Continuous Observability in Embedded Systems

In this white paper, we share how Percepio Tracealyzer and Detect, when used together, bring real-time, automated insights that turn debugging from reactive firefighting into proactive quality assurance.

Continuous Observability in Embedded Systems: Enabling Real-Time Insight From Development to Field Use

Abstract

As embedded systems become increasingly complex and interconnected, ensuring their reliability and performance throughout the development lifecycle is paramount. This white paper introduces the concept of Continuous Observability, emphasizing how the integration of Percepio Tracealyzer and Percepio Detect provides a comprehensive solution for real-time monitoring, debugging, and performance optimization in RTOS-based embedded systems.

1. Introduction

Embedded systems are the backbone of modern technology, powering everything from smart home devices to critical infrastructure. As these systems become more complex and software-driven, the ability to monitor, debug, and optimize system behavior becomes critical to success. Traditional debugging and testing techniques are often insufficient, especially when issues emerge sporadically or under specific conditions that are hard to replicate.

Embedded system developers require tools that offer deeper visibility and actionable insights throughout the product lifecycle to address these challenges. Continuous Observability offers a framework within which such tools can enable proactive monitoring and analysis from development through deployment.

Adopting tools that enable Continuous Observability fits seamlessly into modern software development practices such as DevOps, DevSecOps, and CI/CT pipelines. And observability is not just a debugging aid—it becomes a key enabler for continuous verification and validation (V&V). By integrating Tracealyzer and Detect into existing pipelines, engineering teams gain actionable feedback loops that align with agile methodologies and regulatory compliance efforts.

2. The Need for Continuous Observability

Continuous Observability is a modern approach to embedded system development and maintenance that emphasizes real-time, end-to-end insight into system behavior. Rather than relying solely on logs or isolated debugging sessions, system behavior is continuously monitored to enable rapid detection and analysis on runtime issues, turning nightmare bugs into quick fixes, and identifying brittleness risks before they turn into cost escalating problems such as recalls, cyber vulnerabilities and service interruptions.

Key benefits include:

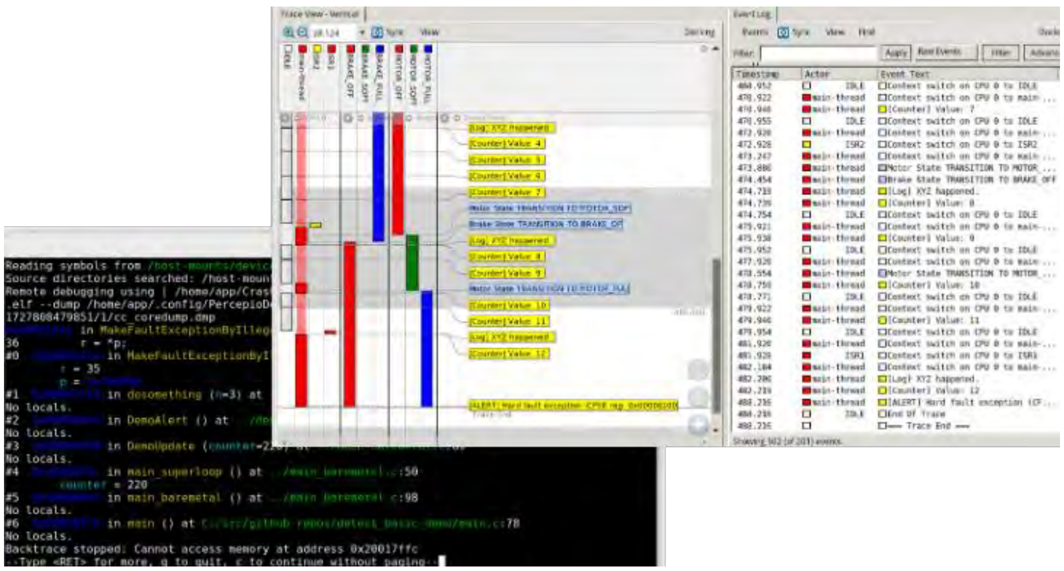
- Early Detection of Issues – Identify bugs and anomalies during development and testing phases.

- Enhanced Debugging Capabilities – Access detailed visual diagnostics for quicker issue resolution.
- Improved System Reliability – Maintain a continuous understanding of system health post-deployment.

