HighIntegritySystems

WITTENSTEIN

Safety Critical RTOS
Adapting Across Applications

Issue 1.1 - February 26, 2019

This document was prepared by WITTENSTEIN aerospace & simulation ltd for the Embedded World Conference 2019, date as document, all rights reserved.

WITTENSTEIN high integrity systems

Americas:	+1 408 625 4712
ROTW:	+44 1275 395 600
Email:	sales@highintegritysystems.com
Web:	www.highintegritysystems.com

Contents

List of Figures

List of Notation

BSP	Board Support Package
COTS	Commercial off-the-shelf
DAP	Design Assurance Pack
DHF	Design History File
MCU	Microcontroller Unit
MPU	Memory Protection Unit
MMU	Memory Management Unit
RTOS	Real Time Operating System
SIL	Safety Integrity Level
SOUP	Software of Unknown Provenance

CHAPTER 1 Introduction & RTOS Basics

1.1 Introduction

With every new update, microprocessors increase in power and features. An unfortunate side effect is the increase in complexity, requiring the developer to read and understand larger amounts of information. In safety critical applications this complexity poses a significant risk. How does the designer know whether they have covered all eventualities?

One approach is to split the design between the embedded platform and the application. The embedded platform typically includes the microprocessor, a Real Time Operating System commonly referred to as an RTOS, the drivers, middleware, and low level verification routines. The embedded platform encapsulates and abstracts the deeply embedded engineering aspects away from the application. The application then only needs to focus on the functional and safety requirements of the overall system.

It is becoming increasingly popular to tackle complexity by constructing the embedded platform from Functional Safety Components - a common solution being to make use of a Functional Safety microprocessor. Often this type of microprocessor is supported by Functional Safety software libraries implementing built-in test functionality. A Functional Safety RTOS can be used to schedule the individual tests required by the microprocessor, and allow the application access to the microprocessor's resources. The RTOS normally provides the isolation between safety and non-safety functions, allowing a mix of different safety integrity level software within a single build.

Once the developer is satisfied with the safety of the design, they must demonstrate the completeness of the design to a Certification Body, and show that the embedded platform meets the rigorous demands of a wide range of international safety design standards.

The Functional Safety RTOS is a key component of most high integrity software architectures, with many corporations choosing to select a single safe RTOS across their entire organization. This white paper examines how a Functional Safety RTOS can be adapted to meet the requirements of applications in different vertical market sectors, with different technology requirements. It also investigates the challenges embedded developers face when building safety critical embedded platforms from Functional Safety Components. Issues discussed within this white paper include:

- Adapting the RTOS for use with different technology;
- Adapting the RTOS to satisfy different safety design standards;
- Building a software architecture from Functional Safety Components;
- Functional Safety RTOS architecture considerations.

This white paper focuses on the development and use of a Functional Safety RTOS, however the topics discussed can equally be applied to any embedded software component that requires functional safety certification.

1.2 The RTOS

An RTOS is a software component that rapidly switches between Tasks, giving the impression that multiple programs are being executed at the same time on a single processing core. The difference between an Operating System (OS) such as Windows or Linux and an RTOS, is the response time to external events. OSs typically provide a non-deterministic, soft real time response, where there are no guarantees as to when each Task will complete, but they will try to stay responsive to the user. An RTOS differs in that it typically provides a harder real time response, providing quicker reaction to external events.

When switching between Tasks, the RTOS has to choose the most appropriate Task to load next. There are several scheduling algorithms available, including round robin, cooperative and hybrid scheduling. However, to provide a responsive system most RTOSs use a pre-emptive scheduling algorithm.

In a pre-emptive system each Task is given an individual priority value. The faster the required response, the higher the priority level assigned. When working in pre-emptive mode, the Task chosen to execute is the highest priority Task that is able to execute. This results in a highly responsive system. The RTOS will support a well-defined API, which allows the creation of Tasks, and provides access to features that enable communication and synchronization between Tasks, and Tasks and ISRs, such as Queues, Messages, Semaphores, and Mutexes etc.

