
+ _start			4
+ memory_set			3
+ _start			16
- start			4
0800038A	MOV	R0, #0	movs r0, #0
0800038C	MOV	R1, #0	movs r1, #0
0800038E	LDR	R2, [0x08000436]	ldr r2, =APP_EN
08000390	BLX	R2	blx r2

# Instruction Tracing via on-chip Trace Buffer



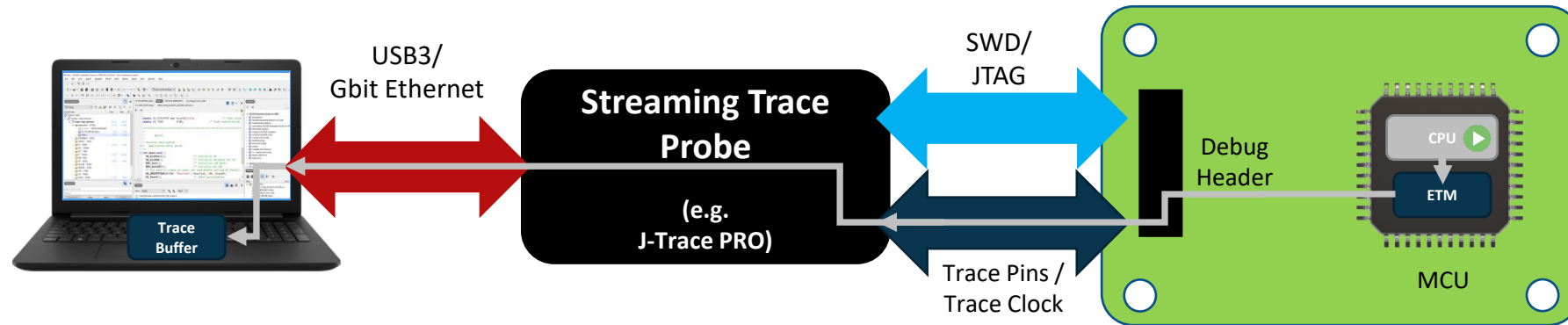
- Examples: Embedded Trace Buffer (ETB), Micro Trace Buffer (MTB), Embedded Trace FIFO (ETF)
- MCU running: Instruction history → on-chip trace buffer
- MCU halted: Trace buffer contents → debugger (via debug probe)
- Drawbacks:
  - Small trace buffer size (kB)
  - Very limited instruction history (ring buffer)

# Instruction Tracing via on-probe Trace Buffer



- MCU outputs trace information on dedicated debug interface pins
  - Embedded Trace Macrocell (ETM) (Cortex-M, Cortex-R)
  - Program Trace Macrocell (PTM) (Cortex-A)
- MCU running: Instruction history → trace probe (stored in trace buffer)
- MCU halted: Trace buffer contents → debugger
- Advantage: Larger trace buffer size (MB to GB)
- Disadvantage: 2 to 5 extra pins required to be routed to the debug interface

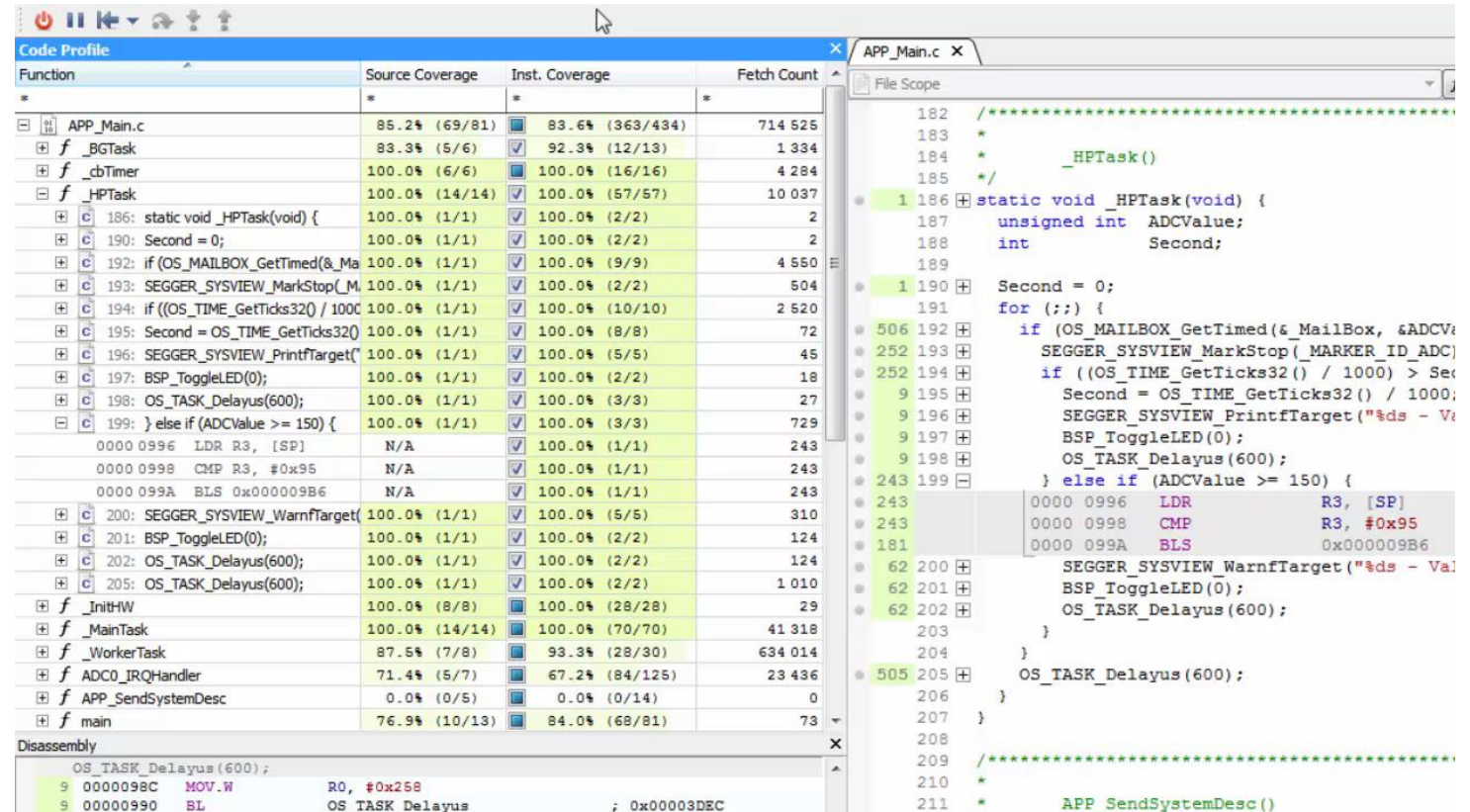
# Instruction Tracing using a Streaming Trace Probe