3. Percepio Tracealyzer: Visual Trace Diagnostics

Percepio Tracealyzer is a powerful visual trace diagnostics tool for embedded systems that enables developers to understand system-level behavior through a suite of over 30 graphical views. These views provide deep insights into:
- Task scheduling and execution
- CPU usage and memory consumption
- Event and interrupt handling
- Inter-process communication and timing

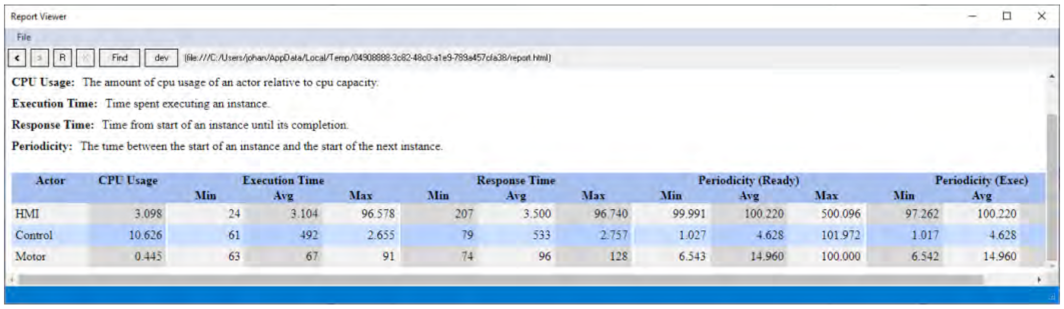
This granular visibility allows engineers to trace issues back to their root cause, analyze system performance, and optimize scheduling and resource usage more effectively. By offering both high-level overviews and detailed drill-downs, Tracealyzer accelerates time-to-resolution and boosts confidence in system stability.



```
Reading symbols from /host-mounts/devic
Source directories searched: /host-moun
Remote debugging using | /home/app/Cras
.elf --dump /home/app/.config/PercepioD
1727808479851/1/cc_coredump.dmp
             in MakeFaultExceptionByIlleg
36         r = *p;
#0               in MakeFaultExceptionByI
        r = 35
        p =
#1               in dosomething (n=3) at
No locals.
#2               in DemoAlert () at   /de
No locals.
#3               in DemoUpdate (counter=22
No locals.
#4               in main_superloop () at ../main_baremetal.c:50
        counter = 220
#5               in main_baremetal () at ../main_baremetal.c:98
No locals.
#6               in main () at C:/src/github repos/detect_basic_demo/main.c:78
No locals.
Backtrace stopped: Cannot access memory at address 0x20017ffc
--Type <RET> for more, q to quit, c to continue without paging--
```

```
Timestamp   Actor         Event Text
468.952           IDLE    Context switch on CPU 0 to IDLE
470.922     main-thread   Context switch on CPU 0 to main-...
470.946     main-thread   [Counter] Value: 7
470.955           IDLE    Context switch on CPU 0 to IDLE
472.920     main-thread   Context switch on CPU 0 to main-...
472.926           ISR2    Context switch on CPU 0 to ISR2
473.247     main-thread   Context switch on CPU 0 to main-...
473.886     main-thread   Motor State TRANSITION TO MOTOR_...
474.454     main-thread   Brake State TRANSITION TO BRAKE_OFF
474.719     main-thread   [Log] XYZ happened.
474.739     main-thread   [Counter] Value: 8
474.754           IDLE    Context switch on CPU 0 to IDLE
475.921     main-thread   Context switch on CPU 0 to main-...
475.938     main-thread   [Counter] Value: 9
475.952           IDLE    Context switch on CPU 0 to IDLE
477.920     main-thread   Context switch on CPU 0 to main-...
478.554     main-thread   Motor State TRANSITION TO MOTOR_...
478.750     main-thread   [Counter] Value: 10
478.771           IDLE    Context switch on CPU 0 to IDLE
479.922     main-thread   Context switch on CPU 0 to main-...
479.946     main-thread   [Counter] Value: 11
479.954           IDLE    Context switch on CPU 0 to IDLE
481.920     main-thread   Context switch on CPU 0 to main-...
481.928           ISR1    Context switch on CPU 0 to ISR1
482.184     main-thread   Context switch on CPU 0 to main-...
482.206     main-thread   [Log] XYZ happened.
482.219     main-thread   [Counter] Value: 12
488.236     main-thread   [ALERT] Hard fault exception (CF...
488.236           IDLE    End Of Trace
488.236                   --- Trace End ---
Showing 102 (of 201) events.
```

Figure 1: Tracealyzer view showing thread execution timing and event logging, used during crash analysis.


Report Viewer
File
Find
dev
(file:///C:/Users/johan/AppData/Local/Temp/04908888-3c82-48c0-a1e9-789a457cfa38/report.html)

CPU Usage: The amount of cpu usage of an actor relative to cpu capacity.

