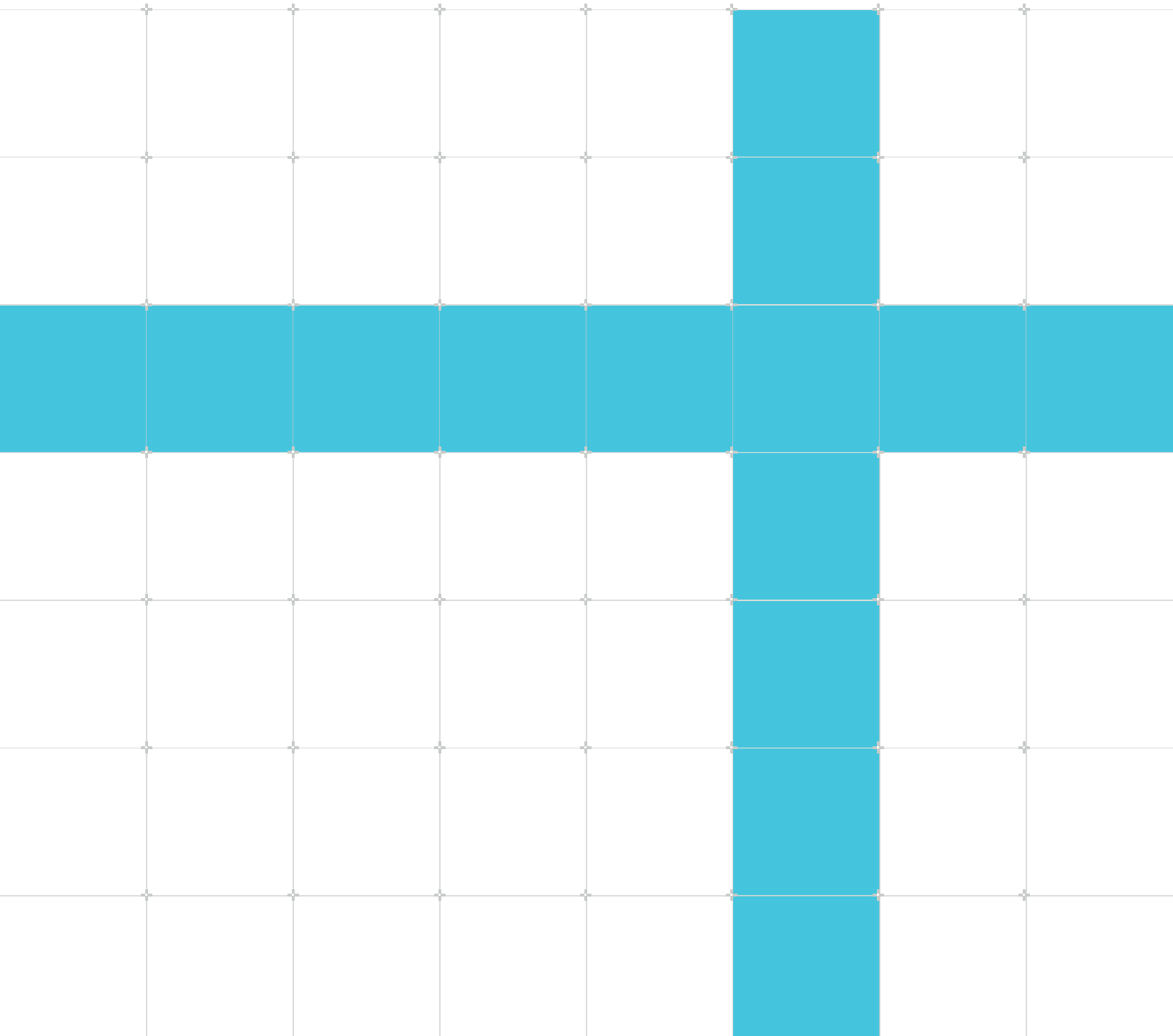
Arm Functional Safety Run-Time System

Version 1.0

Application Note

Issue
KAN345_1.0_en



Arm Functional Safety Run-Time System

Application Note

Release information

Document history

Issue	Date	Confidentiality	Change
1.0	21 December 2022	Non-Confidential	Initial release

Proprietary Notice

and regulations to assure that this document or any portion thereof is not exported, directly or indirectly, in violation of such export laws. Use of the word "partner" in reference to Arm's customers is not intended to create or refer to any partnership relationship with any other company. Arm may make changes to this document at any time and without notice.

This document may be translated into other languages for convenience, and you agree that if there is any conflict between the English version of this document and any translation, the terms of the English version of the Agreement shall prevail.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

(LES-PRE-20349|version 21.0)

Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by Arm and the party that Arm delivered this document to.

Unrestricted Access is an Arm internal classification.

Product Status

The information in this document is Final, that is for a developed product.

Feedback

Arm® welcomes feedback on this product and its documentation. To provide feedback on the product, create a ticket on https://support.developer.arm.com

To provide feedback on the document, fill the following survey: https://developer.arm.com/documentation-feedback-survey.

Inclusive language commitment

Arm values inclusive communities. Arm recognizes that we and our industry have used language that can be offensive. Arm strives to lead the industry and create change.

We believe that this document contains no offensive language. To report offensive language in this document, email terms@arm.com.

Contents

1. Introduction

1.1 Conventions

The following subsections describe conventions used in Arm documents.

Glossary

The Arm Glossary is a list of terms used in Arm documentation, together with definitions for those terms. The Arm Glossary does not contain terms that are industry standard unless the Arm meaning differs from the generally accepted meaning.

See the Arm® Glossary for more information: developer.arm.com/glossary.

Typographic conventions

Arm documentation uses typographical conventions to convey specific meaning.

Convention	Use
italic	Citations.
bold	Interface elements, such as menu names. Terms in descriptive lists, where appropriate.
`monospace`	Text that you can enter at the keyboard, such as commands, file and program names, and source code.
`monospace underline`	A permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.
`<and>`	Encloses replaceable terms for assembler syntax where they appear in code or code fragments. For example: `MRC p15, 0, <Rd>, <CRn>, <CRm>, <Opcode_2>`
SMALL CAPITALS	Terms that have specific technical meanings as defined in the *Arm® Glossary*. For example, **IMPLEMENTATION DEFINED**, **IMPLEMENTATION SPECIFIC**, **UNKNOWN**, and **UNPREDICTABLE**.
Caution	Recommendations. Not following these recommendations might lead to system failure or damage.
Warning	Requirements for the system. Not following these requirements might result in system failure or damage.
Danger	Requirements for the system. Not following these requirements will result in system failure or damage.
Note	An important piece of information that needs your attention.

Convention	Use
Tip	A useful tip that might make it easier, better or faster to perform a task.
Remember	A reminder of something important that relates to the information you are reading.

1.2 Useful resources

This document contains information that is specific to this product. See the following resources for other useful information.

Access to Arm documents depends on their confidentiality:

- Non-Confidential documents are available at developer.arm.com/documentation. Each document link in the following tables goes to the online version of the document.
- Confidential documents are available to licensees only through the product package.