- Passes instruction trace data to the host computer in real-time
- Streaming into files possible → buffer sizes of TB
- Tracing over extended periods of time possible
- Enables other advanced debugging features
  - Real-Time Code Coverage
  - Real-Time Code Profiling

# Code Coverage: Which parts of the code have been executed?

- Shows how much of a source line, block, function or file has been executed
- Detects code which has not been covered by tests
- Detects unreachable code
- Helps improve the code / create suitable test suite for uncovered blocks
- “Only tested code is finished code”

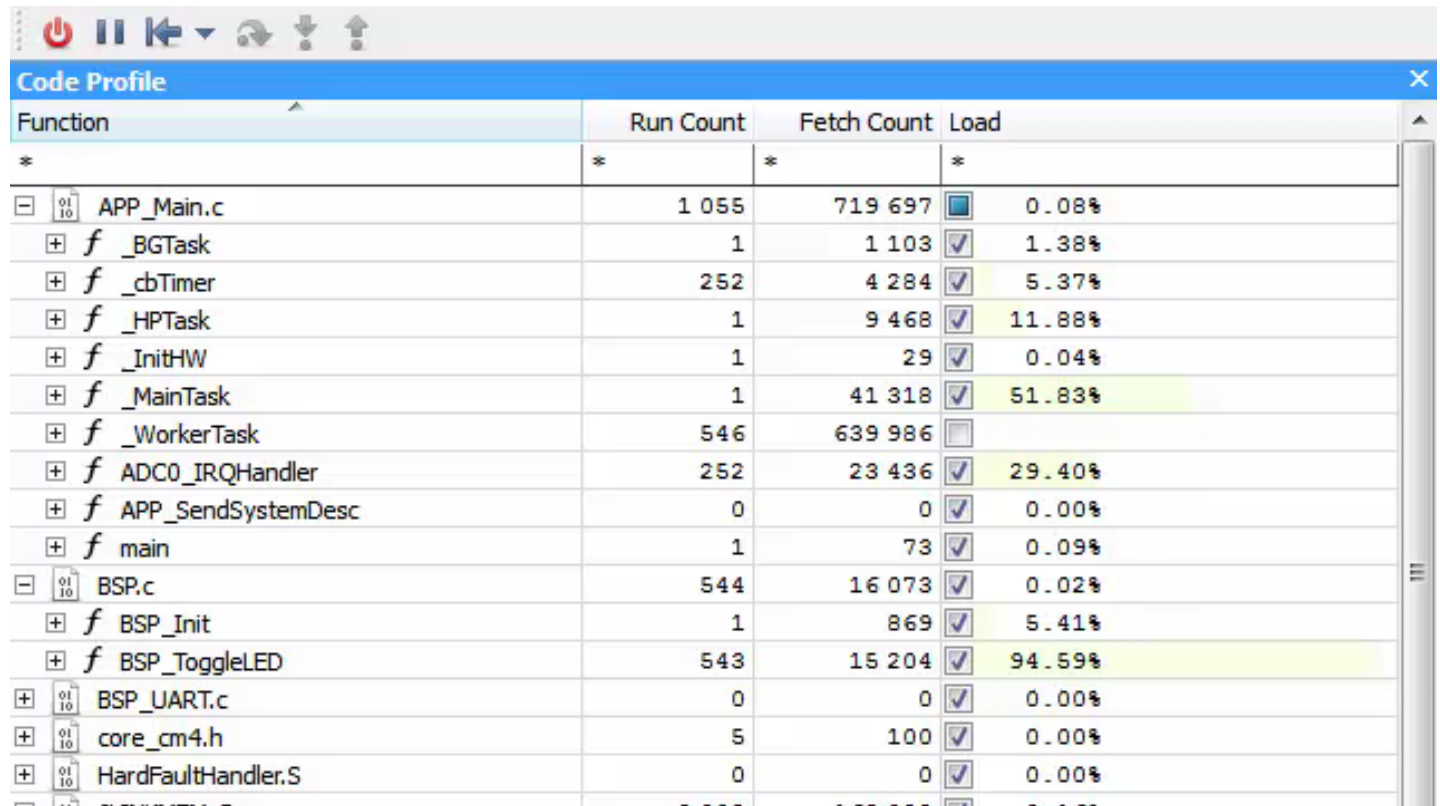


Real-Time Code Coverage with J-Trace PRO and Ozone

# Code Profiling:

## How often has a certain piece of code been executed?

- Measures execution time and frequency of functions, blocks and instructions
- Highlights where computing time is spent
- Potential for optimization:
  - Code that is executed frequently
  - Code that places a high load on the system ("hot spots")