1.3 Adapting the RTOS for different processor/compilers

RTOS ports are created for specific processor/compiler configurations. Typically, the RTOS design, Fig. 1, will consist of two distinct layers, a Core Layer that contains the majority of the RTOS functionality and implements the RTOS API, and a Portable Layer. The Portable Layer contains the code that integrates with the microprocessor's infrastructure, including the saving and restoring of the processor's internal registers, the reprogramming of the processor's Memory Protection Unit (MPU)/Memory Management Unit (MMU) registers, and the operation of the RTOS Tick. There is also self-test functionality and performance monitoring.

The Portable Layer abstracts and encapsulates the specific microprocessor and compiler details and implementation from the RTOS Core Layer. Generally, the Portable Layer will be kept small and concise. To facilitate porting the RTOS to new processor and complier configurations, ideally the only changes required should be contained within the Portable Layer, allowing the Core Layer to remain common across all RTOS ports.

This approach of partitioning the Core and Portable Layers allows the same RTOS functionality and API to be ported across many different processor and compiler combinations. Therefore, the designer is free to choose the optimum processor and compiler for their new project, and still have the ability to use their preferred RTOS.

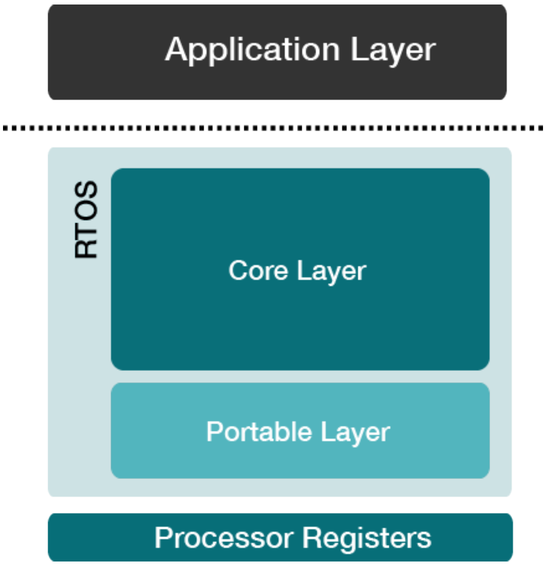

Figure 1. *Typical internal structure of an RTOS*

CHAPTER 2 Functional Safety

2.1 Safety Design Standards

The Safety design standards, illustrated in Figure 2, exist to ensure a consistent, high level of confidence in systems that implement safety critical functionality across different vertical markets. These design standards typically cover all aspects of system, hardware, software, design and verification, and also include integration and usage issues. Different market sectors have created their own sector specific standards. For many market sectors the underlying principles of software design and verification are similar; however the domain specific standards include processes and procedures for managing the unique system level risks present within each sector.

Industry	Standard
Industrial	IEC 61508 [1]
Automotive	ISO 26262 [2]
Medical	IEC 62304 [3] /FDA 510(k)
Rail	EN 50128 [4]
Aerospace	DO178C [5]

Figure 2. *Industry Design Standards*

Certain standards allow for the certification of individual software only components, for example an RTOS. Here the individual component will be certified as a Safety Element out of Context (SEooC), as the final application in which the component will be used is unknown. When designing such software, assumptions will be made about its safety goals and the Safety Integrity Level (SIL) required. These safety goals must be described within the Safety Manual along with the installation and integration instructions. Developers using the software will need to confirm that the safety goals defined during the software's development meet the requirements of their projects.

Other standards take into account the specific product risks, and hence certification can only occur at a product level. Individual software components designed for use in systems where certification is only possible at a system level are normally referred to as "certifiable to". Both IEC 61508 (industrial) and ISO 26262 (automotive) support the certification of software only components; whereas IEC 62304 (medical) and DO178 (aerospace) only support the certification of the final product.

To highlight some of the differences between the standards, IEC 61508 has 4 SIL levels, where SIL 4 is the highest. But a software only component can only be certified to SIL 3, as SIL 4 systems need to take into account the risk presented by the hardware platform. In ISO 26262 software only components can be certified to ASIL 4, which is the highest level. However ISO 26262 requires that all software implementing high ASIL safety functions also have their own software verification monitor to confirm the correctness of operation. Independent Certification Bodies are available to certify or validate individual components and products against a range of international safety standards.