Note

Arm tests its PDFs only in Adobe Acrobat and Acrobat Reader. Arm cannot guarantee the quality of its documents when used with any other PDF reader.

Adobe PDF reader products can be downloaded at http://www.adobe.com

1.3 Other information

See the Arm website for other relevant information.

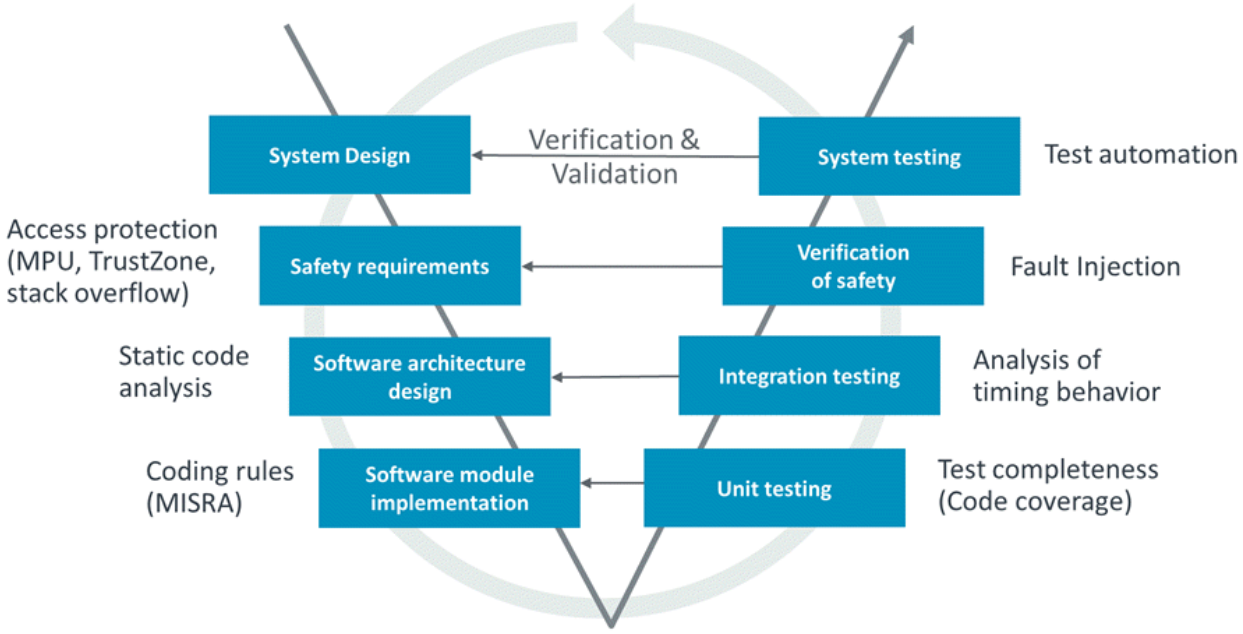
- Arm® Developer.
- Arm® Documentation.
- Technical Support.
- Arm® Glossary.

2. Functional Safety

Many products for such markets as household appliances, automotive, industrial and healthcare have regulatory requirements to be certified against functional safety (FuSa) standards. For example, IEC 61508 for electrical systems, ISO 26262 for the automotive industry, IEC62304 for medical systems, and EN 50128 for railway applications.

Within the standards, there are multiple safety integrity levels (SIL) that specify formal methods to be used during development for verifying that the application code, software components and toolchains are safe for the intended use.

Figure 2-1: The software development V model



To ensure the best outcomes for our technology and customers, Arm plays an active role in the development of international safety guidelines; for example, ISO 26262 and IEC 61508.

Arm products are designed “out of context” to satisfy the widest range of applications. We believe that everyone in the integrated circuit (IC) supply chain has an important role to play in safety certification, and applications must be certified in accordance with market-specific standards. Arm is capable of supporting customers and manufacturers in their certification processes of Arm-based devices.

The Arm Safety Ready portfolio is a collection of Arm products that have been through various and rigorous levels of functional safety systematic flows and development. It combines IP, safety features, tools, and robust methodologies to help reduce risk while fast-tracking the certification phase of projects.

MDK tools for FuSa development

Arm Keil MDK equips software engineers with professional tools that support the V-model development process and simplify creation, analysis and verification of complex embedded applications.

MDK Features	Description
Arm Compiler for Embedded FuSa	MDK-Professional provides access to safety-qualified Arm C/C++ compiler and its supporting documentation.
Static Code analysis and MISRA checking	MDK provides native integration with third-party code verification tools.
Code coverage	MDK with ULINKpro enables non-intrusive code coverage on target hardware visa streaming instruction trace.
Continuous integration	MDK has a command line interface for test automation and can be uses with Continuous Integration (CI) tools such as Jenkins.
Simulation models	MDK-Professional enables robust regression testing at function and module level using Arm Virtual Hardware (AVH).
RTOS-aware debugging	MDK provides full visibility into RTOS operation thus simplifying system debug and optimization.
Timing analysis	Event Recorder provides status details of software components and includes time information. Event Statistics show average, min and max execution times.

3. Arm FuSa RTS

Arm FuSa RTS is a set of embedded software components qualified for use in the most safety-critical applications in automotive, medical and industrial systems.

With FuSa RTS, developers receive a robust real-time operating system (RTOS), independent processor abstraction layer and verified C library that are highly optimized for Cortex-M processors by Arm architecture experts.

While being available as a separately licensable product, FuSa RTS perfectly integrates with Arm Keil MDK and is using the safety-qualified Arm C/C++ compiler to significantly simplify system design, development, validation and certification processes for safety applications.

Supported safety standards

Arm FuSa RTS is certified for the following safety standards:

- Automotive: ISO26262, ASIL D
- Industrial: IEC61508, SIL 3
- Railway: EN50128, SIL 4
- Medical: IEC62304, Class C

FuSa RTS safety compliance is confirmed by the TÜV Süd Certificate.

Supported devices

FuSa RTS fully utilizes advanced hardware features that Arm specifies for its processors. It provides support for devices with the following Arm Cortex-M cores:

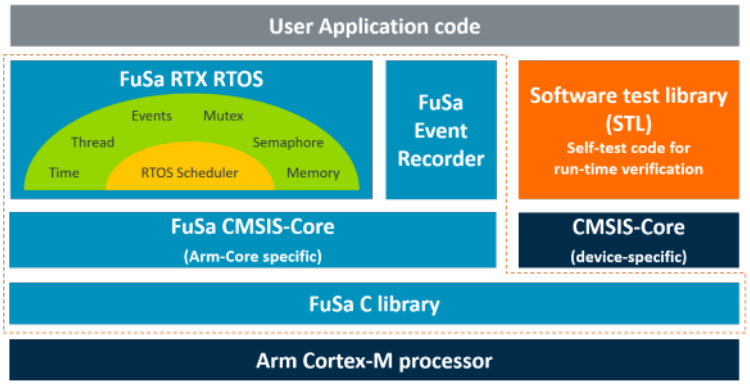
- Cortex-M0/M0+
- Cortex-M3
- Cortex-M4
- Cortex-M7

FuSa RTS components

Arm FuSa RTS package contains following components:

- FuSa RTX RTOS: deterministic real-time operating system for Arm Cortex-M processors.
- FuSa Event Recorder: implements functionality to easily record events and collect execution statistics in the application code.
- FuSa CMSIS-Core: validated vendor-independent software interface to the processor resources.
- FuSa C library: a subset of the C library functions suitable for developing safety-critical embedded applications.
- Safety Package: documentation set explaining the usage of FuSa RTS in safety context.