Function	Run Count	Fetch Count	Load
* * *	*	*	*
APP_Main.c	1 055	719 697	0.08%
f _BGTask	1	1 103	1.38%
f _cbTimer	252	4 284	5.37%
f _HPTask	1	9 468	11.88%
f _InitHW	1	29	0.04%
f _MainTask	1	41 318	51.83%
f _WorkerTask	546	639 986	
f ADC0_IRQHandler	252	23 436	29.40%
f APP_SendSystemDesc	0	0	0.00%
f main	1	73	0.09%
BSP.c	544	16 073	0.02%
f BSP_Init	1	869	5.41%
f BSP_ToggleLED	543	15 204	94.59%
BSP_UART.c	0	0	0.00%
core_cm4.h	5	100	0.00%
HardFaultHandler.S	0	0	0.00%

Real-Time Code Profiling with J-Trace PRO and Ozone

# Code Coverage and Code Profiling Reports

- The information from the Code Coverage/Profiling Window can be exported...
  - ...for further analysis in external tools, or
  - ...as human-readable text files to be stored for QA processes or certification

## Ozone Code Profile Report

Project: C:/Projects/K66FN2M0\_emPower/Start\_K66FN2M0.jdebug  
 Application: C:/Projects/K66FN2M0\_emPower/Output/Start\_K66FN2M0.elf

## Code Coverage Summary

Module/Function	Source Lines			Instructions		
APP_Main.c						
_BGTask	5 /	6	83.3%	12 /	13	92.3%
_cbTimer	5 /	6	83.3%	17 /	18	94.4%
_HPTask	14 /	14	100.0%	57 /	57	100.0%
_InitHW	7 /	8	87.5%	29 /	30	96.7%
_MainTask	15 /	15	100.0%	83 /	83	100.0%
_WorkerTask	7 /	8	87.5%	28 /	31	90.3%
ADC0_IRQHandler	6 /	7	85.7%	50 /	56	89.3%
main	11 /	13	84.6%	60 /	64	93.8%
Total	70 /	77	90.9%	336 /	352	95.5%

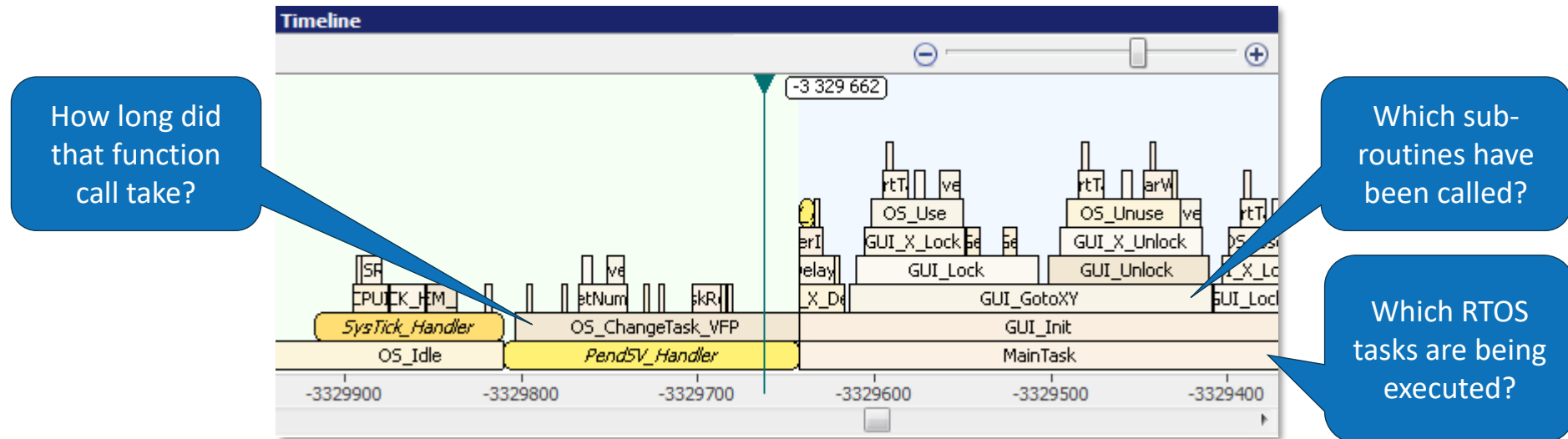
## Code Profile Summary

Module/Function	Run Count	Fetch Count
APP_Main.c		
_BGTask	1	13 022
_cbTimer	291	4 947
_HPTask	1	12 339
_InitHW	1	29
_MainTask	1	51 863
_WorkerTask	646	766 016
ADC0_IRQHandler	291	14 841
main	1	60
Total	1 233	863 117

< >

# Code Timeline: How does everything 'stack up'?

- The Code Timeline provides a graphical representation of the Call Stack over time
- The timeline is based on the recorded trace data
  - Mapped to the source function information