2.2 Adapting the RTOS for Functional Safety

Risk management has a fundamental effect on the design, implementation and deployment of a Functional Safety RTOS. All risks relating to the RTOS need to be identified, and either mitigated within the design or passed up to the system integrator for their consideration. There are many formal methods to identify risks within a software design, remembering the requirement that safety software delivers a very high level of robustness and determinism when compared to commercial grade software.

One way to identify risks within the RTOS is to use a structured tool such as a hazard and operability study (HAZOP). The HAZOP can be used to identify all areas of weakness within the functional RTOS model and API. The results of this process generate a set of safety requirements, which have a fundamental effect on the design and implementation of the RTOS. The resulting functional and safety requirements are passed through a safety critical development life cycle, illustrated in Fig. 2, that supports the safety design standard against which the final product will be certified. The resulting documents and reports generated from the safety critical development life cycle, along with the plans and processes used to manage the development life cycle, form the evidence required to achieve certification. These documents and reports are usually packaged as a Design Assurance Pack (DAP) or Certification Pack (CP). Risks and hazards that cannot be mitigated within the implementation of the RTOS are passed onto the system integrator via the Safety Manual. This passes the responsibility for the residual risks from the RTOS supplier to the system integrators.

The inclusion of the safety requirements creates a significant difference between a commercial grade RTOS and an RTOS designed for Functional Safety use. As the functional requirements are the same, usually the operation and API are similar, but underneath the API the design and code will have been completely redesigned for safety.

2.3 Adapting the RTOS to different Safety Design Standards

When creating a Functional Safety RTOS port for a new processor/compiler combination, designers could implement the work to meet the requirements of one specific safety design standard, for example IEC 61508 SIL 2. On one level this would be very efficient, as the work undertaken would only be that required to meet the requirements of the specific standard in question. However this would severely affect code reuse, as the same RTOS port would need to be re-worked for use with different standards, and different SIL. It also means the development team would have to be trained in many different development life cycles, and the engineers would need to understand which processes and procedures to use for each new work package.

Alternatively, the Functional Safety RTOS could be designed to meet the highest requirements needed to comply with all relevant standards. Each time a new safety design standard is identified, the safety critical development life cycle used to create the RTOS would be updated to comply with the highest SIL requirements for that standard. This maximizes the amount of work and cost when creating a new Functional Safety RTOS port, but means a significant increase in the quality of the RTOS due to the most rigorous elements from each relevant safety design standard being adopted. Code reuse is facilitated, as the same RTOS port can now be used to support all relevant standards, at all SILs. The engineers now only have to learn one safety critical development life cycle, therefore the working practices to create the RTOS can be well and truly institutionalized.

The resulting documents and reports generated from the safety critical development life cycle, along with the plans and processes used to manage the development life cycle, form the Design Assurance Pack (DAP). It is the DAP that is tailored to meet the requirements and expectations of the specific safety design standard. This allows the relevant information to be presented in a form understood by an industry specific Certification Body. For example, the DAP used in industrial applications will be re-formatted as a Design History File (DHF) for medical device manufacturers, however the underlying development life cycle used to develop the RTOS will be the same. Each DAP/DHF will contain a compliance matrix that provides cross references between the relevant standard and the DAP contents.

2.4 Supporting Different Compilers

Safety design standards typically require any tools that have a direct effect on the code to be of the same Safety Integrity Level as the software under design. This includes the compiler, which poses a potential issue for the RTOS developer. Typically the compiler is selected by the customer, but to achieve certification of their RTOS product, the RTOS developer must demonstrate to the Certification Body that the compiler used complies with the relevant safety design standard. It would be expected that an RTOS supplier could support the use of many compilers.

