

Analyzing real-time operating systems using trace techniques

By Jens Braunes, PLS

Code and data trace have absolutely no influence on the real-time behaviour of a system. This advantage can be used extensively when analyzing the runtime behaviour of real-time operating systems.



■ From experience, system observation is among the most critical aspects of testing and analyzing embedded software, because many factors may adversely affect runtime behaviour. One of them arises already during the build process: do we compile the application for debugging and test purposes or for production enabling additional optimizations for code size or speed? Profiling, which is frequently applied in order to optimize performance, also has a considerable influence. Quite often, the application needs to be instrumented for measuring the runtimes of functions and tasks. The added test code will have a small, but measurable influence on the application runtime behaviour. Even the memory layout may change due to the additional code, so that the influence cannot be neglected anymore. Among other variants of high-level system observation, monitoring is frequently used especially in the context of real-time operating systems. For this purpose, the monitor checks for RTOS state modifications that are typically mapped to specific memory locations. Triggered by interrupts which occur on task switches (among others), the monitor records the event for later analysis.

Those engineers who want to go without any instrumentation of the program code and without any monitors for analyzing the runtime behaviour of a real-time operating system (RTOS), should select a microcontroller

providing appropriate trace hardware including a high-bandwidth trace interface. MCU families offering suitable features are now available from almost all semiconductor vendors. Today, tracing capabilities are often a show-stopper for the selection of a platform. Tracing can be used to observe modifications of system states without influencing the system real-time behaviour. Depending on the controller manufacturer and the implemented trace architecture, there will be several trace modes which might be useful to analyze the runtime behaviour of an operating system. However, before going deeper into that, it is necessary to point out what information is required actually in the specific case.

In RTOS-controlled applications, the runtime of each specific task plays a significant role for evaluating execution performance. For instance, the task execution time directly indicates whether the system is running at a reasonable workload or in an overload situation. As another important factor, it is interesting to know how often tasks are interrupted by other tasks and how long tasks can operate without being interrupted. Due to the overhead incurred by task switches, this often bears significant potential for optimization. The same applies for the interrupt load which gives an indication of how often and how long task execution is halted by interrupts.

Obviously, code tracing is the method of choice for obtaining the required runtime information for all necessary measurements as simply and quickly as possible. Basically, tracing consists of recording the addresses of executed branches (more specifically of the branch targets) as well as any deviations from the regular, sequential execution flow as a result of calls, returns and interrupts. For task analysis, code tracing can be used to detect task switches that manifest themselves by multiple function calls and returns (including the scheduler), thus causing disruptions of the sequential execution flow.

However, the code trace is much too large for this purpose because trace data are also recorded for the execution of in-function code, including if-then-else constructs or loops. This increases the size of the trace and represents an unnecessary load for the on-chip or external trace memory. Infineon uses an interesting approach to avoid this dilemma. The so-called Compact Function Trace (CFT) of this company records only function calls and returns. The program flow within functions is not captured by the trace at all. Of course, that reduces dramatically the amount of trace data and saves trace memory. Data trace is another option complementing code tracing. Typically, RTOSs have their own management structures

Name	Value
TriCore_27x	
vs_SMP_NUMCPU	0x3
RUNNINGTASK	
[0]	/M_Core1
[1]	/M_Core1
[2]	/M_Core1
RUNNINGTASK	
SERVIC	
RUNNINGTASKPRIORITY	
CURRENTAPPMODE	OSDEFAULTAPPMODE
TASK_PWM_Core1	
TASK_1MS_Core1	
PRIORITY	"osTcbActualPrio[1]" not a valid expr
vs_HomePriority	11
STATE	READY
STACK	osTaskStackCore_1_1
vs_Schedule	FULL-Preempt
vs_TaskType	EXTENDED
vs_WaitMask	0x0
vs_EventFlag	0x0
CURRENTACTIVATIONS	0x1
vs_MaxActivations	1
CONTEXT	Context_TASK_1MS_Core1
TASK_JoHwAb_Core1	
TASK_Get_Check_Tempt_Core0	
TASK_BSW_MainFunction_Core0	
PRIORITY	"osTcbActualPrio[4]" not a valid expr
vs_HomePriority	2
STATE	READY
STACK	osTaskStackCore0_1
vs_Schedule	FULL-Preempt
vs_TaskType	EXTENDED
vs_WaitMask	0x0

