13 — RTOS Tracing (FreeRTOS)

> **TraxProbe & Traxcope User Manual** · Version 1.0.0 · Updated 2026-08-24
 · https://embedya.com

The headline feature of RTOS tracing: **zero application code**. Once the FreeRTOS port is wired in (two lines in `FreeRTOSConfig.h`), every context switch, queue operation, mutex take, heap allocation, and the kernel tick is traced automatically. Traxcope turns the event stream into a task timeline (Trace View), a searchable event table (Event View), and live memory health (Memory View).

On the device

Enabling

```
/* trax_config.h */
#define TRAX_CFG_RTOS_TYPE          TRAX_RTOS_FREERTOS
#define TRAX_CFG_FREERTOS_VERSION   TRAX_FREERTOS_VERSION(11, 2, 0)  /*
mandatory — your kernel */
```

```
/* FreeRTOSConfig.h — bottom of the file */
#define configUSE_TRACE_FACILITY  1

#if (configUSE_TRACE_FACILITY == 1)
#include "trax.h"
#endif
```

The RTOS port is selected from `trax_config.h` — there is nothing extra to add to the build. The version triple must match `tskKERNEL_VERSION_NUMBER` in your kernel's `task.h` (supported range V10.2.0 … V11.2.x). Build details and the recommended startup pattern (reuse an application task + optional `trax_wait_stream_started()`) are in Chapter 02 §5.

What gets traced automatically

Category	Events
Task lifecycle	create (with name + priority), delete, switch in/out, ready, suspend/resume, delay, priority set, **priority**

Category	Events
	inheritance/disinheritance, create-failed, stack overflow
Queues	send/receive (with depth), peek, failures, **about-to-block** events (the explanation for every SWITCH_OUT), queue sets
Semaphores & mutexes	give/take (with count), failures, recursive mutex take/give with nesting level
Task notifications	notify/give/take/wait incl. blocking variants
Stream & message buffers	send/receive/reset, failures, blocking
Event groups	set/clear/wait/sync bits
Software timers	command send/received (queue latency measurable), expired
Heap	every `pvPortMalloc` / `vPortFree` with address, size, and remaining heap; allocation failures
Stack	periodic per-task high-water-mark samples
Kernel tick	SysTick enter/exit (also drives the probe's timestamp tick — no manual hook needed)
Low-power	tickless-idle sleep begin/end, tick-count jumps

Object **names** are captured at create time and via the queue registry (`vQueueAddToRegistry`), so Traxcope shows "work_q", not `0x20001234` . Set `configQUEUE_REGISTRY_SIZE > 0` to use the registry.

Recommended FreeRTOSConfig extras

Define	Enables
`INCLUDE_uxTaskGetStackHighWaterMark 1`	Per-task stack headroom in the Memory View (also snapshotted at session start)
`configCHECK_FOR_STACK_OVERFLOW 2`	Library's weak `vApplicationStackOverflowHook` emits a stack-overflow event the moment the kernel detects it
`configQUEUE_REGISTRY_SIZE 8`	Named queues/semaphores/mutexes in all views