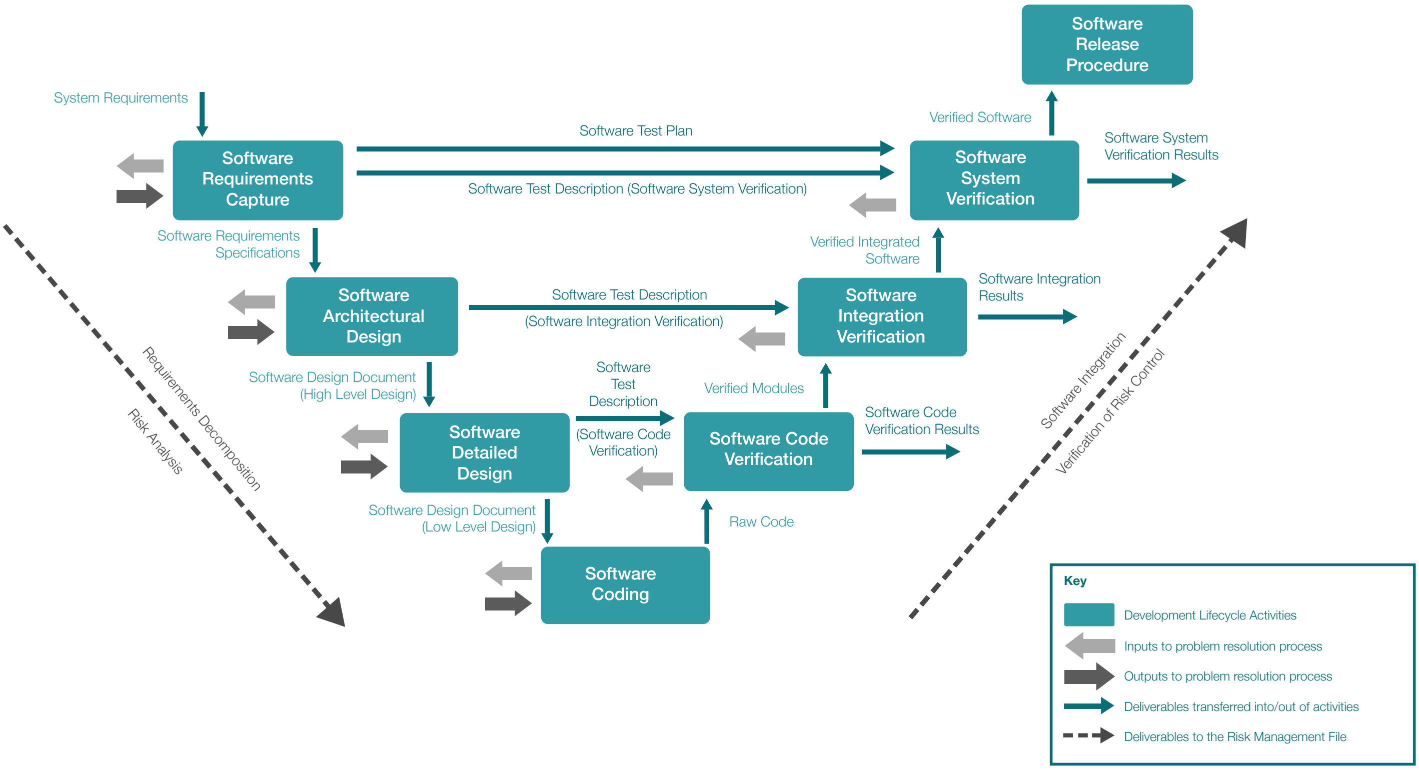
System Requirements
Software Requirements Capture
Software Requirements Specifications
Software Test Plan
Software Test Description (Software System Verification)
Software Release Procedure
Verified Software
Software System Verification
Software System Verification Results
Software Architectural Design
Software Test Description
(Software Integration Verification)
Verified Integrated Software
Software Integration Verification
Software Integration Results
Software Design Document (High Level Design)
Software Detailed Design
Software Test Description
(Software Code Verification)
Verified Modules
Software Code Verification
Software Code Verification Results
Software Design Document (Low Level Design)
Raw Code
Software Coding
Requirements Decomposition
Risk Analysis
Software Integration
Verification of Risk Control
Key
Development Lifecycle Activities
Inputs to problem resolution process
Outputs to problem resolution process
Deliverables transferred into/out of activities
Deliverables to the Risk Management File