Figure 1. Processed ORTI file

located in MCU memory reflecting the current system state and the task scheduling. This includes active tasks and information about the interrupt processing. Changes of the system state, including task switches,

thus always trigger a write access to these management structures. Some trace implementations allow data trace that can be used to record these write accesses. That enables a precise tracking and analysis of state changes

of the operating system. However, data tracing is quite expensive in terms of memory space. Without additional measures, all memory accesses of the application would be recorded. Similar to code tracing, this is undesirable here as well. Appropriate filters must therefore be applied to ensure that recording is exclusively restricted to relevant accesses targeting the management structures. Regardless of which trace mode is used for obtaining runtime information, unique, uniform timestamps are required here. Otherwise, it would be virtually impossible to get precise measurements and useable results.

How an analysis using trace data will work in the field can be demonstrated with an example of an OSEK-compliant operating system. OSEK is the abbreviation for "Offene Systeme und deren Schnittstellen für die Elektronik in Kraftfahrzeugen" (open systems and their interfaces for automotive electronics). It designates a standard for implementing real-time operating systems particularly in the automotive environment. The well-known Autosar approach represents an advanced development and reuse of the OSEK specification. The OSEK also defines an interface format called ORTI (OSEK Runtime Interface) for the communication of the operating system with analysis tools and debuggers. In a text file (the so-called ORTI file) this format describes all relevant internal operating system data, facilitating their use by tools and their visualization for users.

The debugger - the Universal Debug Engine (UDE) from PLS in this case - extracts the data structure from the ORTI file containing for example the current task (figure 1). This information is used for configuring the trace filter, ensuring that only data trace is recorded