Figure 3-1: FuSa RTS Components



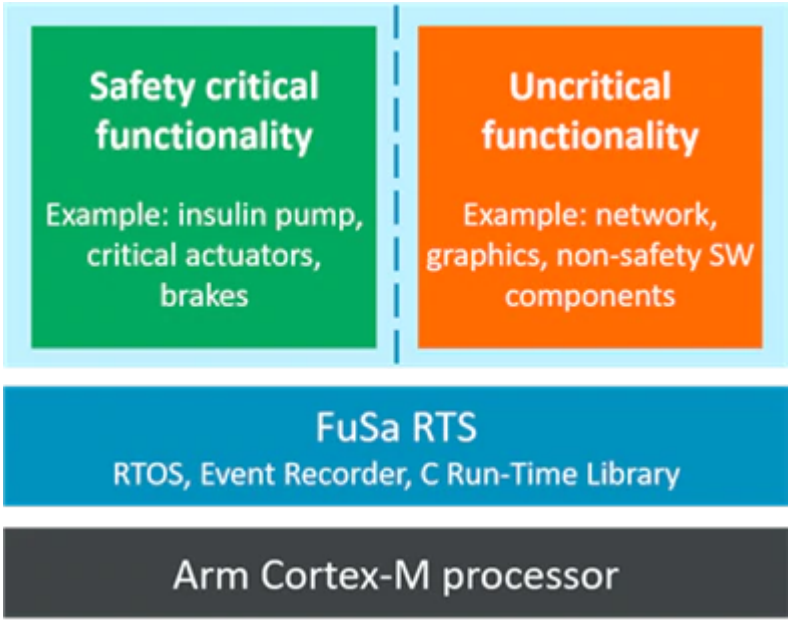
Process Isolation

FuSa RTS contains protection mechanisms that control access to system resources (such as memory, peripherals, processor execution time). These process isolation capabilities prevent undesired interference between software elements of different safety integrity levels and allow building of mixed-criticality systems on a single-core microcontroller.

FuSa RTS Process isolation is achieved with the following features:

Feature	Description
Spatial Isolation	Spatial isolation is enforced by MPU Protected Zones that use processor's Memory Protection Unit (MPU) to shield access to memory and peripherals. Access to RTOS objects and Kernel operations is additionally controlled with assigned Safety Classes.
Temporal Isolation	Temporal isolation is enabled with Thread Watchdog mechanisms that control the timing constraints in the system.
Controlled System Recovery	Controlled system recovery provides control over system operation in case of a failure and enables blocking the execution of non-safety components or proceeding to a safety state.

Figure 3-2: FuSa RTX Process Isolation



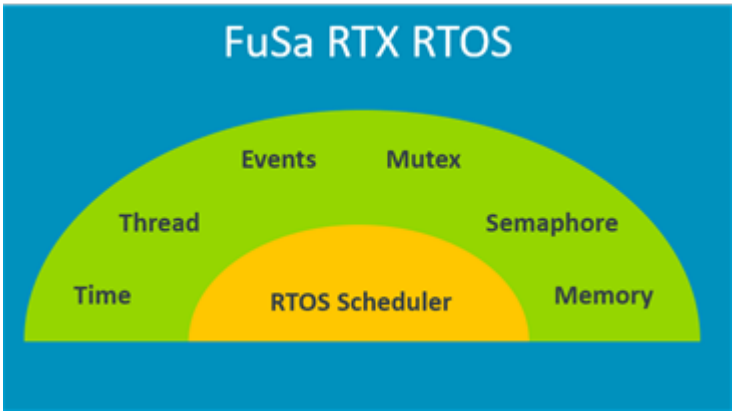
3.1 FuSa RTX RTOS

The use of a real-time operating system (RTOS) in a safety-critical system demands that the RTOS component also undergoes rigorous verification. In cases when regulatory certification is mandatory this also implies specific documentation and testing processes for the targeted safety standards.

To enable and streamline the product safety certification, Arm provides FuSa RTX RTOS as part of FuSa RTS package, that is qualified for use in automotive, industrial, railway and medical applications:

- It is a deterministic real-time operating system (RTOS) that reliably manages multiple application threads with priority-based, pre-emptive scheduling.
- It offers all services needed in complex real-time applications, such as threads, timers, memory and object management, message exchange and others.
- The kernel is highly optimized for Cortex-M architecture and has multiple provisions that naturally improve the reliability of an embedded application.

Figure 3-3: FuSa RTX RTOS Components



Strictly validated code

- MISRA C rules: RTX is written in C using C99 language extensions with MISRA C:2012 guidelines being applied to it.
- Safety compliance: FuSa RTX code has gone through stringent safety analysis and rigorous testing. It is approved for use in applications with the most demanding safety integrity levels (SIL). See FuSa RTS for the list of applicable safety standards.

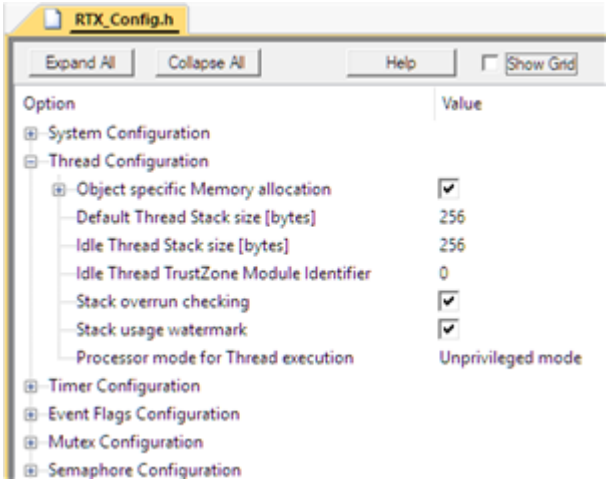
Designed for engineering efficiency

- Small memory footprint: requires minimum amount of system memory, starting from 5 KB ROM
- Low-power mode: has tick-less operation mode for low power devices

Easy to configure and use

- CMSIS-pack support: FuSa RTX is provided as a CMSIS component and can be easily managed in a µVision Run-Time Environment dialog.
- Configuration Wizard support: FuSa RTX provides a number of configuration parameters for the kernel operation as well as for the RTX objects such as threads, mutex and semaphores. Integrated support of MDK Configuration Wizard makes the parameter settings clear and intuitive.

Figure 3-4: RTX RTOS Configuration Wizard Window



Reliable execution

- Time-deterministic interrupt execution: RTX utilizes the `LDEX/STEX` instruction available on most Cortex-M processors and therefore user interrupts are never disabled

Safe operation

- Separate stacks for ISR/RTOS and threads: the RTOS kernel executes in handler mode with stack separated from user threads which avoids unexpected stack loads.
- Stack overflow checking: RTX implements a software stack overflow checking that traps stack overruns.
- Runtime check of kernel objects: object identifiers are validated at run-time for type-mismatches and are protected from inadvertently accesses by the user application.