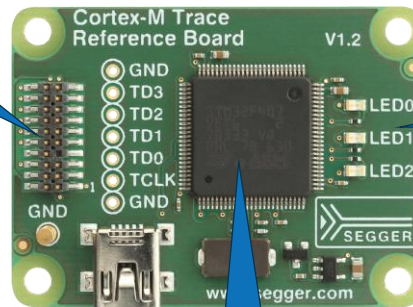


3

**Live Demo:  
Streaming Instruction Trace /  
Real-Time Code Coverage / Real-  
Time Code Profiling**

# SEGGER Cortex-M Trace Reference Board

Debug header  
with Trace pins



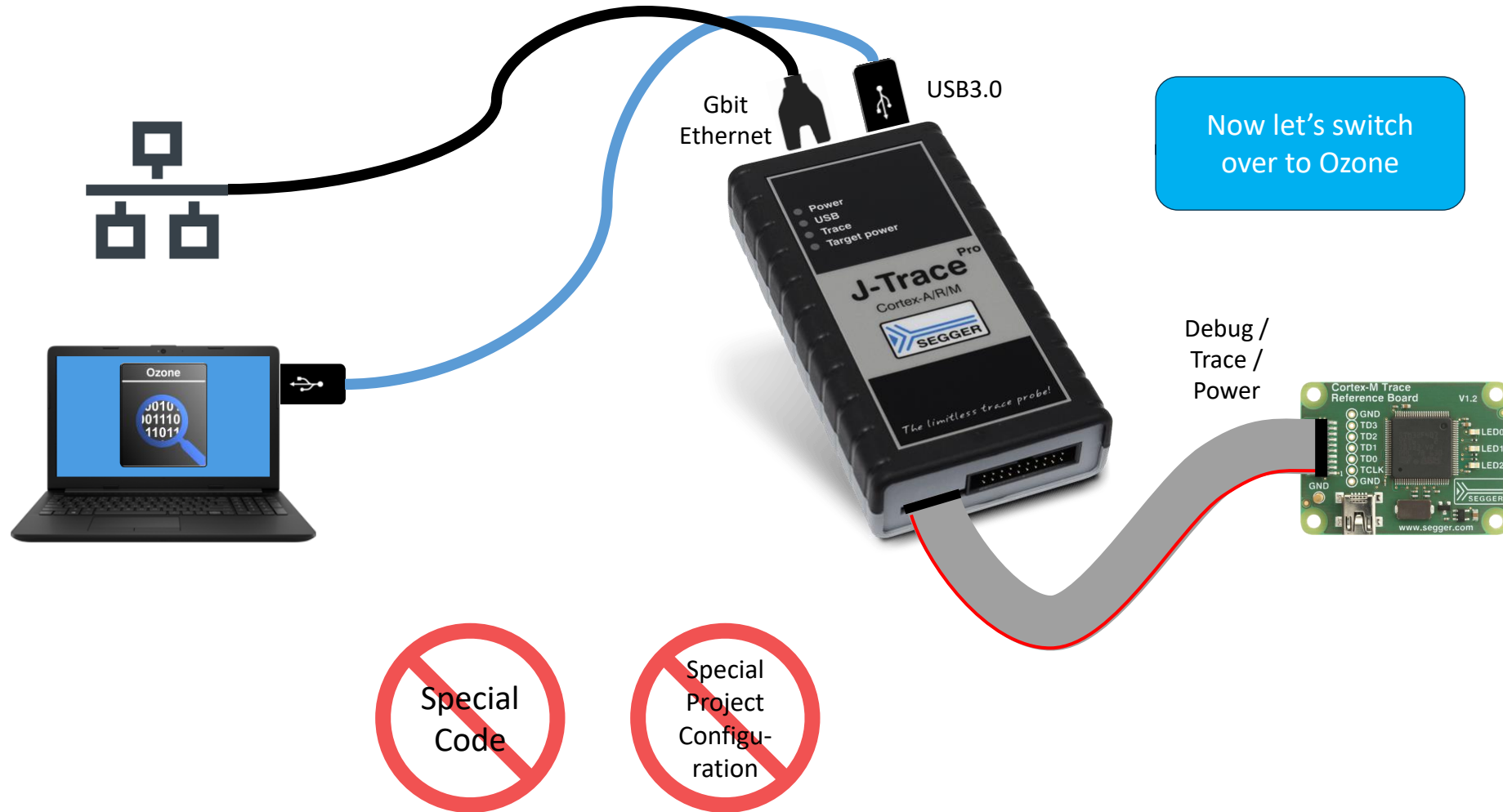
3 LEDs for visual  
feedback

Also available: Trace  
Reference Boards for:  
NXP RT1050  
STM32H7



STM32F407  
Arm Cortex-M4  
MCU

# Hardware Setup





4

## **Other Advanced Debugging Features**

# RTOS Awareness

- Ozone's RTOS awareness plugins provide information about the application's OS, such as:
  - Which task did the system halt at?
  - What are the other tasks doing?
  - How much stack are they using?
- Additionally, a JavaScript interface is available to add RTOS awareness for any OS

Tasks					
ID	Priority	Name	Stack Info (Used / Size)	Status	Run Count
40	113	IP_Task	564 / 1280 @ 0x20006708	Waiting (event), with Timeout	2943
➔ 198	108	GUI_Task	1324 / 4096 @ 0x200060F0	Running	4170
97	106	USBMSDTask	408 / 4096 @ 0x20008C20	Suspended, with Timeout	468
140	105	IP_WebServer	672 / 2048 @ 0x20006E78	Waiting (event object)	2
244	101	IP_FTP_Server	372 / 2072 @ 0x20007A50	Waiting (event object)	2
	Idle	Idle			