Execution Time: Time spent executing an instance.

Response Time: Time from start of an instance until its completion.

Periodicity: The time between the start of an instance and the start of the next instance.

Actor	CPU Usage	Execution Time			Response Time			Periodicity (Ready)			Periodicity (Exec)	
		Min	Avg	Max	Min	Avg	Max	Min	Avg	Max	Min	Avg
HMI	3.098	24	3.104	96.578	207	3.500	96.740	99.991	100.220	500.096	97.262	100.220
Control	10.626	61	492	2.655	79	533	2.757	1.027	4.628	101.972	1.017	4.628
Motor	0.445	63	67	91	74	96	128	6.543	14.960	100.000	6.542	14.960

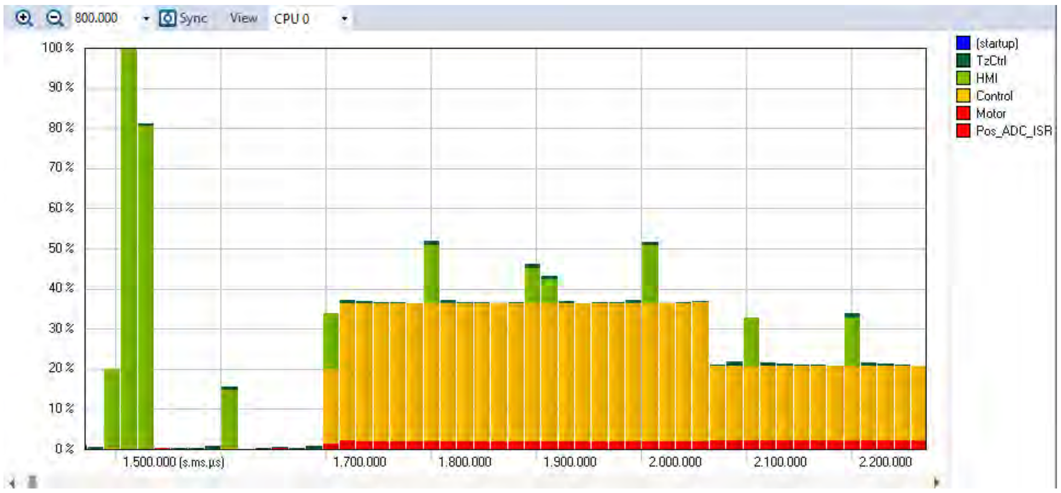


Figure 2: CPU usage and task activity profile and CPU load graph, used for load balancing optimization and bottleneck avoidance.

4. Percepio Detect: Real-Time Anomaly Detection

Percepio Detect extends observability into runtime anomaly detection and in-field monitoring. Designed to operate with minimal performance impact, Percepio Detect continuously observes key system metrics and captures snapshots when anomalies occur.

Core capabilities include:
- Automated detection of crashes, hangs, and performance issues
- Snapshot logging with contextual state information for post-mortem debugging
- Seamless integration into CI/CT pipelines and test automation frameworks

By surfacing critical runtime issues as they occur, Percepio Detect empowers development and QA teams to resolve problems before they impact production deployments.