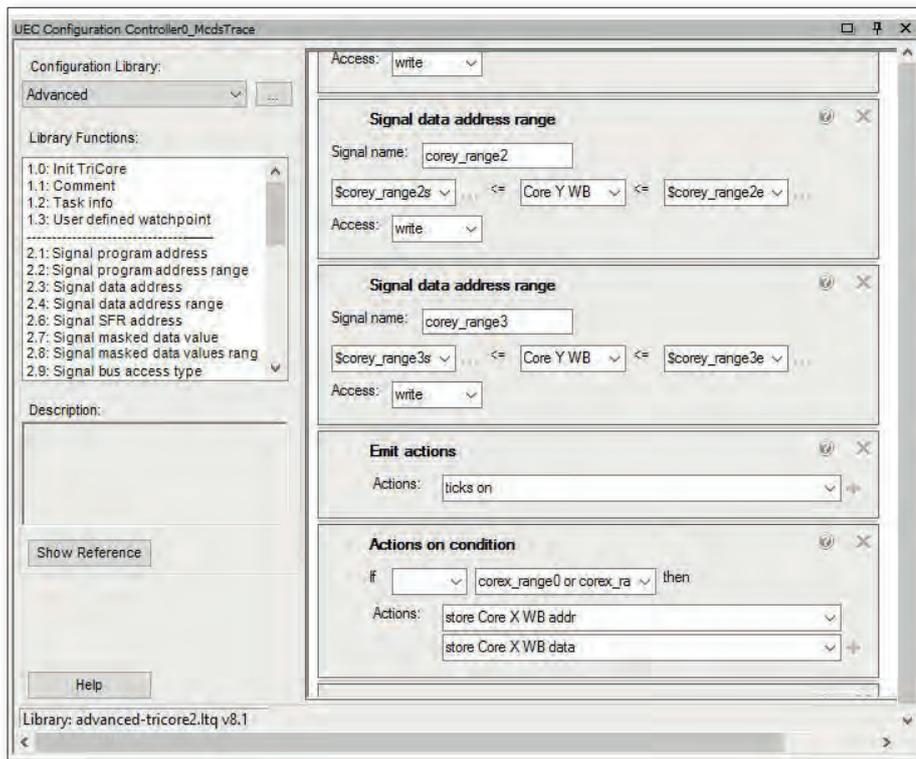


Figure 2. Configuration for a data trace based on ORTI data

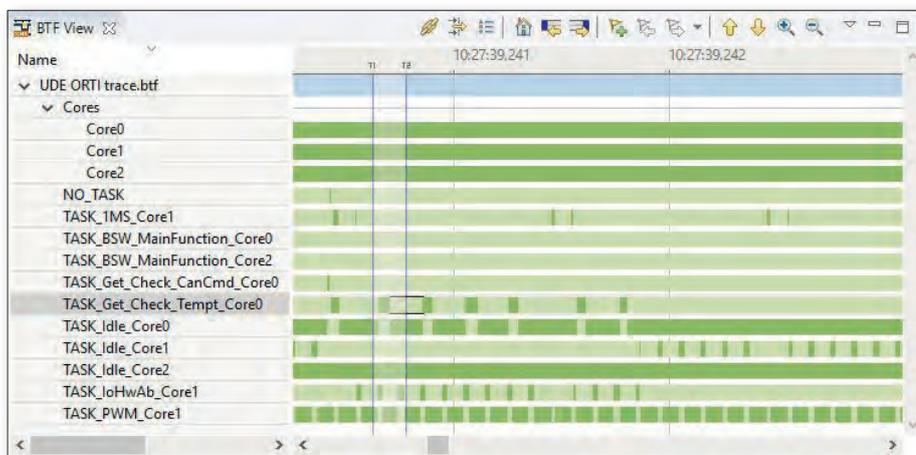


Figure 3. Visualization of operating-system tasks in Eclipse Trace Compass

actually relevant for task analyses (figure 2). In the next step, the trace is preprocessed to allow a proper visualization of the sequences of executed operating-system tasks including their precise timing. Specialized tools like Eclipse Trace Compass (www.tracecompass.org) can be used afterwards for sophisticated visualizations and additional analyses (figure 3).

For this purpose, the UDE provides an export function based on the BTF format (Best Trace Format) defined by Timing Architects, Inc. BTF, which was specifically developed as an exchange format for event traces, is used in many simulation, profiling and trace analysis tools. As demonstrated by this example, a combined tool solution consisting of a debugger and hardware tracing is now a genuine

alternative to collecting runtime information using instrumentation or special monitors. Even though hardware trace will often operate too finely-grained - it will typically record the entire control flow including in-function instructions and will thus be quite memory-hungry - developments like CFT or the use of ORTI for data tracing can quickly compensate for this disadvantage.

All in all, tracing provides a method for collecting runtime information in a fully reactionless and highly precise manner. And not least, it can also be used for examinations at the task and operating-system level. ■

Product News

Hall-Stand 4929

Keysight to address IoT, digital, RF test challenges at embedded world 2017

Keysight Technologies will exhibit a number of new products at embedded world 2017. Keysight's technical experts and application engineers will be available at the event to demonstrate the company's latest design, verification and test solutions. The solutions are focused on the general electronics, automotive, communications, education and energy industries and were designed to help embedded engineers solve today's toughest test challenges. This includes IoT projects, digital and RF communication, and (low) power requirements. Demonstrations at the booth will include the following comprehensive set of hardware and software-based solutions:

Entire range of low to high-end oscilloscopes – showing the latest solutions for debugging low- and high-speed serial buses and the testing of communication signals. This includes USB-C, NFC and wireless charging application support.

RF and wireless communication solutions – including the EMI/EMC aspects of the IEEE 802.11 protocol (WiFi), Bluetooth and ZigBee channels. Simulating and testing these wireless channels, which are used in a wide range of embedded applications, have become a key, and essential step in the embedded design process.

New low power and ultra-low power measurement solutions – which are relevant to embedded designers working on IoT, semiconductor, automotive and wireless applications. This includes the CX3300A, industry's first 100pA dynamic current analyzer with 14/16-bit resolution and up to 200 MHz bandwidth and up to 1 GSa/s sampling rate. New hardware and software tools for signal integrity analysis and verification – including Keysight's new simulation tools SIPro and PIPro, allowing designers to investigate signal integrity and power integrity at the design stage. This also includes a new test solution for evaluating thermal issues on the board as well as measuring TDR. New IoT measurement tools – including a teaching kit with courseware to assist universities in building competencies to match industry needs in product design, validation and testing, and manufacturing.

General purpose instruments – offering a comprehensive portfolio of bench and PXI instruments for precise and reliable measurement needs.

News ID 4929