Enabling Task-Aware Timing Analysis on RISC-V via Pseudo Data Trace

2026-09-14 - Comments (0) - Trace

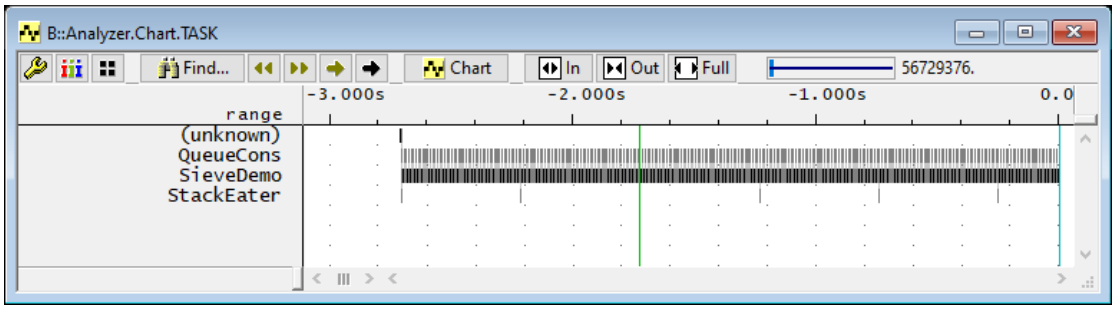
Analyzing timing behavior in OS-based applications necessitates tracing task context switches. This is especially valuable when task-switch tracing is performed in conjunction with program flow tracing, which enables the simultaneous analysis of functions within the context of the executing task. However, task-aware tracing requires either data-based context tracing or dedicated context-switch signaling—both of which are sometimes unavailable in the core architecture's trace infrastructure.

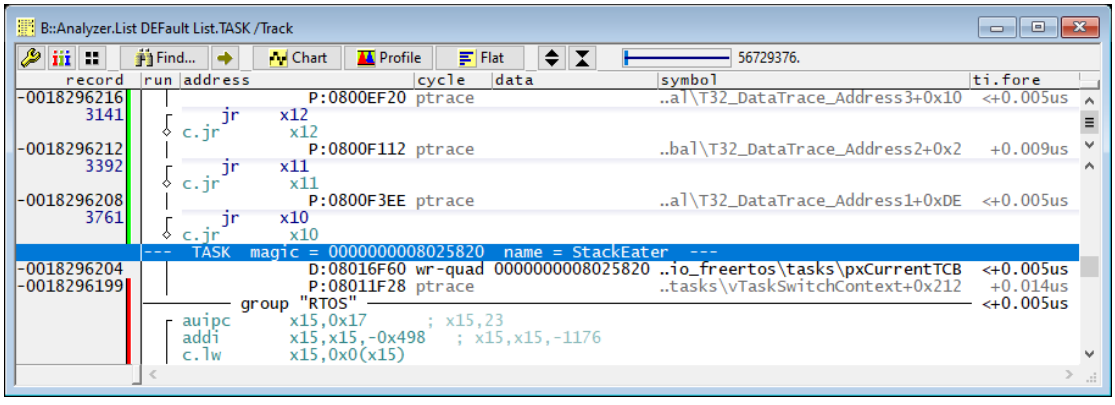
A practical example of this limitation can be found in the PolarFire SoC (RISC-V, 64-bit). When running FreeRTOS on a single U54 hart, the SoC provides program flow tracing but lacks the necessary data and context tracing mechanisms to support task-aware analysis. In this article, we present a solution based on Lauterbach's TRACE32 Pseudo Data Trace to overcome these limitations, enabling full task-aware timing analysis on the PolarFire SoC.

Pseudo Data Trace works by generating 'artificial' data cycles within the trace stream based on the existing program flow. This is achieved by executing a specific sequence of branch instructions on the target, which transmits the required context information through the trace infrastructure. The corresponding code is provided by Lauterbach. To implement this, the FreeRTOS scheduler must be patched to call a specific function, `T32_DataTrace_Write64()`, which encodes the address and value of the variable where the RTOS stores the currently executing task:

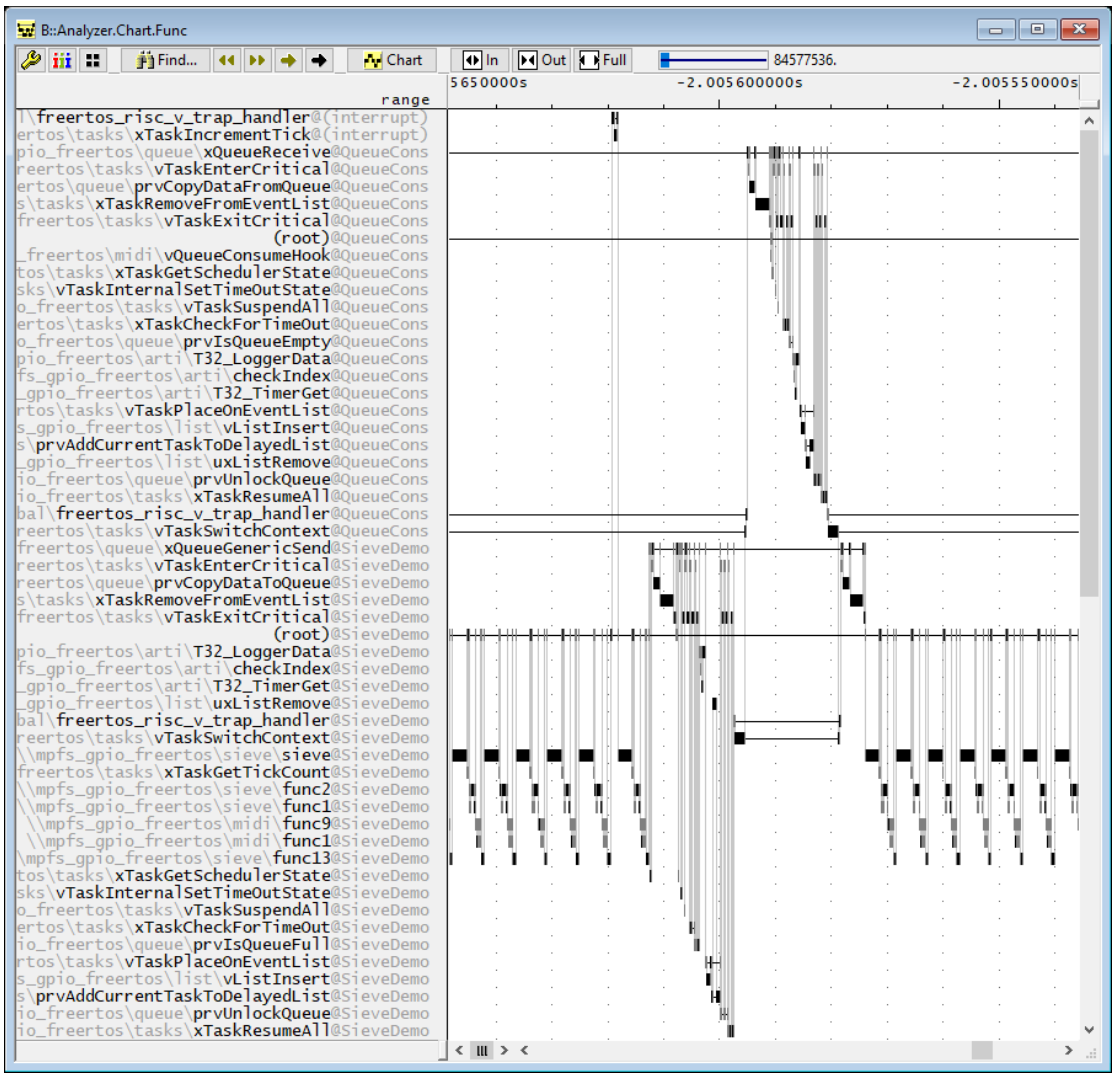
```
#define traceTASK_SWITCHED_IN() { \
                T32_DataTrace_Write64((void *) &pxCurrentTCB, (unsigned long long)
pxCurrentTCB);\
                }
```

As a result, task switches can be clearly visualized in the trace, as illustrated in the screenshots below:

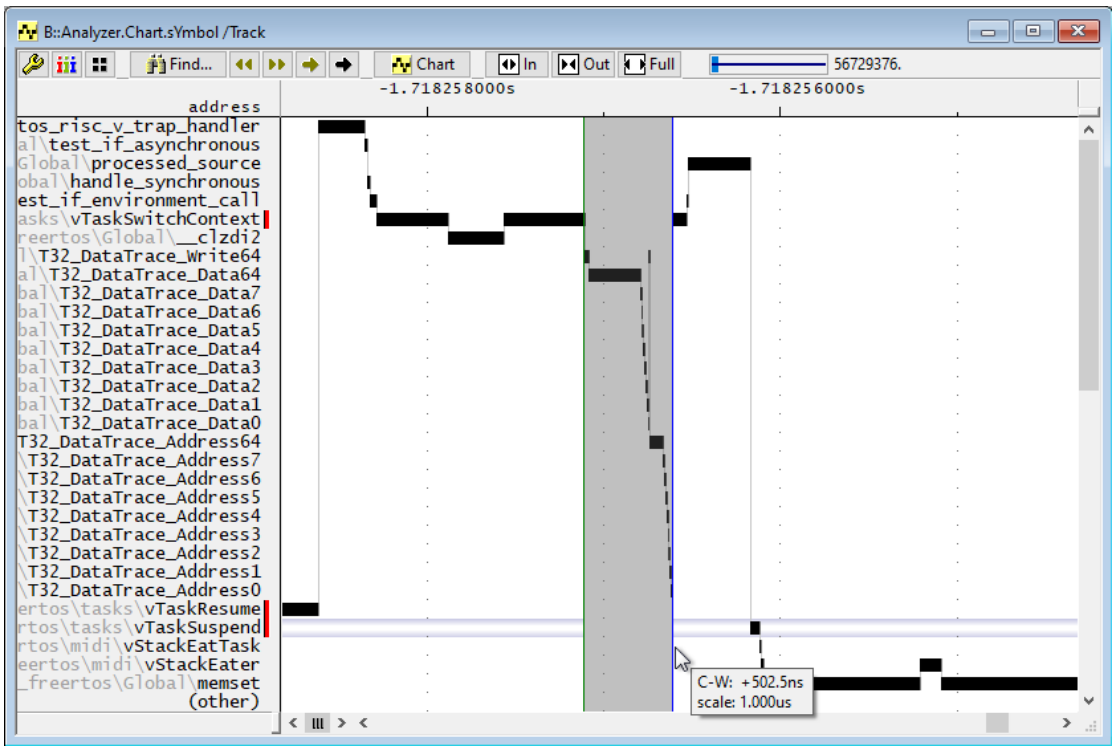




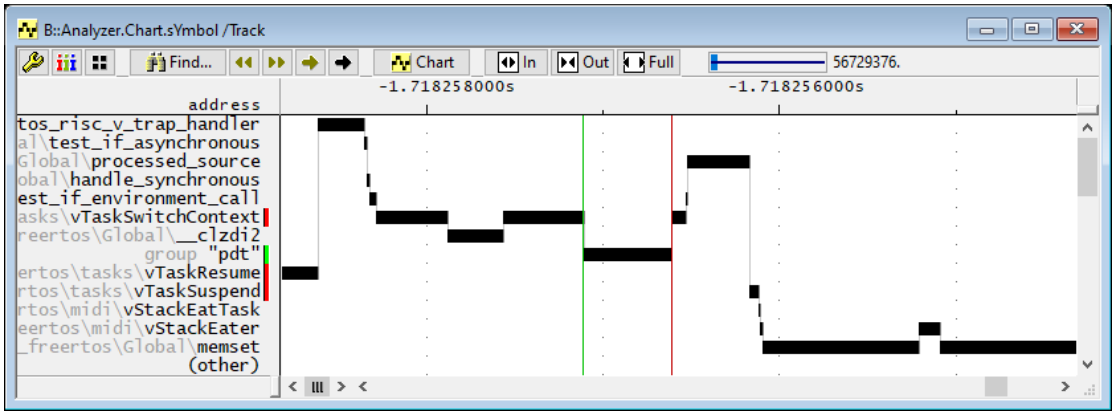
By correlating task identities with the executed program flow, users can further leverage TRACE32 commands for Function Run-Time Analysis. The screenshot below demonstrates this capability, showing the decoded program flow trace interleaved with the pseudo data trace to align function execution with the specific task context:



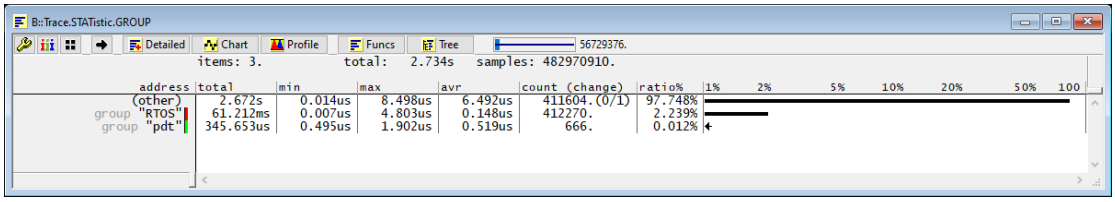
In the trace, the mechanism of the pseudo data trace is visible as a series of specific injected functions. This is illustrated in the screenshot below, which shows the decoded program flow trace alongside the pseudo data trace. This visualization demonstrates how the added data trace packets align with the execution of FreeRTOS functions



Because these functions are used solely by TRACE32 to decode the address and data, they do not carry independent analytical value. To maintain a clean trace, these functions can be grouped in the visualization using specific TRACE32 commands. The final result is shown in the following screenshot, where these functions are consolidated into a single **"pdt"** group marked with green:



Once the target code has been instrumented, it is important to quantify the runtime overhead introduced by this process. We can determine this by evaluating the execution time attributed to the**"pdt"** group within the program trace. For a hart running at 300 MHz, the overhead for a single task switch event is approximately 500 ns. The total impact on real-time execution is minimal, estimated to be on the order of 0.01%, as demonstrated in the following analysis:



Finally, it is important to note that the Pseudo Data Trace feature remains effective in multi-core RTOS environments. Because the trace protocol manages the association between task-switch events and the specific hart executing them, TRACE32 can accurately attribute each task switch to its respective core.

To test this feature or for further technical assistance, please contact Lauterbach support at https://support.lauterbach.com/new-ticket

Related Content:

- Trace support for PolarFire® SoC (RISC-V + FPGA)