Flexible memory management

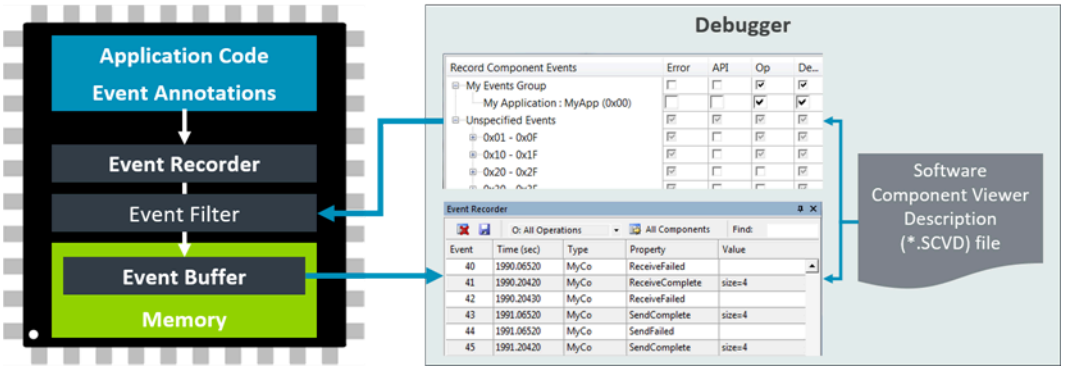
- Object-specific memory pools: dedicated fixed-size memory blocks for each object type avoids memory fragmentation during run-time and makes object creation and destruction time deterministic.
- Static object memory allocation: the user application may rely on static memory for kernel objects, which guarantees that the RTOS system can never run out of storage during run-time.

3.2 FuSa Event Recorder

Event Recorder provides an API (function calls) for event annotations in the application code. These functions record events along with timestamps and additional information. The data is stored in the event buffer located in the RAM of the target hardware.

The µVision debugger reads the content of the event buffer and displays it in the Event Recorder window. The graphical display over time is available in the System Analyzer window. Other timing and power data can be observed in the Event Statistics window.

Figure 3-5: Event Recorder Block Diagram



Event Recorder Benefits

- Visibility to the dynamic execution of an application at little (memory) cost.
- Adding RTOS awareness to a development tool does not require complex DLL programming.
- For Arm Cortex-M3/M4/M7/M33 processor based devices, Event Recorder functions will not disable interrupts.
- Adding printf re-targeting for devices without ITM, such as Arm Cortex-M0/M0+/M23.
- Fast time-deterministic execution of event recorder functions with minimal code and timing overhead.
- No need for a debug or release build as the event annotations can remain in production code.
- Saving the event data in local memory ensures fast recording.
- Collecting the data from the on-chip memory is done using simple read commands. These commands work on all Cortex-M processor based devices and require only JTAG or SWD connectivity to the debug adapter.
- Using the DWT Cycle Count register for creating time stamps reduces code overhead (available on Arm Cortex-M3/M4/M7/M33M55/M85).

3.3 FuSa CMSIS-Core

CMSIS-Core implements the basic run-time system for a Cortex-M device and gives the user access to the processor core and the device peripherals.

CMSIS-Core defines:

- A Hardware Abstraction Layer (HAL) for Cortex-M processor registers with standardized definitions for the SysTick, NVIC, System Control Block registers, MPU registers, FPU registers, and core access functions.
- System exception names to interface to system exceptions without having compatibility issues.
- Methods to organize header files that makes it easy to learn new Cortex-M microcontroller products and improve software portability. This includes naming conventions for device-specific interrupts.

- Methods for system initialization to be used by each MCU vendor. For example, the standardized SystemInit() function is essential for configuring the clock system of the device.
- Intrinsic functions used to generate CPU instructions that are not supported by standard C functions.
- A variable to determine the system clock frequency which simplifies the setup the SysTick timer.

3.4 FuSa C Library

The FuSa C library is a subset of the standard C library that consists of approximately 200 functions that have been specifically implemented and optimized for use in safety development.

The FuSa C library comes with:

- Certificate from TÜV SÜD and supports the same functional safety standards as Arm Compiler for Embedded FuSa.
- Qualification Kit which contains the Safety Manual and Defect Report.

3.5 FuSa RTS Evaluation

Evaluation version of Arm FuSa RTS allows you to analyze the software components and documentation provided in the FuSa RTS. The access channel and content packages are organized in the same manner as for the commercial version.

Deliverables

The evaluation version of Arm FuSa RTS consists of the following deliverables:

- Software pack file
- Safety package

The evaluation content is slightly different from the one delivered with the commercial version. It is based on the FuSa RTS for Cortex-M3, but the structure and components are very similar for other cores. Only the low-level processor abstraction files are different. More details about the content and installation are provided in sections below.

Software pack file

Arm FuSa RTS Evaluation software is provided as components in a CMSIS Software Pack that has .pack file extension. The CMSIS pack format is supported by multiple IDEs, but the easiest way is to use it with Arm Keil MDK.

Differences to the commercial version pack file

- FuSa C library and header files are not included
- Example application is slightly modified to compile without FuSa C library

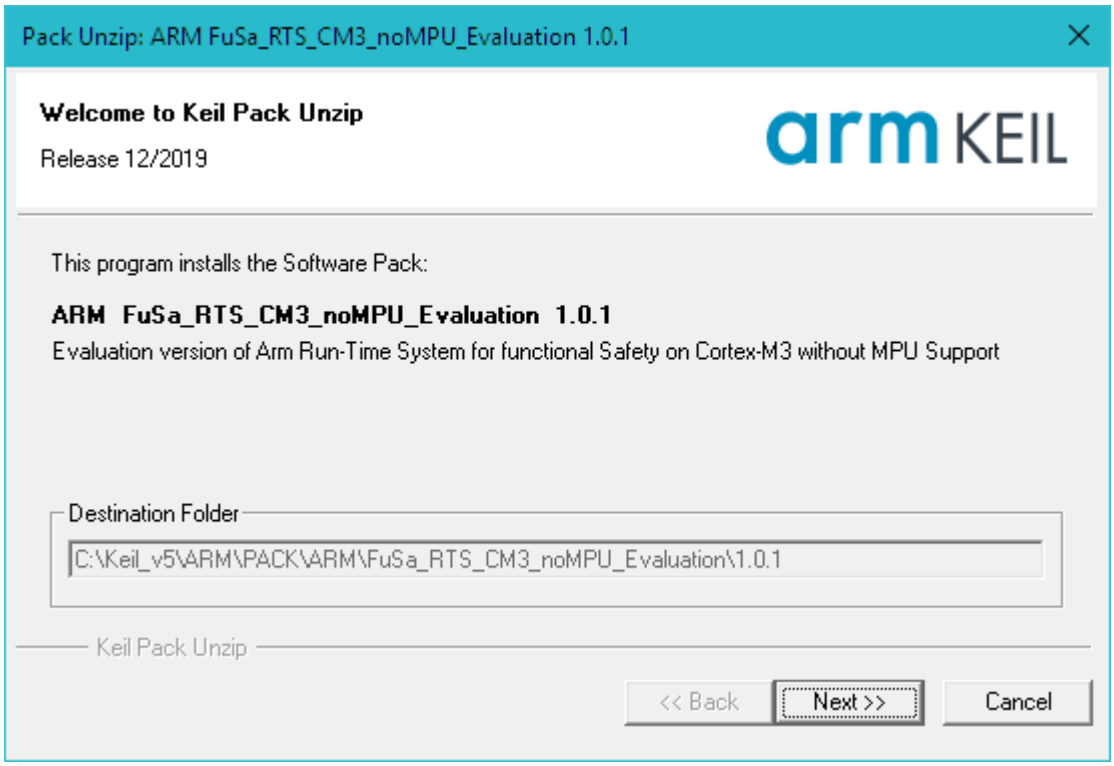
- Evaluation version of safety manual is provided.