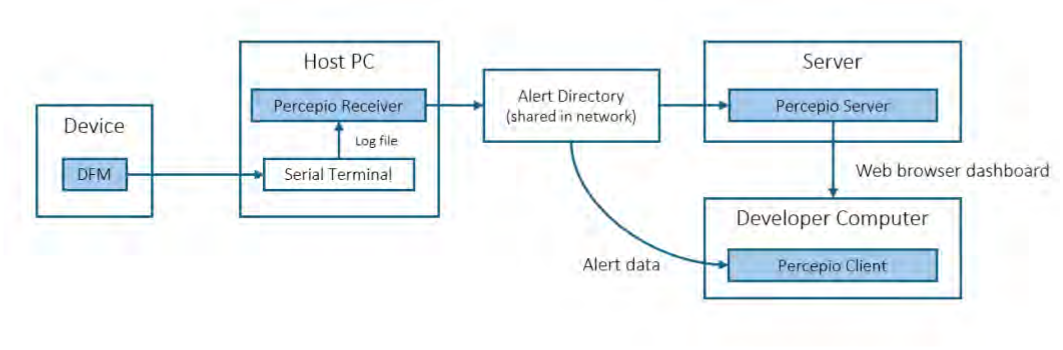
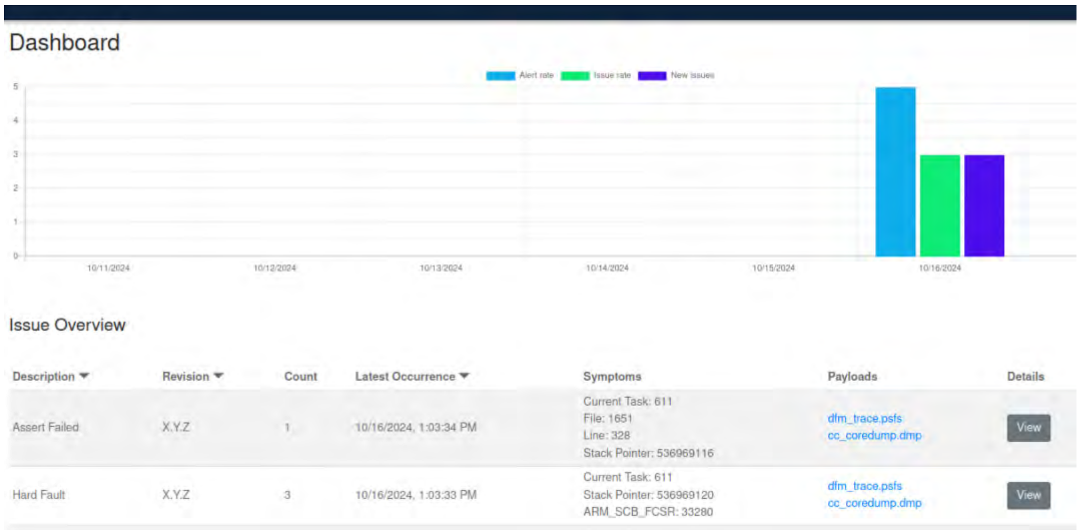


Figure 3: High-level architecture of Percepio Detect, capturing alerts and logs for visualization in the web browser dashboard.



Description	Revision	Count	Latest Occurrence	Symptoms	Payloads	Details
Assert Failed	X.Y.Z	1	10/16/2024, 1:03:34 PM	Current Task: 611 File: 1651 Line: 328 Stack Pointer: 536969116	dfm_trace.psfs cc_coredump.dmp	View
Hard Fault	X.Y.Z	3	10/16/2024, 1:03:33 PM	Current Task: 611 Stack Pointer: 536969120 ARM_SCB_FCSR: 33280	dfm_trace.psfs cc_coredump.dmp	View

Figure 4: Snapshot of the Percepio Detect server dashboard. Each "alerts" has its associated payload of diagnostics data, easily accessible for analysis by clicking on the links.

5. Synergizing Tracealyzer and Percepio Detect for Continuous Observability

While Percepio Tracealyzer and Detect each bring valuable observability capabilities, their true power is realized when used together as part of an integrated solution.

Tracealyzer has traditionally been used reactively, where developers manually capture trace recordings to analyze errors after observing malfunctions. With Percepio Detect, this approach evolves: now, trace data collection can be automated based on runtime anomaly detection. Instead of continuously recording, Detect monitors system behavior 24/7—especially during automated CI/CT test cycles or field operations. When something unexpected is detected, Detect triggers and stores a snapshot and optionally a full trace recording.

The next morning, engineers reviewing the Percepio Detect dashboard can quickly spot flagged anomalies, review trace data, and start troubleshooting without needing to reproduce the problem manually. This workflow aligns perfectly with DevOps and DevSecOps principles: short feedback loops, proactive error handling, and continuous quality assurance.

This synergy enables a proactive observability pipeline across the entire lifecycle:
- During development: Tracealyzer helps visualize and debug RTOS behavior.
- During CI/CT testing: Detect monitors for regressions or rare bugs and captures actionable evidence.
- During deployment: Detect ensures in-field robustness and enables trace-based diagnostics if needed.

By embedding observability into daily builds, nightly tests, and post-deployment monitoring, Percepio's solution transforms quality assurance from a reactive step to a continuous, integrated activity.

6. Case Study: Solving a Mysterious Watchdog Reset with Continuous Observability

A European manufacturer of building automation systems was preparing a firmware update for their next-generation HVAC controller—an RTOS-based device powered by an Arm Cortex-M33 MCU running FreeRTOS. During overnight integration testing, a seemingly random watchdog reset was recorded. The reset was rare, occurred at different times, and left no useful debug logs.

Challenge: An Intermittent Watchdog Reset with No Clues

This kind of failure is a developer's nightmare. It passed through static analysis and unit tests undetected. It didn't occur in development builds with logging enabled. And it couldn't be reproduced at will in the lab. With the release deadline approaching, the team needed answers fast.