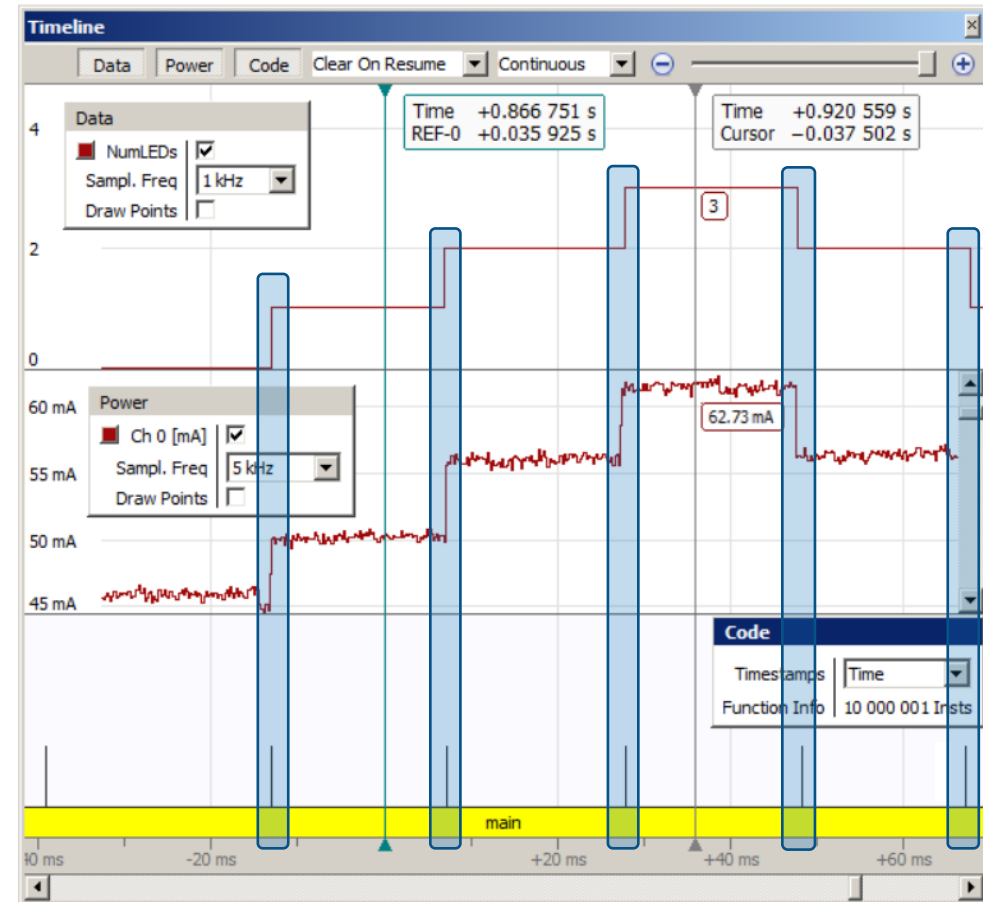
Call Stack			
Function	Stack Info	PC	Return Address
➔ _RadialMenu	0 @ 2000 61E8	0000 27A4	0000 2A0E
GUIDEMO_RadialMenu	8 @ 2000 61E8	0000 2A0A	0000 1162
_Main	8 @ 2000 61F0	0000 1160	0000 1508
GUIDEMO_Main	64 @ 2000 61F8	0000 1504	0000 B3B0
OS_StartTask	0 @ 2000 6238	0000 B3AE	N/A

Quick demo...

# Data & Power Sampling

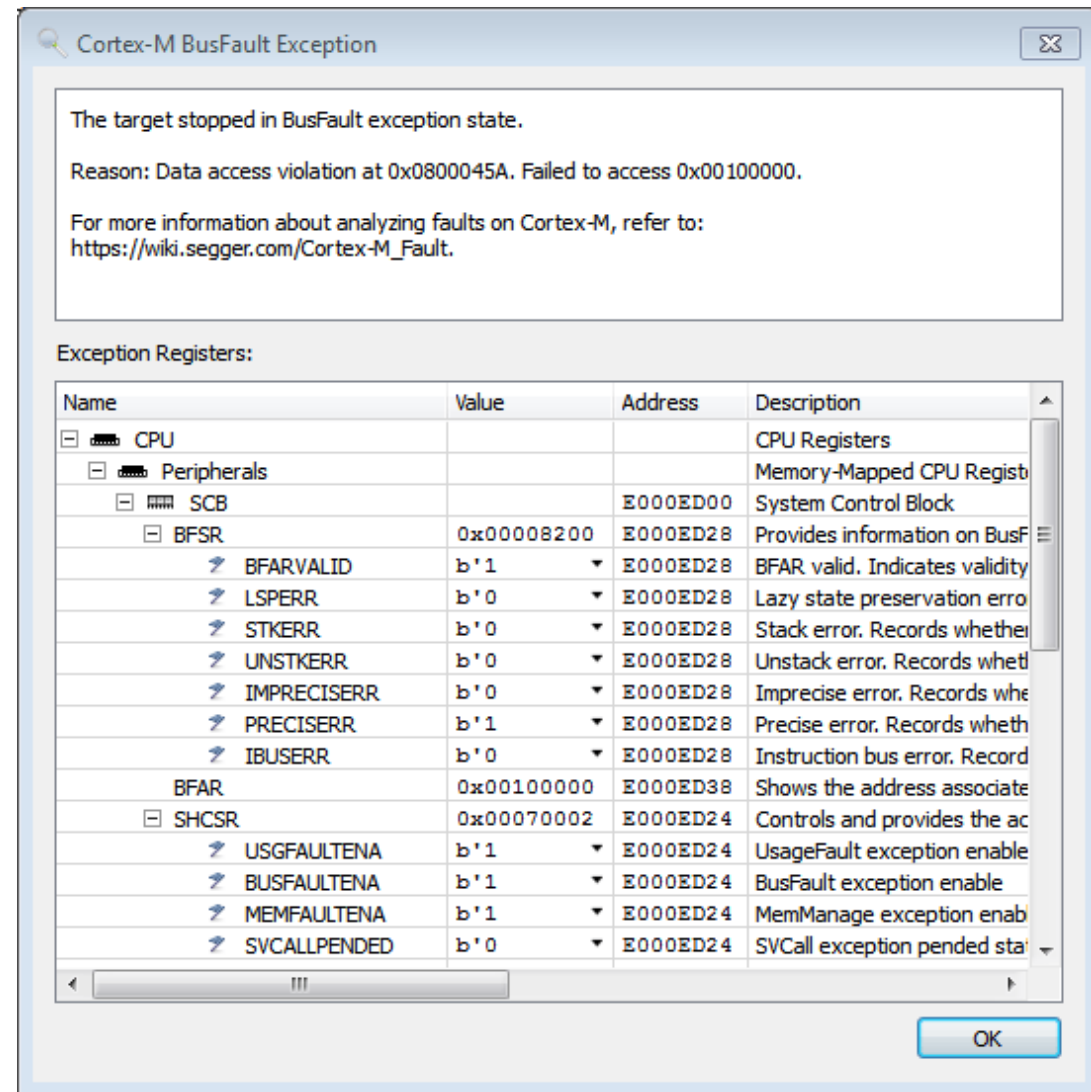
- Part of the Timeline Window
- Allows correlating and visualizing:
  - Data Sampling
  - Current Consumption
  - Program Execution
- Also see video here:  
<https://youtu.be/lu9XpFNgu7Q>

Quick demo...



