HighIntegritySystems

WITTENSTEIN

Coping with Complexity, Designing for Safety

Building an embedded platform based on the STM32 SIL Functional Safety Design Package

Issue 2.2 - October 14, 2020

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

1.1 Introduction

With every new update, microcontrollers 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 if they have covered all eventualities? One approach is to split the design between the embedded platform and the application. The embedded platform includes the microcontroller, Real Time Operating System (RTOS), drivers and all low level verification routines, and encapsulates and abstracts all of the 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 pre-certified safety components - a common solution being to make use of a safety certified microcontroller. Often this type of microcontroller is supported by pre-certified software libraries implementing built-in test functionality. A pre-certified functional safety RTOS can be used to schedule the individual tests required by the microcontroller, and provide access by the application to the microcontroller's resources. The RTOS normally provides the isolation between safety and non-safety functions, and most RTOS vendors can provide safety certified embedded drivers.

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

This paper will examine the advantages of using a safety certified processor to tackle complexity, and how to demonstrate completeness of design to a certification body using an STM32 embedded microcontroller and SAFE**RTOS**®, a safety critical Real Time Operating System.

CHAPTER 2 Safety Standards

Safety critical design standards exist to ensure a consistent, high level of confidence in systems that implement safety critical functionality across different market sectors. These design standards typically cover all aspects of system, hardware and software design and verification, but also include integration and usage issues. Different market sectors have created their own sector specific standards. For many market sectors the underlying principles of 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.

Certain standards allow the certification of individual components, such as a software only component like 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. Other standards take into account the specific product risks, and hence certification can only occur at a system/product level. Individual components designed for use in systems where certification is only possible at a system level are normally referred to as "certifiable to".

Independent Certification Bodies are available to certify individual components and products against a range of international safety standards.

Table 2-1. *Safety Standards*

Industry	Standard	Certification of:
Industrial	IEC 61508	Component and systems level
Medical	FDA 510(k), IEC 62304	System only
Automotive	ISO 26262	Component and systems level
Aerospace	DO178C	System only

CHAPTER 3 Pre-Certified Embedded Components

Pre-certified functional safety components provide specific and well documented functionality, and are designed to meet the rigorous demands of a specific international safety standard. Pre-certified functional safety components are supplied with a Design Assurance Pack (DAP) containing all design, verification, validation and certification evidence required for certification. The most important element of the DAP is the Safety Manual, which defines how to install, integrate and use the component within a safety system. Any residual risks that have not been fully resolved by the component will be clearly identified within the safety manual, for the system integrator to address.

Pre-certified functional safety components will be created for a specific environment. For example, a safety critical RTOS will be delivered for a specific processor/compiler combination, including the version of the compiler and the specific compiler settings. Using the module within its defined environment, in accordance with the safety manual, removes the need for re-testing.

Pre-certified functional safety components allow the purchasing of certification in advance for a core component of the system. This removes the time, effort and resources required for the design, verification and certification of the module. It should also help to shorten development times and reduce risk within the development program. It significantly helps if the component has been pre-certified by a respectable 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 pre-certified safety components is scale of use. Safety modules from a well-established company will be in use by many different developers across many different applications. This multiple of use provides a higher level of confidence in the product. If 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.

CHAPTER 4 Creating a Safety Critical Embedded Platform

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

Creating the safety critical embedded platform requires the integration of the processor, test libraries, RTOS drivers and board support packages according to the detailed instructions within the safety manual. Taking SAFE**RTOS**® as an example, the accompanying safety manual explains exactly how to install and integrate SAFE**RTOS** into a development environment. Following the concise instructions will also generate the evidence required by certification bodies confirming that the process has been followed correctly.

The remainder of this paper details the construction of a safety critical embedded platform constructed from a STM32 microcontroller with the Self-Test Library supplied by STMicroelectronics, and the RTOS SAFE**RTOS** with accompanying drivers, being supplied by WITTENSTEIN high integrity systems.

4.1. STM32 Microcontrollers

The STM32 family of 32-bit Flash microcontrollers is based on the ARM® Cortex®-M processor. The STM32 family has several series from the low-end STM32F0 up to high performance STM32H7, demonstrated by figure 4-1.