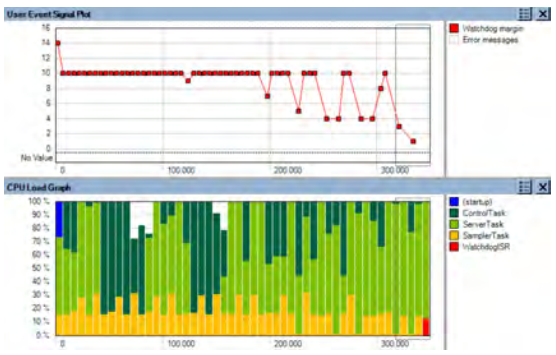
To accelerate root cause analysis, the team deployed **Percepio Detect** with watchdog reset detection enabled. The system was configured to capture trace snapshots and system metrics whenever a reset occurred during CI test runs.

Diagnosis: The Trace Tells the Story

The next morning, Detect had flagged a watchdog reset at 03:47 AM. A trace recording showing events leading up to the reset was attached to the alert.

Opening the trace in **Percepio Tracealyzer**, the team reviewed the final milliseconds of system activity. They observed that a critical sensor-handling task had been blocked for a long time. This task had a watchdog "kick" responsibility—it was supposed to reset the watchdog timer regularly. Since the task took too long to resume, the watchdog eventually fired.

Tracing backward in time, the engineers discovered the root cause: the watchdog thread also sent data to a message queue, that sometimes became full and blocked the thread due to a priority inversion problem. This timing-sensitive flaw had escaped all earlier testing due to its rarity and dependence on external sensor activity patterns.



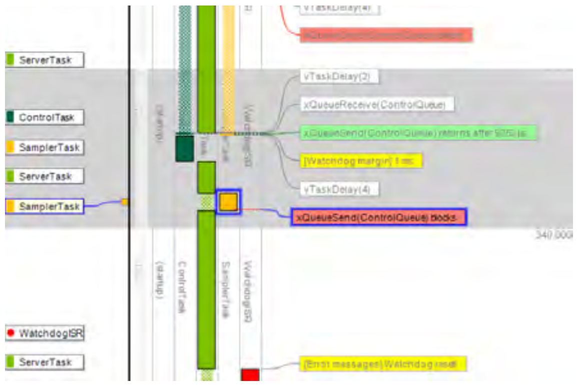


Figure 5 Left: Tracealyzer view showing the watchdog margin shrinking over time (top), and CPU load distribution (bottom). The final spike in red marks the Watchdog ISR just before system reset, caused by the blocked telemetry/control task.

Figure 6 Right: Zoomed-in trace timeline showing the precise blocking behavior: a `xQueueSend()` call blocks, preventing the control task from kicking the watchdog. The resulting timeout triggers a reset, clearly marked as an error event in the trace.

Resolution: One Bug Fixed, Several Prevented

Armed with this trace data and understanding of the flaw, the team restructured the logic to ensure correct behavior even in edge cases. They also added additional Detect alerts on task blocking exceeding warning levels using Stopwatch alerts.

Several minor improvements were also made to increase fault tolerance and observability across the firmware. What started as an unsolved mystery became a catalyst for system-wide improvement—thanks to Continuous Observability powered by Detect and Tracealyzer.

7. Conclusion

As embedded systems continue to evolve in complexity and criticality, developers must adopt practices that provide ongoing, real-time visibility into their systems. The integration of Percepio Tracealyzer and Detect enables a new standard of Continuous Observability, helping teams deliver more robust, performant, and reliable products.

For organizations practicing DevOps or working within tightly regulated industries, this integration provides measurable value. Continuous Observability strengthens the traceability and transparency required for safety-critical systems and simplifies audit readiness. Moreover, the ability to detect and triage issues within hours rather than days supports faster iteration and continuous improvement.

8. Learn More and Contact Percepio

Percepio is proud to support embedded software teams worldwide in achieving higher reliability, faster debugging, and deeper runtime insights. Whether you're developing mission-critical firmware, scaling CI/CT pipelines, or transitioning to software-defined architectures, Continuous Observability can transform your development process.

Visit us online to explore:

Product details and trials
Percepio Tracealyzer
Percepio Detect

Technical insights and blogs
https://percepio.com/blog

Request a demo or evaluation
https://percepio.com/contact-us

Further Reading

- **White paper**: Stop Guessing: See Inside RTOS Firmware with Visual Tracing
- **Blog post**: A Dynamic Duo for Complex Embedded Environments
- **Magazine Article:** Recap of Latest Innovations at EW25: Percepio: enhancing embedded software reliability with Tracealyzer and Detect. Marizio Di Paolo Emilio covers Embedded World 2025 in embedded.com.