Figure 3. *An example of an embedded software Safety Critical Development Lifecycle*

Functional Safety compilers are available, and provide the necessary level of performance and documentation for use within a safety critical development project. However not all developers will select a Functional Safety compiler, instead they will adopt a standard compiler and develop a 'proven in use' argument. This 'proven in use' argument does have its challenges, it may be possible for the product developer, but it's unlikely that an RTOS developer would be able to independently collect enough data to make a 'proven in use' claim.

An alternative approach is to eliminate the compiler from the safety case. The Functional Safety RTOS developer will have a comprehensive and detailed specification for the RTOS, from which the source code is developed. This source code forms the input to the compiler. The RTOS developer utilizes an extensive testing strategy, which tests the output of the compiler. If the output of the compiler is fully verified from tests derived from the requirements, the translation made by the compiler can be removed from consideration.

To confirm that the extensive testing undertaken is extensive enough, a Modified Decision/Condition Coverage (MC/DC) measurement needs to be taken. MC/DC is a measurement of structure coverage. It requires that test cases exercise each condition of each decision within the program, and that when doing so each condition within a decision can independently affect the outcome of the decision. Achieving 100% MC/DC demonstrates that each component of each decision has been tested in isolation, that the object code generated for both the pass and fail cases of each component has been executed, and that resulting behavior has been documented and justified.

By implementing a comprehensive structure coverage measurement analysis to the complete requirements coverage testing such as those discussed above, it can be demonstrated that:

- The output of the compiler is operating as intended by the input to the compiler;
- All requirements have been tested successfully;
- The object code contains no used, or undocumented code segments.

This approach verifies the compiler's interpretation of its input, rather than trusting the compiler to behave as expected. A change in development tools would only require a change impact analysis followed by the re-execution of the RTOS tests in order to have confidence in the compiler output [6].

2.5 Creating a new Functional Safety RTOS port

Porting an RTOS to a new processor / compiler combination involves creating a new Portable Layer for the RTOS, and then integrating the core code layer with the new port layer. When creating a new Functional Safety RTOS port the process is similar, but includes extra safety activities. The RTOS porting task is made easier if the RTOS has been designed to the highest quality level for all relevant safety design standards, as this maximizes design reuse.

The initial activity undertaken is an impact analysis. This involves implementing a HAZOP and errata study to ensure the RTOS is suitable for use on the new processor/compiler combination, and that no known errors have an effect on the use of the RTOS.

The existing RTOS Core Layer design defines the requirements for the new Portable Layer, and it is from these requirements that the new Portable Layer is designed and test cases constructed. Requirement tracing is implemented to trace from the existing Core Layer design to the new Portable Layer design and testing documentation.

Once the code is ready, verification and validation work can commence. Static, inspection, module and integration tests are undertaken on the new Portable Layer. For completeness, module and integration regression tests for the pre-existing Core Layer code are also repeated on the new processor/compiler platform. Then the high-level system verification, validation, MC/DC and soak tests are implemented, to complete the testing.

The resulting DAP is constructed from the existing Core Layer design and testing documents and the new Portable Layer documentation. The DAP will also include all verification and validation test results.

CHAPTER 3 Functional Safety Components

3.1 Scope of a Functional Safety Component

By implementing the safety activities described, and using the resulting documentation to create a DAP, the RTOS becomes a Functional Safety Component. Generally, Functional Safety Components provide specific and well documented functionality, and are designed to meet the rigorous demands of specific safety design standards.

Functional Safety Components are supplied with a DAP (or equivalent) supporting use within a safety critical product, and allowing the component to be certified once integrated into the application. Functional Safety Components are delivered for a specific use case. Using the component within its defined environment, in accordance with the Safety Manual, removes the need for retesting. Functional Safety Components should help to shorten development times and reduce risk within safety critical development programs. It significantly helps if the Functional Safety Component has been pre-certified by a recognized Certification Body, as this reduces the risk of any certification issues relating to the component being identified late in the development life cycle.