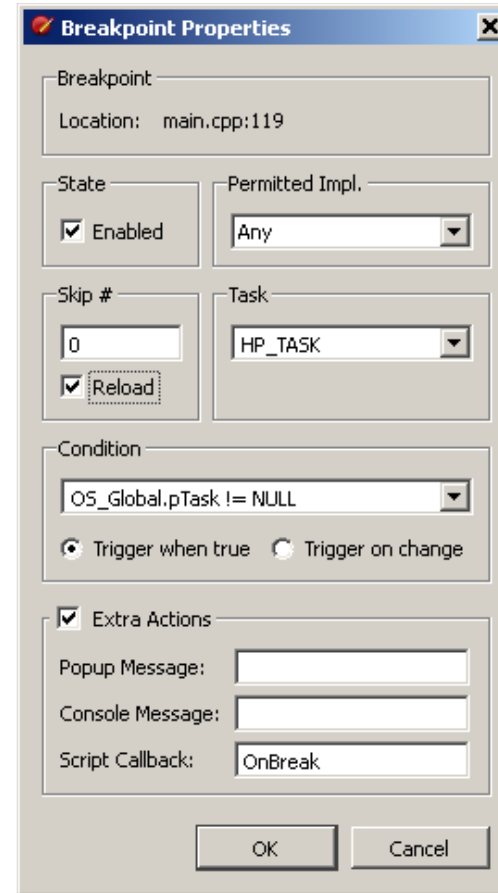
# Target Exception Dialog

- Shows when the target's CPU enters an exception state, such as a fault on Cortex-M
- Also see videos here:  
<https://youtu.be/RaiTXe9huyo>  
<https://youtu.be/oL8qVAVMA0o>



# Conditional Code and Data Breakpoints

- Ozone's (Code) Breakpoint capabilities enable users to specify advanced breakpoint properties
  - Trigger condition
  - Implementation type
- Ozone's Data Breakpoint capabilities enable users to place data breakpoints on global program variables and individual memory addresses



**Breakpoint Properties**

Breakpoint  
Location: main.cpp:119

State  
☒ Enabled

Permitted Impl.  
Any

Skip #  
0

Task  
HP\_TASK

☒ Reload

Condition  
OS\_Global.pTask != NULL

☒ Trigger when true ☐ Trigger on change

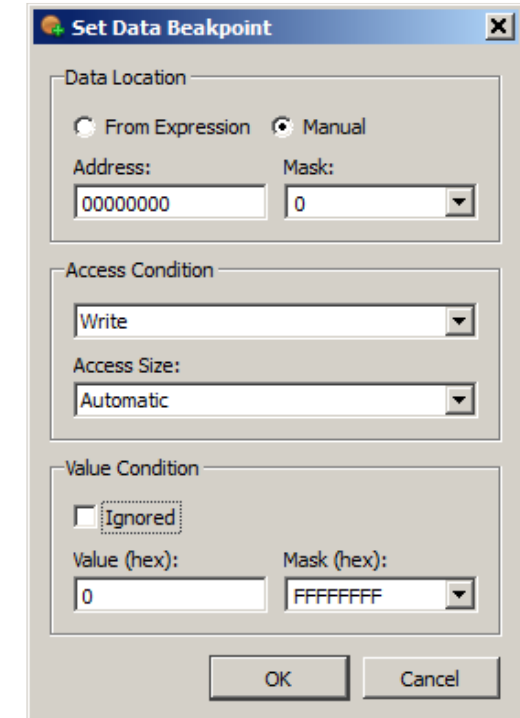
☒ Extra Actions

Popup Message:

Console Message:

Script Callback: OnBreak

OK Cancel



**Set Data Breakpoint**

Data Location  
☐ From Expression ☒ Manual

Address: 00000000 Mask: 0

Access Condition  
Write

Access Size: Automatic

Value Condition  
☐ Ignored

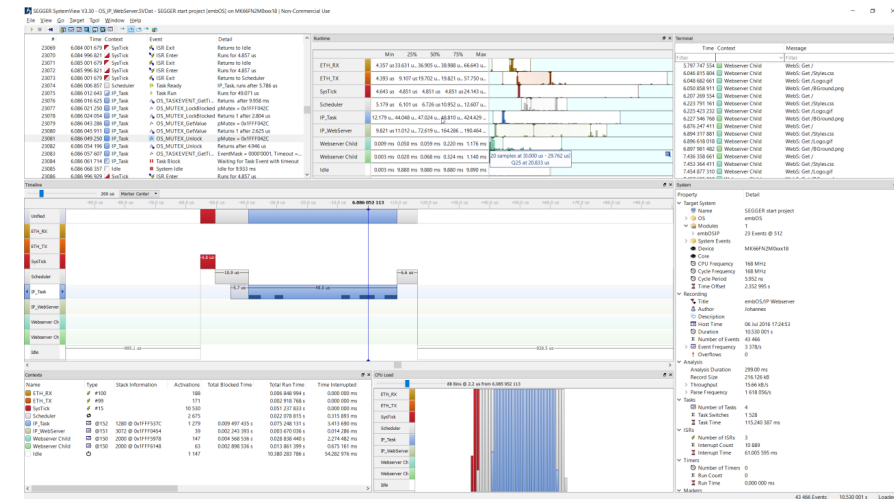
Value (hex): 0 Mask (hex): FFFFFFFF

OK Cancel

# Real-Time Analysis with SystemView



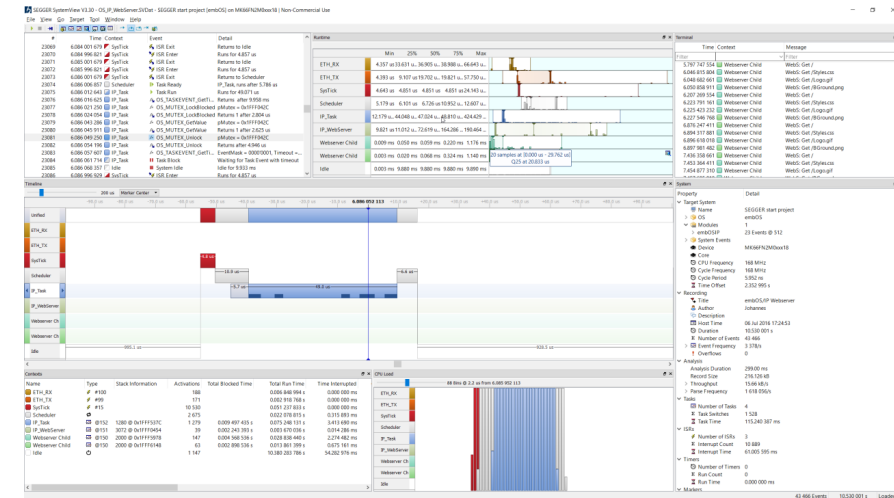
- Toolkit for continuous real-time recording of an embedded application
- Live analysis and visualization of captured data
- Captures tasks, interrupts, timers, resources, API calls, and user events
- Recording via J-Link and SEGGER RTT Technology, IP, or UART
- Minimally system-intrusive
- Works with RTOS and bare-metal systems
- Provides complete insight into an application, to gain a deep understanding of the runtime behavior
  - Particularly advantageous when developing and working in complex systems with multiple tasks and events
- Consists of two parts:
  - A visualization/Recording app, running on any host computer (Windows, MacOS, or Linux)
  - Some embedded code running on the target system