Installation

To install the FuSa RTS software pack in Keil MDK:

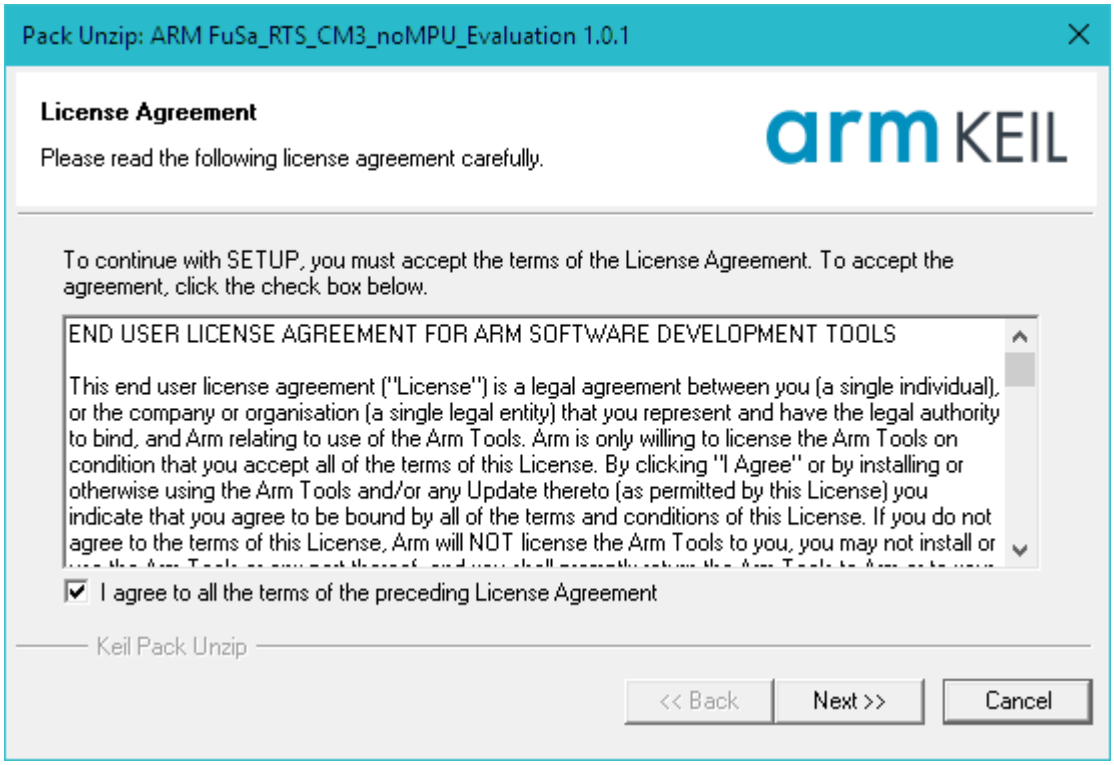
1. Verify that Keil MDK is installed on your PC.
2. Extract the .pack file from the downloaded .zip archive.
3. Double-click on the FuSa RTS Evaluation software pack file (.pack extension). Provide the destination path where the pack will be installed. Click Next ». Alternatively you can open the Pack Installer (typically available at `C:\Keil_v5\UV4\PackInstaller.exe`) and there use the menu item File - Import...

Figure 3-6: Pack Unzip: Welcome



4. Read and accept licensing terms, then press Next %raquo; button.

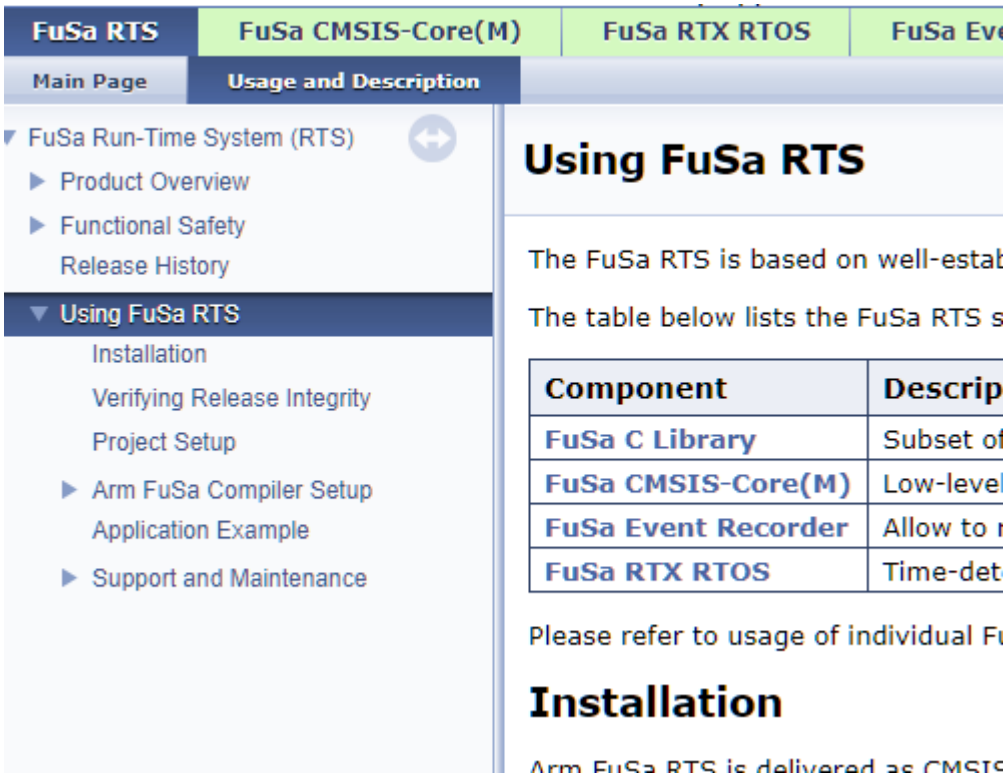
Figure 3-7: Pack Unzip: License Agreement



5. The content of the software pack is extracted to the directory `<MDK_Root>\ARM\PACK\ARM\<PackName>\<PackVersion>`, for example: `C:\Keil_v5\ARM\PACK\ARM\FuSa_RTS_CM3_noMPU_Evaluation\1.0.1\`.