A hidden benefit of Functional Safety Components is scale of use. Functional Safety Components from a well-established company will be in use by many different developers across many different applications. This multiplicity of use provides a higher level of confidence in the product. In the unfortunate event a problem is discovered with a component in one product, this can be fed back to all the developers using the component.

3.2 Creating a safety critical embedded platform from Functional Safety Components

A Functional Safety embedded platform is created from the microprocessor, self-test libraries, RTOS, and drivers. In most instances the microprocessor and the self-test libraries are supplied from one company, as only the microprocessor manufacturer has the in-depth knowledge of the microcontroller's design and manufacturing methods needed to create a set of tests that provide enough coverage. The RTOS and the drivers are also normally supplied by a single company, allowing the same safety critical design life cycle to be used.

Creating the Functional Safety embedded platform requires the integration of the individual Functional Safety Components according to the detailed instructions within their Safety Manuals. This process should be straightforward. Taking the RTOS as an example, the accompanying Safety Manual will concisely explain how to install the RTOS into a development environment, ensuring each element is present, in the correct location and is free from corruption. The Safety Manual will also give concise instructions on how to integrate the RTOS with the rest of the application, and clearly detail any residual risks that need to be managed by the overall design.

Following the concise Safety Manual instructions will also generate the evidence required by the Certification Body. The Certification Body will want to see evidence that the installation and integration has taken place correctly, that a competent person has undertaken the installation and integration, and that someone competent and independent from the process has reviewed the evidence and has confirmed the work has taken place as expected. The Certification Body will typically want to see that the Functional Safety Component has been licensed for use, and that it can be supported in the long term. [7]

CHAPTER 4 Functional Safety RTOS Architecture Considerations

Functional Safety Components are robust, deterministic and provide a smooth path to achieving certification once integrated within a product. They should provide the building blocks required to build safety critical products. The following sections detail the building blocks that are sometimes provided by Functional Safety RTOSs.

4.1 Creating a Mixed SIL Software Architecture

Functional Safety RTOSs contain building blocks that enable the separation and isolation of individual Tasks. This feature can assist developers to create a single build of code that contains software of differing SILs. To do this the RTOS will use the processor's underlying MPU or MMU feature. MPUs are becoming a standard feature of microprocessors. They provide a means to establish access permissions for regions of memory. Code execution can be allowed or disallowed for each region. A region can be set for read-only access, read/write access, or no access for both privileged and user modes.

A Functional Safety RTOS should allow the definition and manipulation of MPU regions on a per Task basis, where each Task is assigned specific memory regions. Memory regions can be used for either safety or non-safety code. A useful protection feature is to protect each Task stack with its own MPU memory region. Each time a new Task is given the context, the MPU will be re-programmed with the memory regions for that new Task. Any access violation of a region will cause a Memory Management Fault, and the processor fault handler will be activated. The fault handler is invoked prior to the actual memory access. With care, this allows developers to create a build of software containing software designed for different SILs. An example is shown in Fig.4 [8].