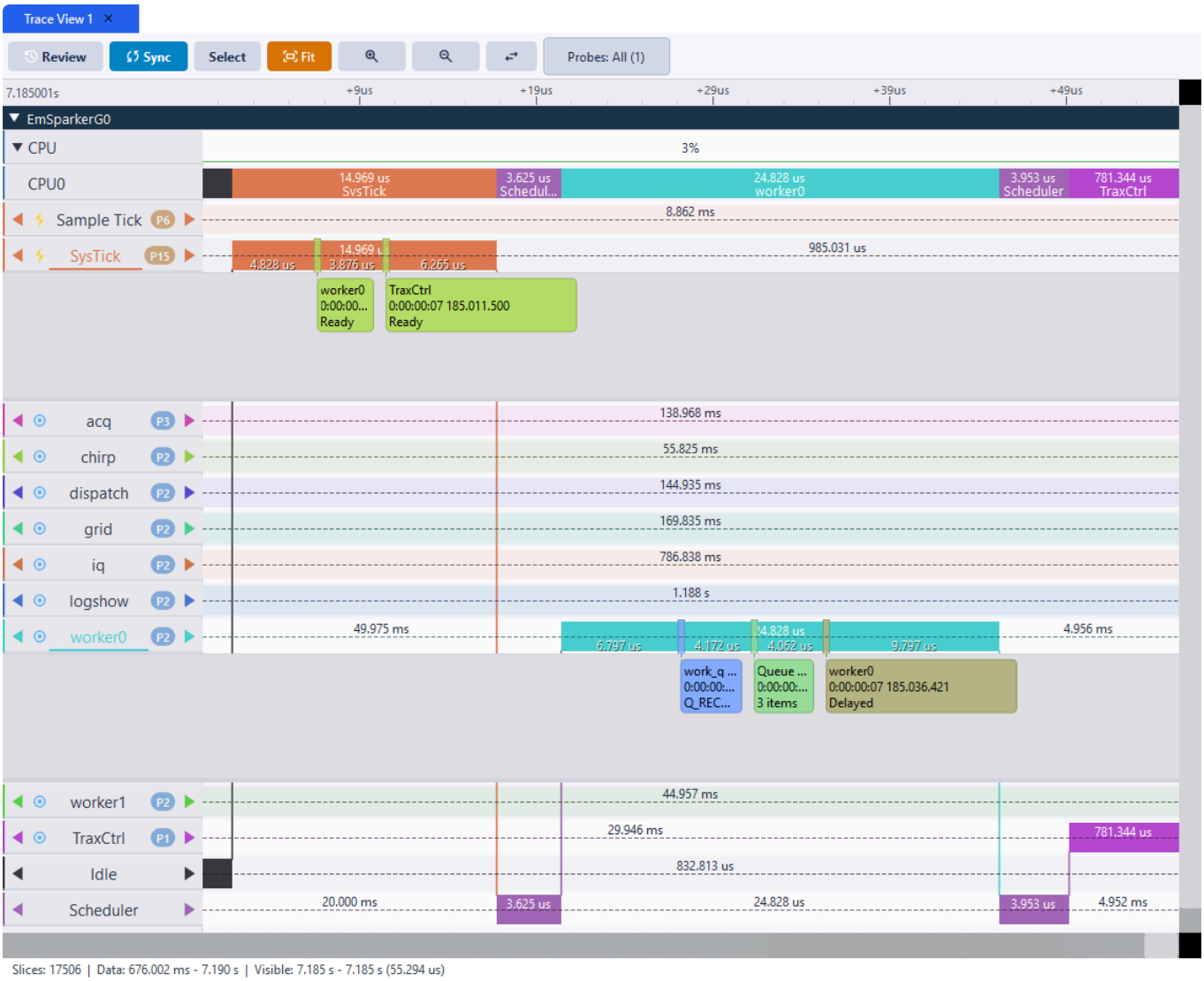
The combination of periodic stack polling ("saw it coming") and the overflow event ("watched it happen") covers the most common RTOS failure mode end to end.

Related `trax_config.h` knobs (full table in Chapter 02 §5): `TRAX_CFG_ISR_YIELD_TO_SCHEDULER` (default `1` — omit `ISR_EXIT` when the tick/user ISR yields so the timeline is ISR → Scheduler → new task), `TRAX_CFG_OWN_FREERTOS_TICK_HOOK`, `TRAX_CFG_MAX_RTOS_TASKS` / `_OBJECTS`, `TRAX_CFG_RTOS_TASK_NAME_MAX`.

In Traxcope

Trace View — the execution timeline

Open via Toolbox → **Trace**. This is the flagship RTOS view: horizontal **swimlanes**, one per execution context, with colored **execution slices** showing exactly when each context ran.



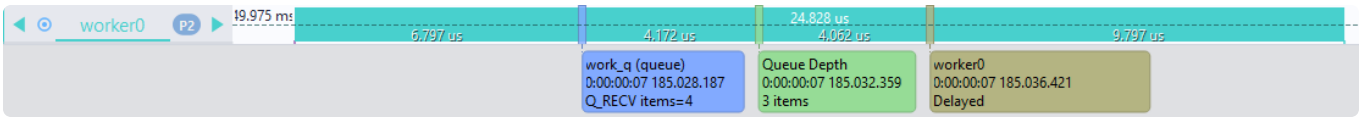
Screenshot: Trace View, several task lanes + ISR lanes + idle lane, slices and event ticks visible

Lane types:

Lane	Contains
Task lanes	One per FreeRTOS task; slices = running time, event markers for queue/sem/mutex/notify activity on that task
ISR lanes	Interrupt handlers (kernel tick automatically; your own ISRs via Chapter 14), sorted by hardware priority
Idle	The idle task — your CPU headroom at a glance
Scheduler	Kernel-level activity
Main	Bare-metal main-loop slices (Chapter 14)