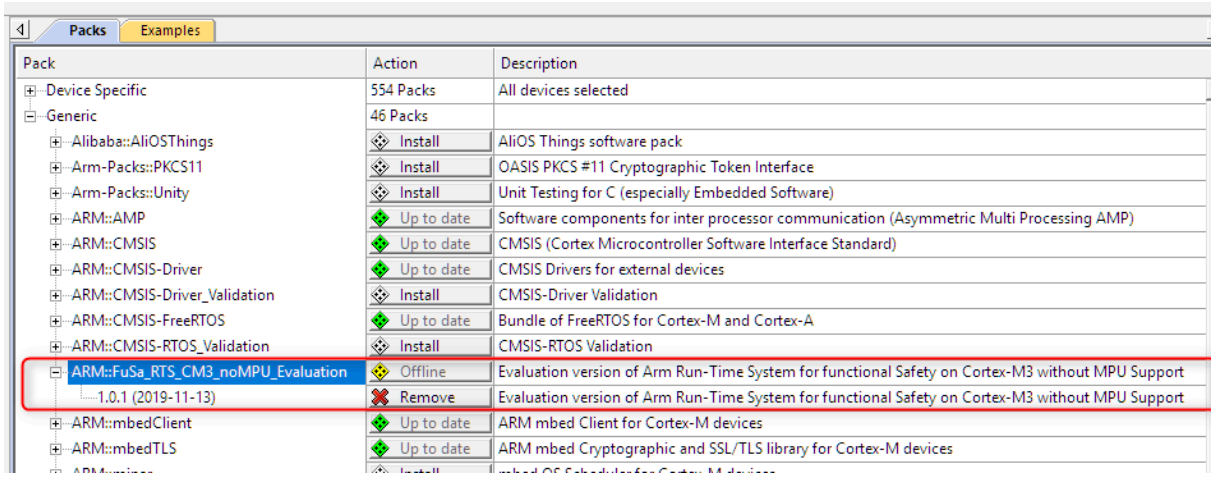
   From this folder you can also access user and safety manual by clicking on SafetyManual.html file. Its section “Using FuSa RTS” provides additional details on how to verify the package integrity, add FuSa RTS components to a uVision project, add safety-qualified compiler, etc. It also describes an example included with the FuSa RTS package.

Figure 3-8: FuSa RTS Documentation



6. The FuSa RTS Evaluation pack can be also seen in the Pack Installer under Generic Packs as for example shown on the figure below:

Figure 3-9: FuSa RTS Documentation



Example application

The Blinky application provided in the evaluation FuSa RTS pack has been slightly modified to work without the FuSa C library.

- By default, the example is configured to use Arm safety-qualified compiler 6.6.2 (same as used for testing and certifying Arm FuSa RTS).
- It is also possible to switch to a default Arm Compiler 6.xx provided with Keil MDK and compile the example. The warning `.\Flash\Blinky.axf: Warning: L6092W: Section startup_armcm3.o(.gnu.linkonce.common) has deprecated prefix '.gnu.linkonce'. Support for legacy common sections shall be removed in a future version of the linker. Please use COMDAT groups as a replacement.` shall be ignored.

Safety Package

FuSa RTS Evaluation Safety Package is provided in form of a .zip file that can be simply extracted. It contains following documentation:

- FuSa RTS Qualification Package Overview - lists the content of the Safety Package in commercial version.
- FuSa RTS Evaluation safety manual - corresponds to the manual included in the FuSa RTS software pack
- License terms for FuSa RTS Safety Package are provided in `/license_terms/` directory.

Differences to the commercial version safety package

- TÜV Süd documents are available only with the commercial version. But they are referenced in the FuSa RTS Qualification Package Overview document and certificates can be verified in TÜV Süd certificate explorer.
- Safety manual is provided in evaluation version with following differences
  - Details of some safety requirements are not fully included. However, the brief description for all safety requirements is present.
  - List of known issues for FuSa C library is not included

Defect Report

The defect report is an HTML-based document that lists issues discovered for already released FuSa RTS revisions. It is not included in the evaluation version.

The issues known at the moment of the release can be found in the Release History section of the Safety Manual.

4. RTOS-aware Debugging

Using a real-time operating system (RTOS), significantly simplifies development and maintenance of complex embedded applications with multiple parallel tasks. However, due to increased complexity and the use of 3rd-party software components, it can become challenging to analyze the operation of RTOS-based programs using classic code debug techniques.

Keil RTX5 integrates multiple mechanisms that provide full visibility into RTOS operation and thus speed up debugging of potential problems and assist in program optimization.

RTX RTOS Component Viewer Window

Keil RTX5 supports Component Viewer and provides all key information about current RTOS state.

Detailed status information

RTX5's system configuration, its operation status and details about all allocated objects are displayed in the RTX RTOS window in µVision. The data is updated at run-time and is easy to browse, understand and analyze.

Always available

The RTX RTOS view is available when RTX5 is used in the project and doesn't require any special configuration. It works with all Cortex-M targets and with any debug adapter.