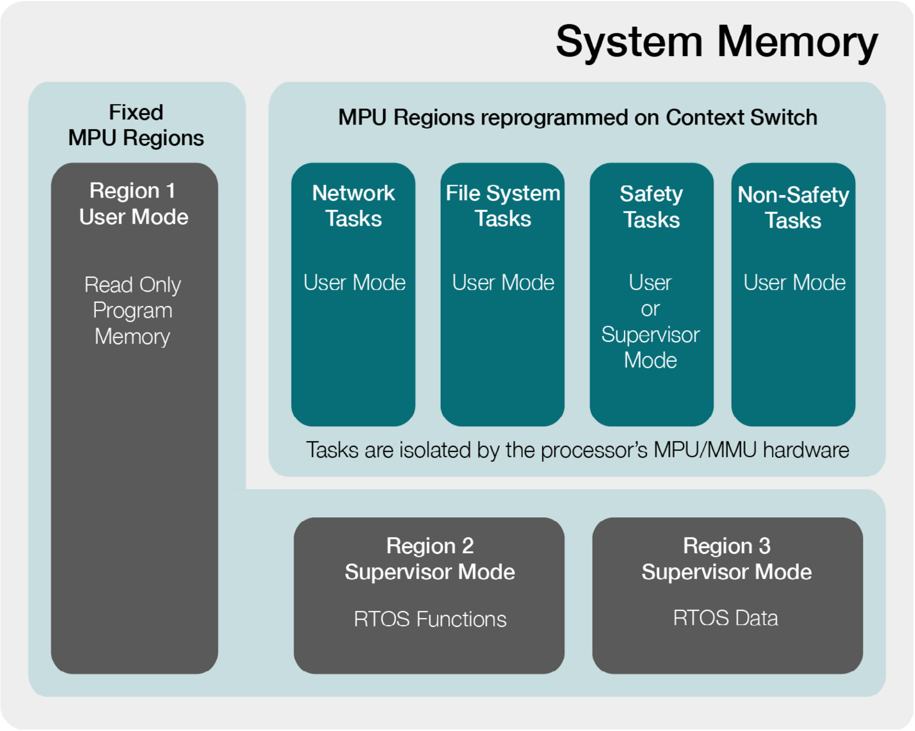

Figure 4. *Building Mixed SIL Software Applications*

4.2 Controlling Access to the Processor's Resources

Controlling access to the processor's resources can be achieved in a variety of ways. There are two risks that need to be managed; ensuring that only one Task has access to the resource at any one time; and avoiding priority inversion, whereby a low priority Task has control of a resource that blocks a higher priority Task from gaining access to the resource. One approach is to use Mutexes, which are binary semaphores that include a priority inheritance mechanism. When used for mutual exclusion the Mutex acts like a token that is used to guard a resource. When a Task wishes to access the resource it must first obtain ('take') the token. When it has finished with the resource it must 'give' the token back – allowing other tasks access.

Mutexes employ priority inheritance. This means that if a high priority Task blocks while attempting to obtain a Mutex (token) that is currently held by a lower priority Task, then the priority of the Task holding the token is temporarily raised to that of the blocking Task. This mechanism is designed to ensure the higher priority Task is kept in the blocked state for the shortest time possible, and in so doing minimize the 'Priority Inversion' that has already occurred. Priority inheritance does not cure priority inversion, it just minimizes its effects. It's best to ensure real time applications avoid priority inversion issues by design.

An alternative option is to use a Gatekeeper Task Fig. 5, to control access to the resource. A basic Gatekeeper Task consists of a Task that controls the 'resource', a queue for receiving data/command, and a callback function. Application Tasks write data/commands to the queue instead of directly accessing the resource. The Gatekeeper Task processes the data commands and the resource is updated accordingly. When the resource changes state an Interrupt Service Routine (ISR) is triggered which is placed in the Queue, the Gatekeeper Task will then execute the registered callback function to pass back the new data to the Application Task.

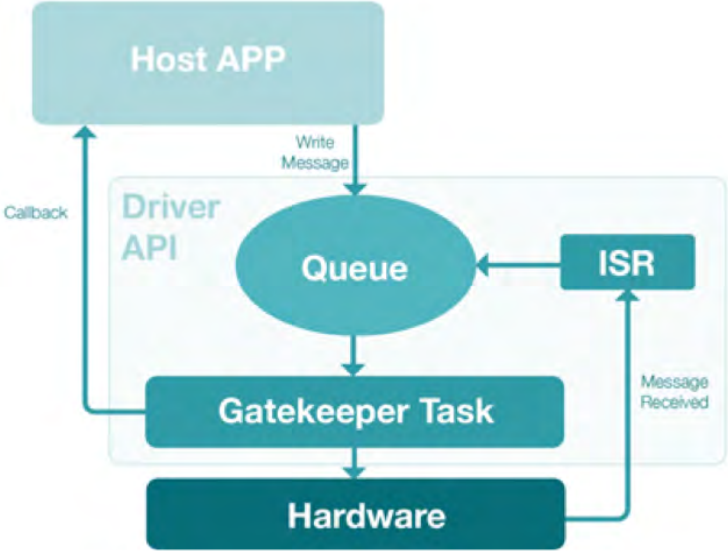

Figure 5. *Architecture of a Gatekeeper Task*

4.3 Multicore Support

It is becoming common for embedded systems to incorporate more than one CPU on a single System on Chip. From a hardware perspective, there are broadly two types of multicore design. If all the CPUs have exactly the same architecture, it is termed homogeneous multicore; if there is variability in the architectures, it is heterogeneous multicore.