Working in the Trace View:

- **Live / Follow Live** — the timeline scrolls with incoming data; drop out to inspect.
- **Fit / zoom / X-drag zoom** — navigate from hours down to microseconds.
- **Slice selection & hover** — click a slice for its details (context, duration, bounding events); blocked tasks show *why* they blocked thanks to the about-to-block events.
- **Select mode + Sync** — drag a span or set a cursor and every synced view follows (Chapter 03 §5). The classic move: select an error log line in the Log View and land exactly on the guilty slice here.
- **Per-instance probe visibility** — in multi-probe setups each Trace View instance picks which probes to show; with hardware sync enabled the lanes of different boards share one true timeline (Chapter 18).
- When a **trigger** fires, the view exits live mode and centers on the trigger instant (Chapter 17).

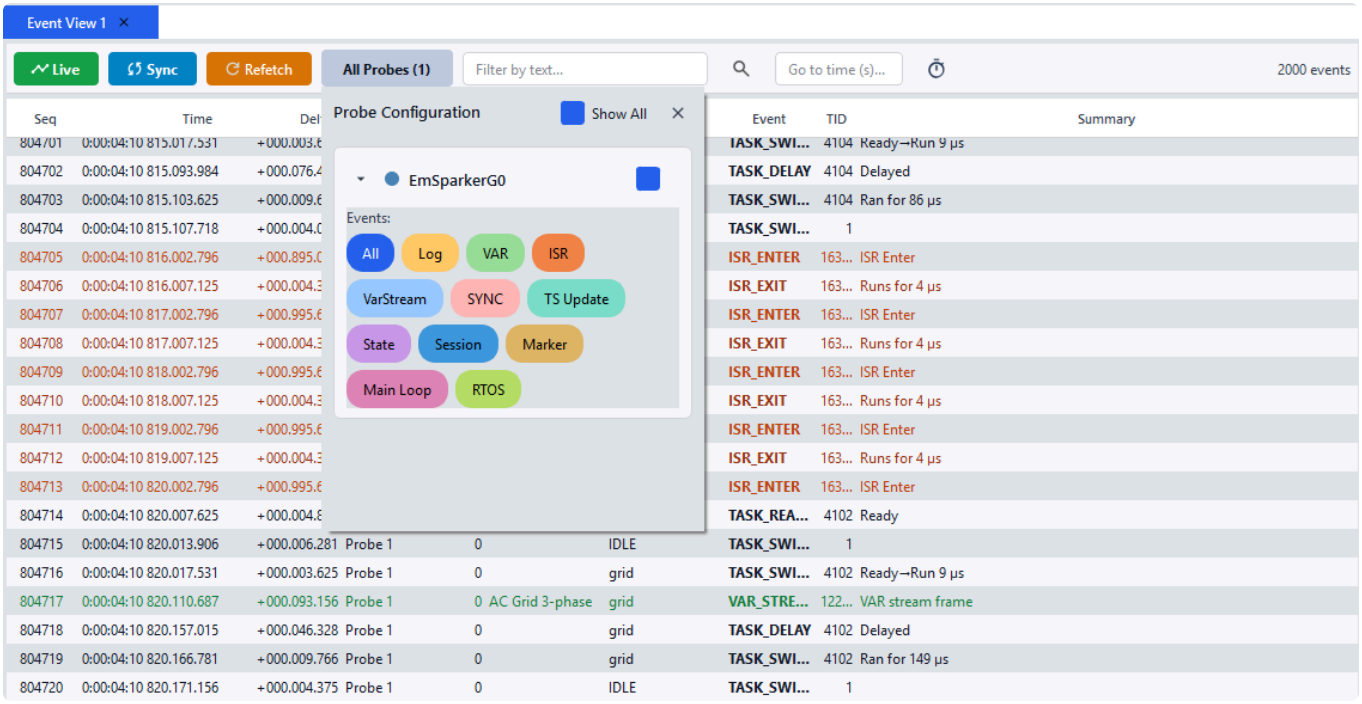


Screenshot: Trace View, one slice selected, detail panel showing duration and events

What to look for: priority inversion (a high-priority task's lane gap bridged by a `PRIORITY_INHERIT` event), queue starvation (recurring `RECV_BLOCK` events), jitter in a periodic task's slice cadence, ISR storms compressing task lanes.

Event View — the searchable table

Open via Toolbox → **Event**. Every kernel (and custom) event as a row: timestamp, probe, type, object, parameters — decoded with names.