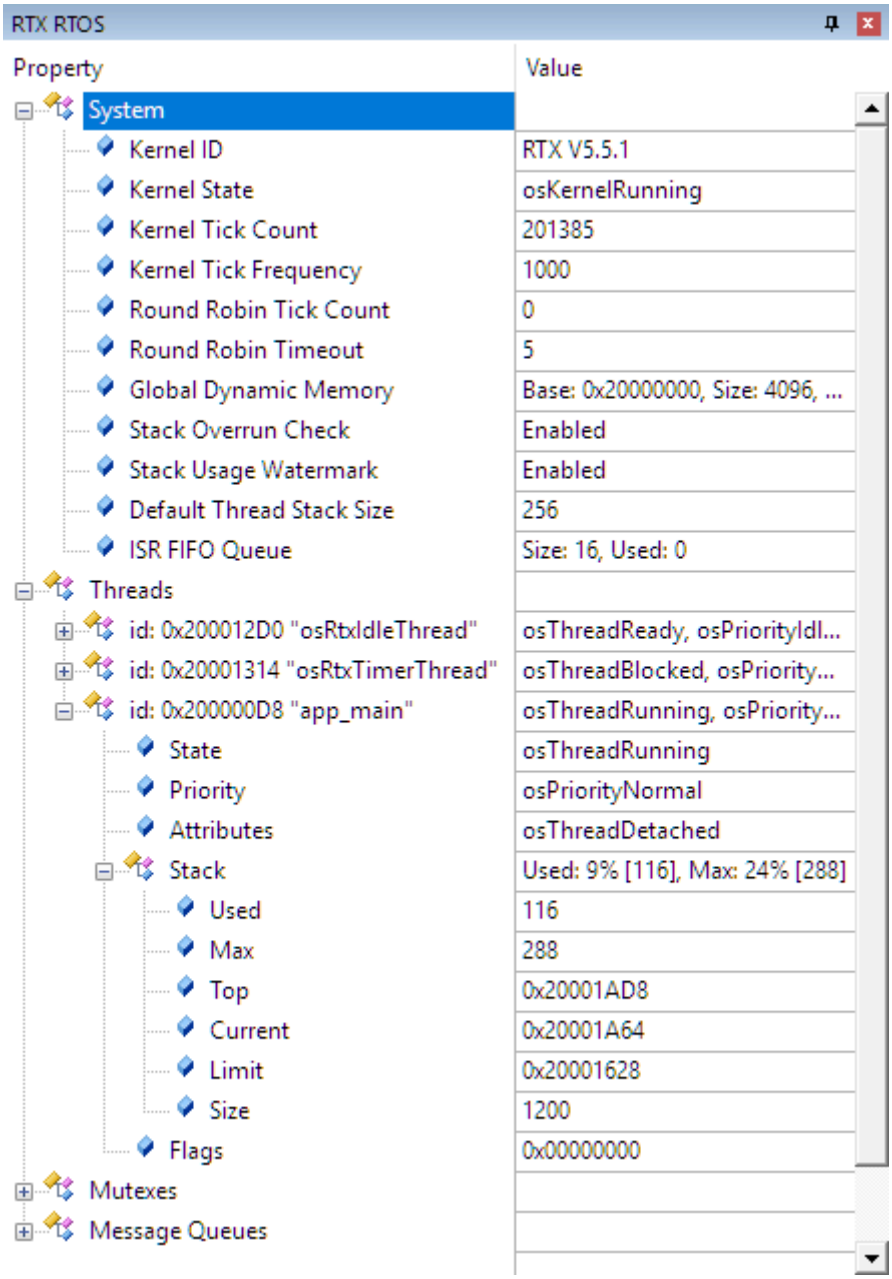
Stack usage analysis

Application developers can observe in real-time the actual stack usage for individual threads. The stack watermarking feature even shows the current maximum stack usage. This allows to optimize the RAM usage and avoid stack overflows.

Object memory usage counters

The maximum memory usage for each RTX5 object is displayed which enables fine-tuning of memory resources.

Figure 4-1: RTX RTOS Component Viewer Window



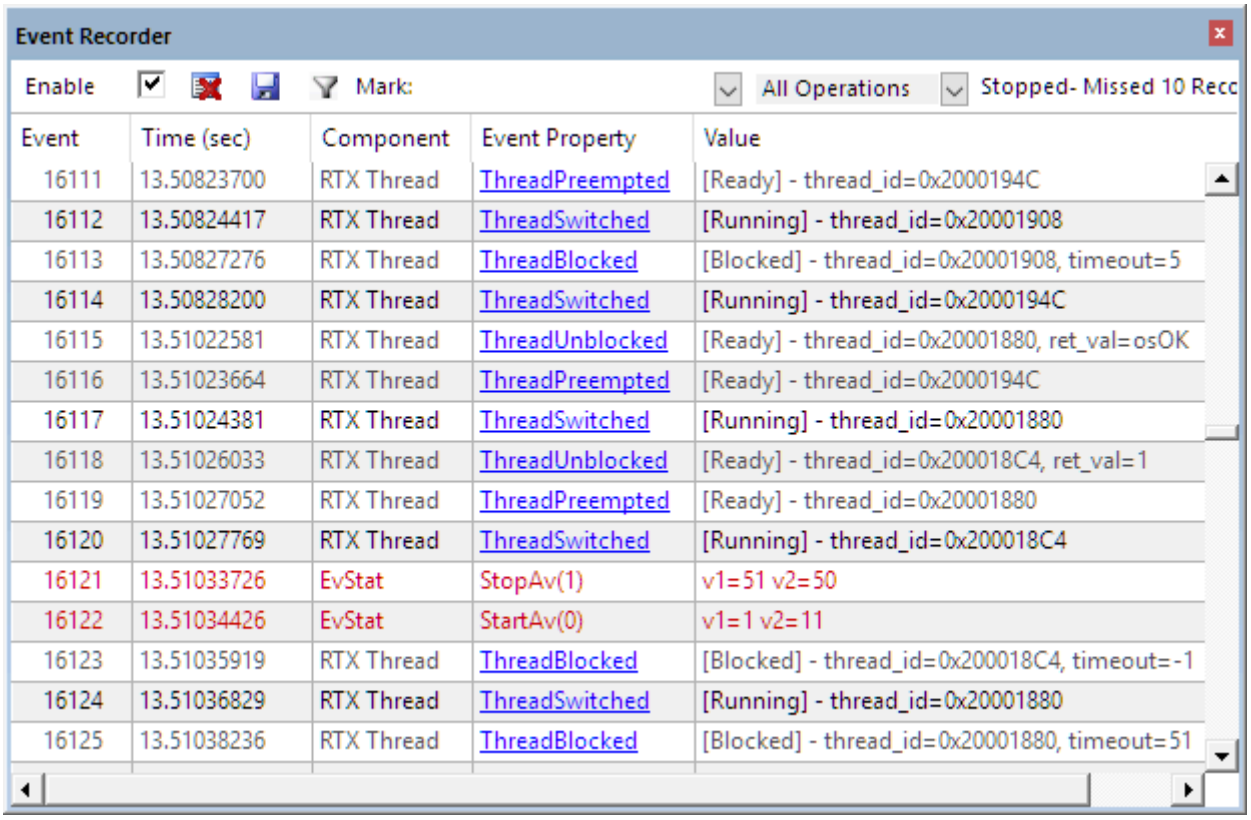
Event Recorder Support

RTX5 is annotated with more than 170 events for use with the Event Recorder.

The System Analyzer view in µVision graphically displays thread operation over time.

Event details which include accurate timestamps are provided in the Event Recorder window. You can filter the events to capture and display only the target events of interest.

Figure 4-2: RTX RTOS Event Recorder Window



Event	Time (sec)	Component	Event Property	Value
16111	13.50823700	RTX Thread	ThreadPreempted	[Ready] - thread_id=0x2000194C
16112	13.50824417	RTX Thread	ThreadSwitched	[Running] - thread_id=0x20001908
16113	13.50827276	RTX Thread	ThreadBlocked	[Blocked] - thread_id=0x20001908, timeout=5
16114	13.50828200	RTX Thread	ThreadSwitched	[Running] - thread_id=0x2000194C
16115	13.51022581	RTX Thread	ThreadUnblocked	[Ready] - thread_id=0x20001880, ret_val=osOK
16116	13.51023664	RTX Thread	ThreadPreempted	[Ready] - thread_id=0x2000194C
16117	13.51024381	RTX Thread	ThreadSwitched	[Running] - thread_id=0x20001880
16118	13.51026033	RTX Thread	ThreadUnblocked	[Ready] - thread_id=0x200018C4, ret_val=1
16119	13.51027052	RTX Thread	ThreadPreempted	[Ready] - thread_id=0x20001880
16120	13.51027769	RTX Thread	ThreadSwitched	[Running] - thread_id=0x200018C4
16121	13.51033726	EvStat	StopAv(1)	v1=51 v2=50
16122	13.51034426	EvStat	StartAv(0)	v1=1 v2=11
16123	13.51035919	RTX Thread	ThreadBlocked	[Blocked] - thread_id=0x200018C4, timeout=-1
16124	13.51036829	RTX Thread	ThreadSwitched	[Running] - thread_id=0x20001880
16125	13.51038236	RTX Thread	ThreadBlocked	[Blocked] - thread_id=0x20001880, timeout=51

The logging functionality enables post-execution analysis and test automation.