From the RTOS perspective, the choice is between AMP and SMP. Where AMP stands for Asymmetric Multi- Processing and SMP means Symmetric Multi-Processing. An SMP system is typically arranged with multiple homogeneous CPUs. There is normally only a single instance of the RTOS, and Tasks are shared across cores with the objective of improving processor throughput. In embedded applications, SMP is used to implement processing heavy routines, for example AI or image processing activities.

An AMP system has multiple CPUs. Each CPU has its own instance of the RTOS, and the RTOS provides a communication facility between the CPUs to share data and provide coordination, using the standard RTOS API calls. AMP solutions are normally deployed where each CPU is undertaking a different role within the system. AMP is suited for use in safety critical or mixed safety critical embedded systems. As each processing core is performing a specific function, there could be a mix of different RTOSs within an AMP solution. For example, a large SoC used for autonomous driving is likely to contain ARM Cortex-A clusters alongside ARM Cortex-R safety cores; the RTOS requirements between these core types could be very different. RTOS multicore solutions are available as Functional Safety Components, and provide a very fast and effective way of creating functional safety capability across a multicore System on Chip.

CHAPTER 5 Conclusion & References

5.1 Conclusion

This paper has examined the complexity, effort and detailed knowledge required to adapt an RTOS to a range of different technology platforms, whilst maintaining compliance to a wide spectrum of vertical specific safety design standards. It has detailed how the experience and 'know how' of safety critical development can be captured, and packaged as a Functional Safety Component, and how Functional Safety Components enable developers to quickly create a Functional Safety embedded platform with minimum effort. The principles discussed go beyond RTOS development, and can be applied to the development of any standalone software component.

This paper has demonstrated that the advantages of using a Functional Safety RTOS are not limited to simpler certification once embedded into an application, but also the Functional Safety features that an RTOS can provide. These features allow software of mixed safety criticality to be developed, the processor's resources accessed safely, and the capability to scale from straightforward single core use up to complex multi core devices. For more information please refer to www.highintegritysystems.com.

5.2 Reference List

[1] IEC 61508:2010 Functional safety of electrical/electronic/programmable electronic safety-related systems.
[2] ISO 26262- 2011 -- Road vehicles – Functional Safety
[3] IEC 62304:2006 Medical device software – Software life cycle processes.
[4] CENELEC – EN 50128 Railway applications – Communication, signaling and processing – Software for railway control and protection systems. June 2011.
[5] DO-178C Software Considerations in Airborne Systems and Equipment Certification, December 13, 2011.
[6] R. Barry "How to verify your compiler for use in IEC 61508 safety critical applications" October 2007.
[7] "Coping with Complexity, Designing for Safety", WITTENSTEIN high integrity systems, June 2018.
[8] "Using an MPU to enforce spatial separation" WITTENSTEIN high integrity systems, May 2017.

Contact Information

User feedback is essential to the continued maintenance and development of SAFERTOS. Please provide all software and documentation comments and suggestions to the most convenient contact point listed below.

Contact WITTENSTEIN high integrity systems

Address: WITTENSTEIN high integrity systems
Brown's Court, Long Ashton Business Park
Yanley Lane, Long Ashton
Bristol, BS41 9LB
England

Phone: +44 (0)1275 395 600
Fax: +44 (0)1275 393 630
Email: support@HighIntegritySystems.com

Website www.HighIntegritySystems.com

All Trademarks acknowledged.

This document was prepared by WITTENSTEIN aerospace & simulation ltd for the Embedded World Conference 2019, date as document, all rights reserved.