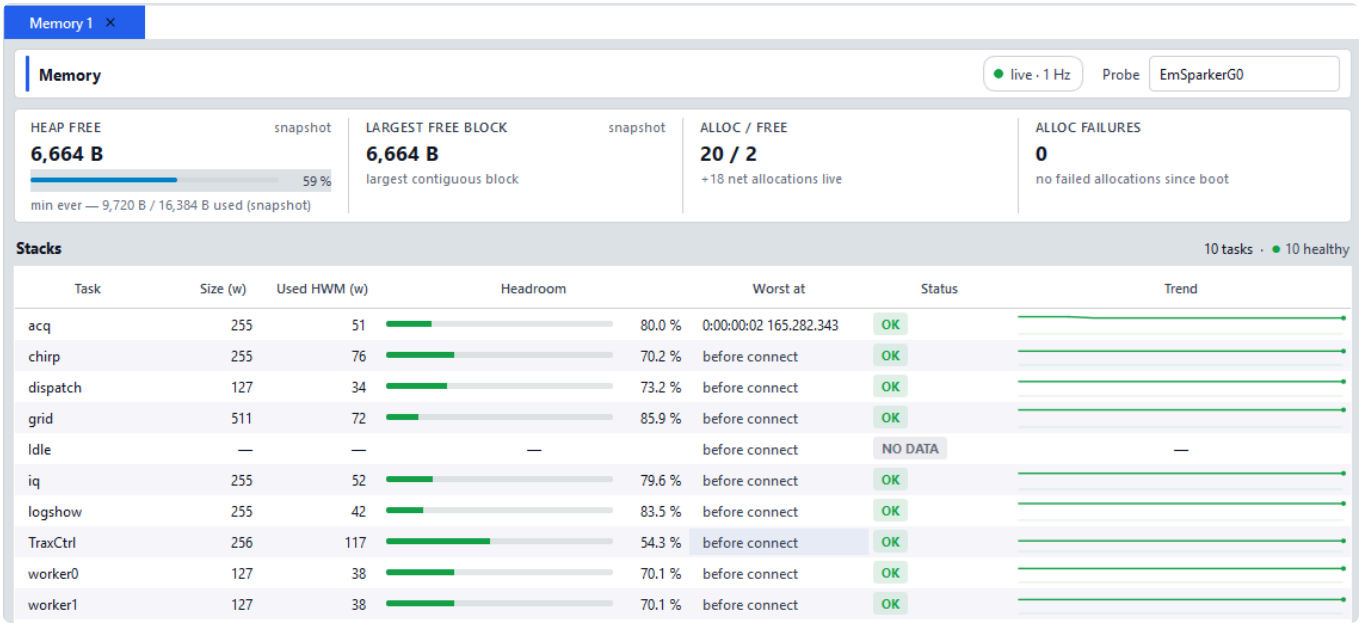
Screenshot: Event View rows with type filter popup open

- Live following or Review paging through history.
- **Text search** plus per-probe **event-type filters** (e.g. show only queue operations).
- Row selection broadcasts the timestamp via sync — jump the Trace View to any event.

Use the Trace View to see *shape* and the Event View to see *exact order and parameters*; they're complementary.

Memory View — heap & stack health

Open via Toolbox → **Memory**. A point-in-time dashboard (refreshed ~1 Hz) built from the heap/stack trace events:



Screenshot: Memory View, heap KPI cards on top, per-task stack table below

- **Heap KPIs** — current free, largest free block, allocation/free counts, allocation failures.
- **Per-task stack table** — current usage, worst case (high-water mark), trend, and a severity rating per task.

The Memory View intentionally shows *now*; for **time-series** of heap free or a task's stack headroom, add the corresponding memory channels in a Scope View (Chapter 11) and watch trends, place cursors, and correlate with the Trace View.

Troubleshooting

Symptom	Likely cause
No task lanes at all	`configUSE_TRACE_FACILITY` not 1, or `trax.h` not included at the bottom of `FreeRTOSConfig.h` , or `TRAX_CFG_RTOS_TYPE` not set
Tasks shown as raw handles	Task created before streaming started and not re-announced — create it from a running task after `trax_wait_stream_started()` (Chapter 02 §5.2); queues unnamed → add them to the queue registry
Stack columns empty	`INCLUDE_uxTaskGetStackHighWaterMark` not enabled
Timeline gaps under load	Ring buffer overflow — enlarge `TRAX_CFG_OUT_BUFFER_SIZE32` or reduce other traffic (Chapter 02 §7)