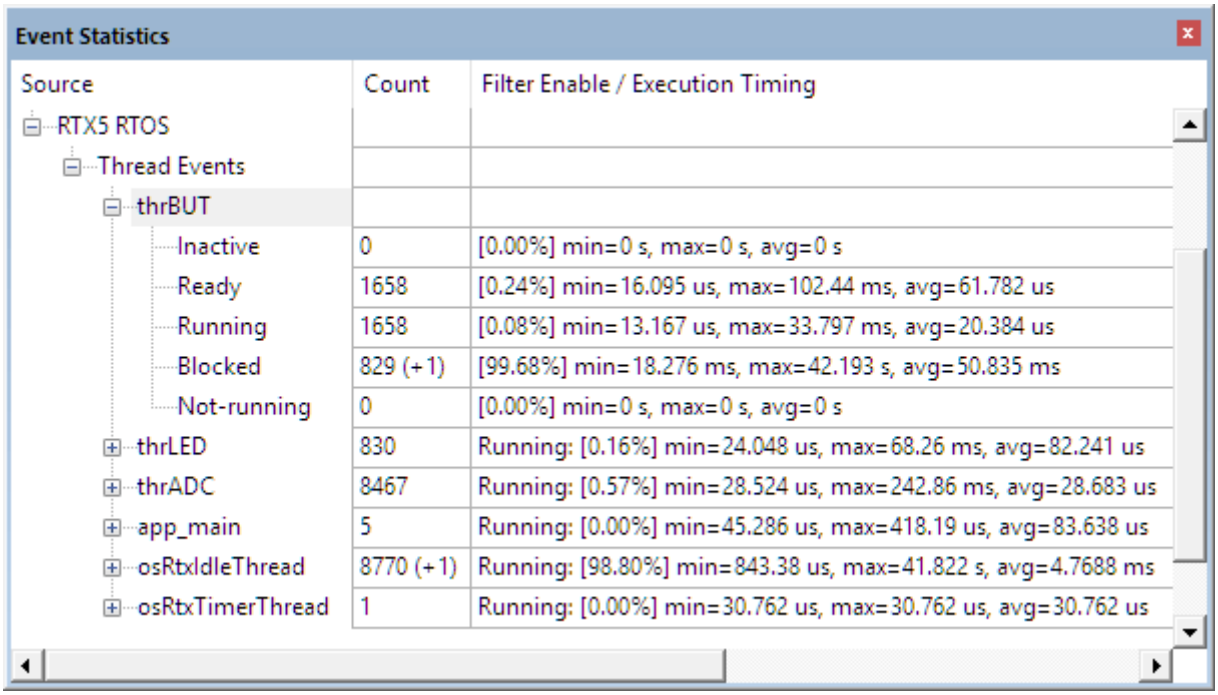
RTX5 documentation explains how to add Event Recorder visibility in RTX RTOS.

RTOS Thread Statistics

The Event Statistics for RTX5 threads becomes automatically available in µVision when Event Recorder is enabled.

At run-time, it displays how many times each thread has been put in a particular state as well as minimum, maximum and average thread execution times in each state.

Figure 4-3: RTX RTOS Event Statistics Window



RTX5 Event Statistics can be saved in a log file for analysis and post-processing as part of a test automation framework.

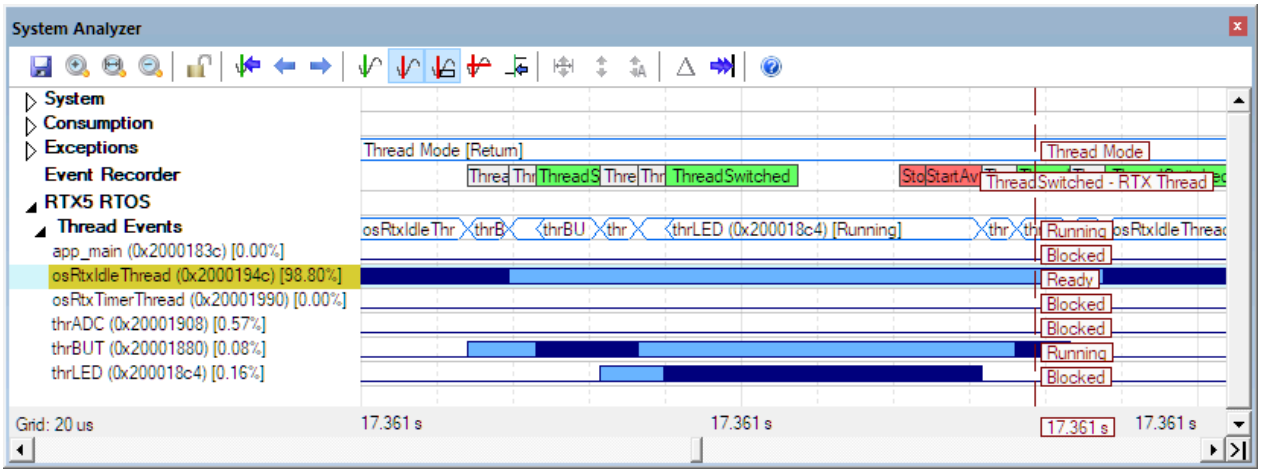
Thread Events in System Analyzer

µVision's System Analyzer window gives developers a graphical view on the program operation over time at it displays the status of each thread. Use the cursor and markers to execute in-depth timing analysis directly in the window.

When trace is used, all interrupts are displayed in the System Analyzer. This provides a time-synchronized view on the RTOS-related exceptions such as SysTick, SVCall and others, but also any interrupts used by the application.

Finally, voltage and current consumption values are also available in the view when using a ULINKplus debug adapter.

Figure 4-4: RTX RTOS Threads in System Analyzer



3rd-Party Tools Support

Keil RTX5 is supported by Percepio's Tracealyzer that visualizes the runtime behavior of embedded software with over 25 graphical views and complements the debugger's low-level perspective with the big picture.

Appendix A Further Reading

Here is a list of additional resources regarding Arm FuSa RTS and functional safety in general.

- Blog: Software building blocks for faster functional safety certification
- Blog: Process isolation with Arm FuSa runtime system
- White Paper: Components and Tools for Functional Safety Applications
- Web Page: Safety
- Web Page: Arm FuSa RTS
- Web page: Arm Compiler for Embedded FuSa
- Web Page: Keil MDK
- Web Page: Keil RTX5 RTOS
- Application Note: KAN307 - Test automation with MDK and ULINKplus
- Application Note: KAN326 - Using X-CUBE-STL with Arm FuSa RTS
- Application Note: KAN336 - TrafficLight: Arm FuSa RTS process isolation example