# Benefits of SystemView



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





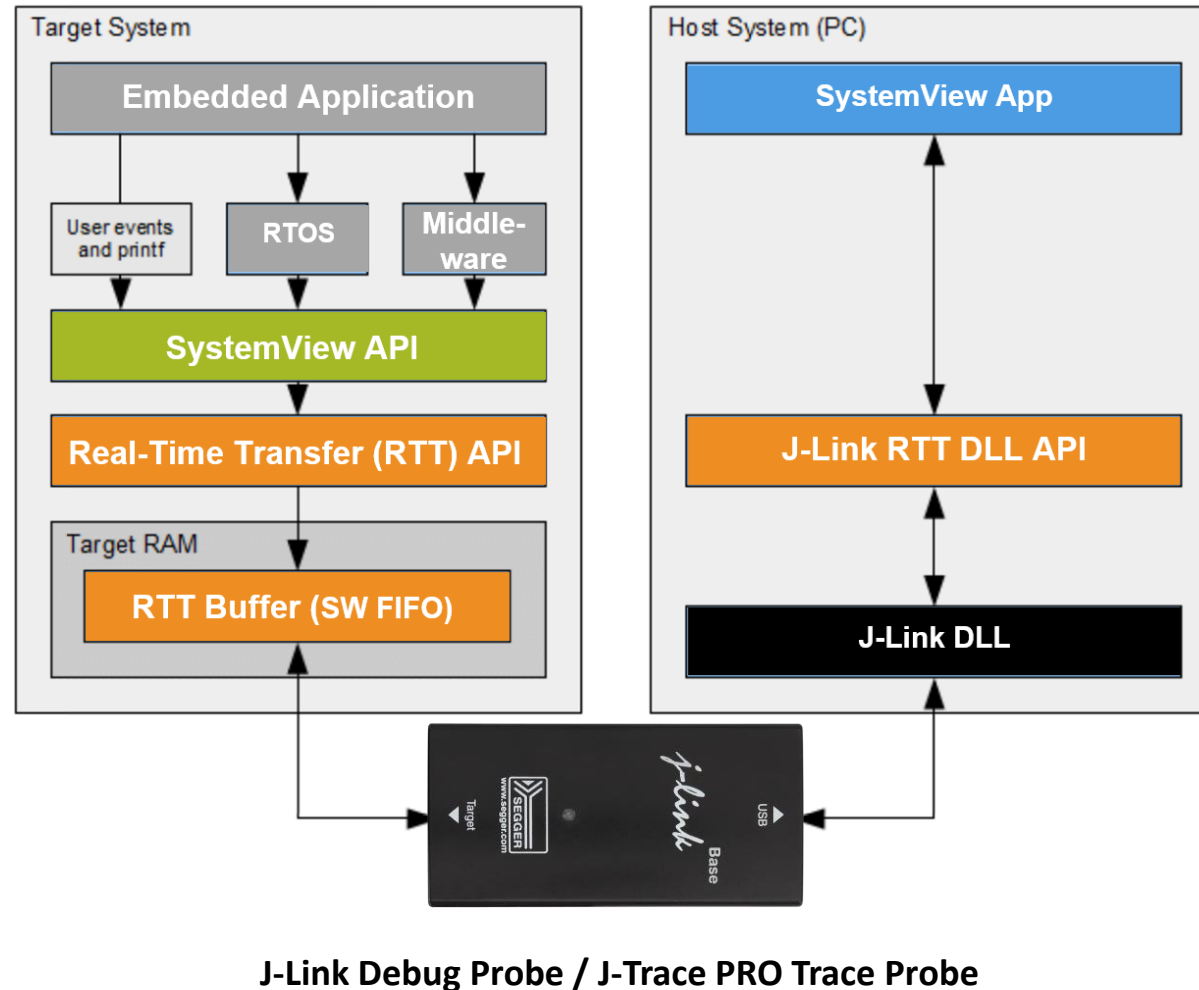
5

## **Live demo: Real-time Recording and Runtime Analysis**

# Hardware Setup



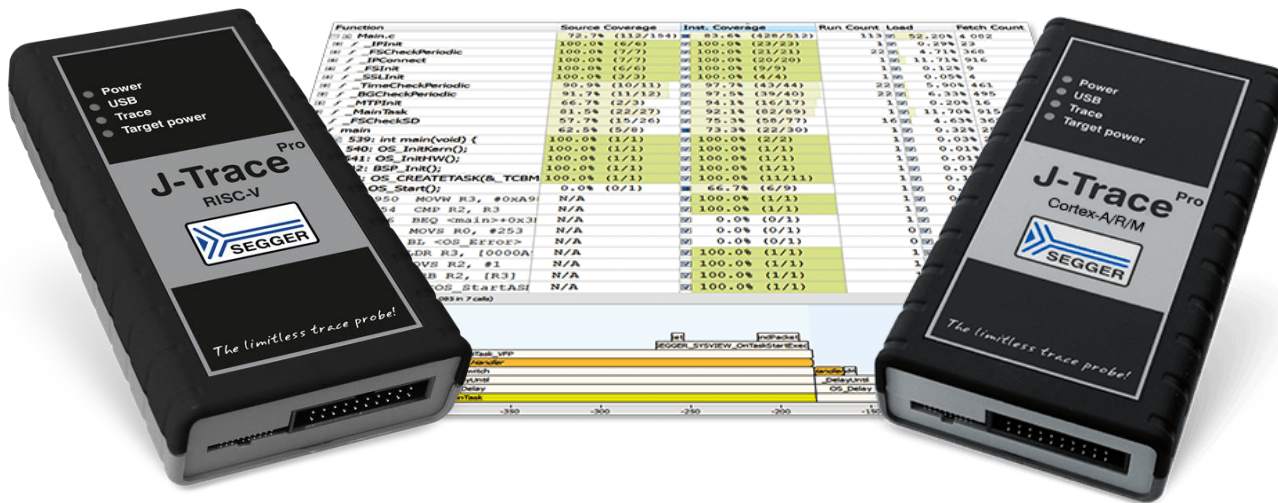
# What's happening under the hood





## **Summary**

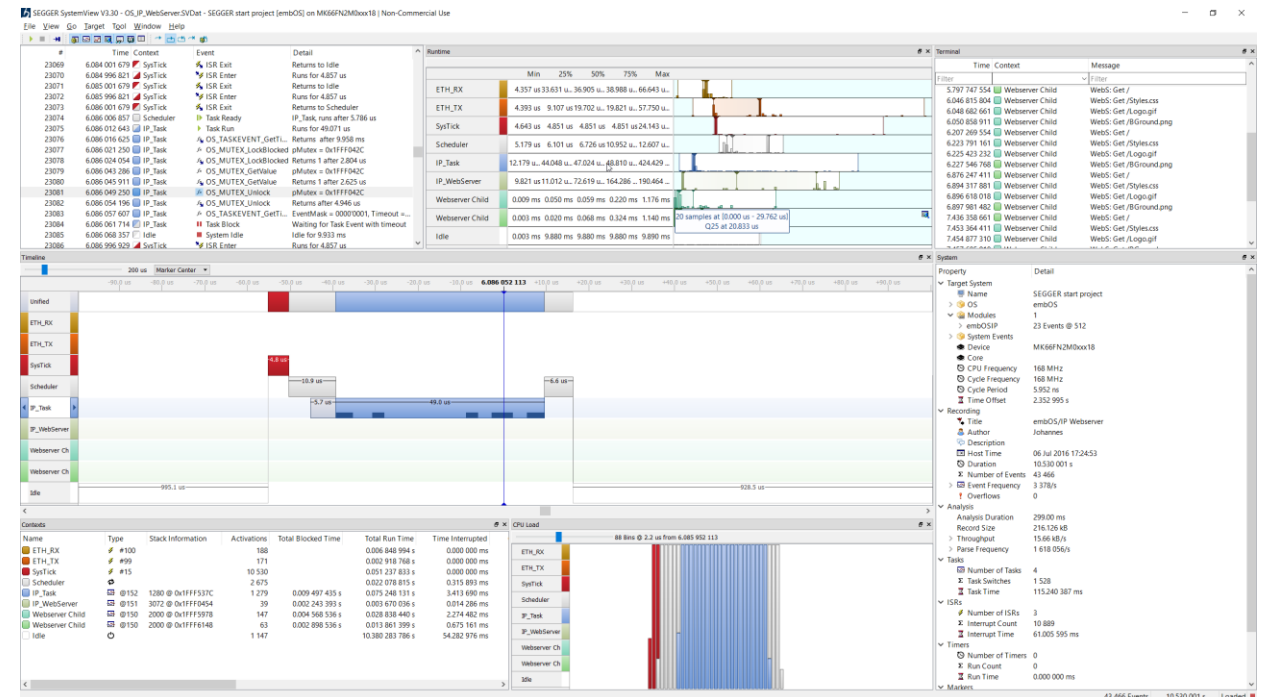
# Summary (I)



- J-Trace PRO enables streaming trace on Cortex-M, Cortex-A, and Cortex-R based targets with ETM/PTM
- J-Trace PRO now also available for RISC-V
- Ozone is a cross-platform debugger and performance analyzer for J-Link and J-Trace
- Together, they offer advanced debug features like
  - Instruction Trace
  - Real-Time Code Coverage
  - Real-Time Code Profiling
  - Code Timeline
  - and more
- Advanced debugging can save you time, money, and frustration

# Summary (II)

- SystemView can provide real-time visualization and analysis of a target application's runtime behavior
- System behavior can be recorded for off-line analysis
- SystemView and RTT don't require additional port pins or hardware (other than the J-Link debug probe)
- RTT can now also be used with RISC-V based devices that support System Bus Access
- SEGGER's plea to RISC-V core and SoC Designers: When implementing the RISC-V Debug Module, include the System Bus Access block to take advantage of RTT and SystemView
- This is an optional feature, but worth adding...



# THANK YOU

Embedded  
Online  
Conference



w w w . e m b e d d e d o n l i n e c o n f e r e n c e . c o m

# Embedded Online Conference

w w w . e m b e d d e d o n l i n e c o n f e r e n c e . c o m