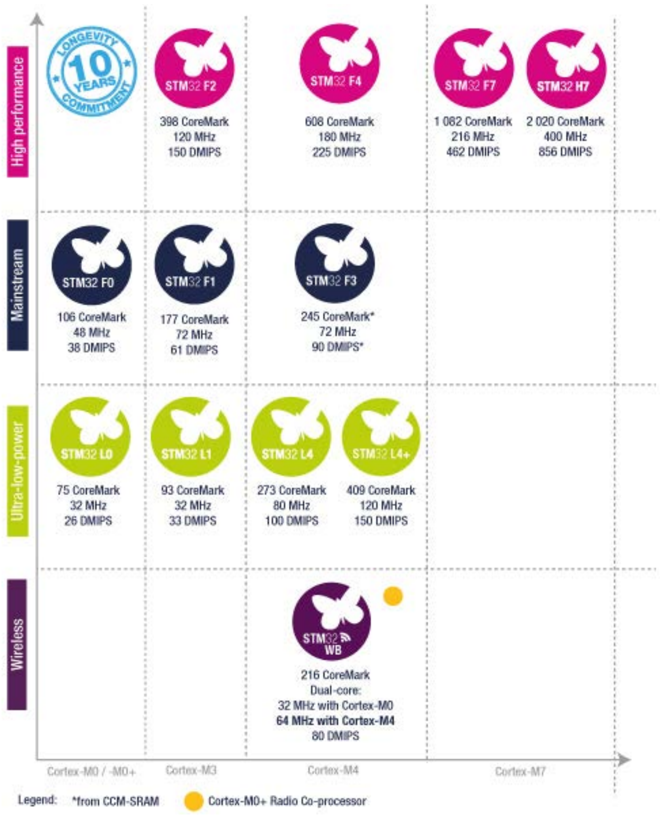
High performance
Mainstream
Ultra-low-power
Wireless
LONGEVITY 10 YEARS COMMITMENT
STM32 F2
398 CoreMark 120 MHz 150 DMIPS
STM32 F4
608 CoreMark 180 MHz 225 DMIPS
STM32 F7
1 082 CoreMark 216 MHz 462 DMIPS
STM32 H7
2 020 CoreMark 400 MHz 856 DMIPS
STM32 F0
106 CoreMark 48 MHz 38 DMIPS
STM32 F1
177 CoreMark 72 MHz 61 DMIPS
STM32 F3
245 CoreMark* 72 MHz 90 DMIPS*
STM32 L0
75 CoreMark 32 MHz 26 DMIPS
STM32 L1
93 CoreMark 32 MHz 33 DMIPS
STM32 L4
273 CoreMark 80 MHz 100 DMIPS
STM32 L4+
409 CoreMark 120 MHz 150 DMIPS
STM32 WB
216 CoreMark Dual-core: 32 MHz with Cortex-M0 64 MHz with Cortex-M4 80 DMIPS
Cortex-M0 / -M0+
Cortex-M3
Cortex-M4
Cortex-M7
Legend: *from CCM-SRAM
Cortex-M0+ Radio Co-processor

Figure 4-1. The *STM32 Family*

4.2. STM32 SIL Functional Safety Design Package

The STM32 SIL Functional Safety Design Package helps safety critical systems customers achieve certification, targeting the industry standard IEC 61508 Safety Integrity Level (SIL2/3). It has STM32 embedded safety features, built on the quality foundations of the STM32 product portfolio. This includes ST's 10-years longevity commitment.

The STM32 SIL functional safety design packages provide the safety manuals and Self-Test Library that developers need to create STM32-based safety critical systems, targeting the industry standard IEC 61508 Safety Integrity Level (SIL2/3). The key features of the STM32 SIL functional safety design package are:

- MCU Safety Manual
- MCU Failure Modes and Effects Analysis (FMEA)
- Failure Modes Effects and Diagnostics Analysis (FMEDA) Snapshot
- X-CUBE-STL Self-Test Library

The MCU Safety Manual details the list of safety requirements (conditions of use) and examples to guide STM32 users to achieve safety integrity certification in compliance with IEC 61508.

The MCU Failure Modes and Effects Analysis (FMEA), details a list of MCU failure modes and related mitigation measures. The Failure Modes Effects and Diagnostics Analysis (FMEDA) provides a static snapshot reporting IEC 61508 failure rates and safety metrics, computed at both MCU and basic function levels.

The STM32 SIL functional safety design packages, while using the IEC61508 safety standard as main reference, can be used in the framework of other different safety standards. The MCU Safety Manual includes an Appendix mapping the IEC61508 results and outcomes to other widely adopted safety standards.

Please check with STMicroelectronics which STM32 devices are fully supported for safety use.

4.2.1. X-CUBE-STL: STM32 Self-Test Library

To support its safety critical developers STMicroelectronics has released a software-based diagnostic suite: X-CUBE-STL.

X-CUBE-STL, or STL, is a STM32Cube Expansion Package. STL is an application-independent software test library released by ST to implement a relevant subset of safety mechanisms required by the safety concepts applicable to microcontrollers of the STM32 Series. The STL is HAL/LL independent, and dedicated to the STM32 Series.

STL is an autonomous software component, which executes on application/RTOS demand selected tests to detect hardware issues, and reports the outcomes to the application. The STL is delivered partly in object code (for the library itself) and part in source code for the user interfaces definition and user parameters settings. The STL tests the Arm® Cortex®-M core, the Flash memory and the RAM - implementing the set of safety mechanisms addressing microcontroller main functions.

STL is developed according IEC 61508-3 V-cycle, with a systematic integrity level of SC3 (SIL3). The compliance of development flow to IEC 61508 is certified by an independent agency - helping the integration of STL with other application softwares in an IEC 61508 compliant environment.

The Diagnostic coverage is verified by a state-of-the-art ST proprietary fault injection methodology according to IEC 61508 requirements. The compliance of diagnostic coverage verification method to IEC61508 is certified by an independent agency.

The STL also gives the developer the possibility to:

- Schedule User tests, i.e. tests defined by users and executed through the scheduler. This feature is available by user configuration.
- Use an artificial failing feature. The developer can emulate the application behaviour in presence of hardware failures by forcing the STL to return a requested test result value. This feature is available by specific user API and is useful during end user integration phase.

The STL includes:

- ARM® Cortex®-M core Tests
- Embedded Flash Memory tests
- RAM tests
- User tests

The test library is supplied as compiler independent object code. Users and safety manuals are supplied supporting use of the libraries up to IEC 61508 SIL 3.

Please check with STMicroelectronics which STM32 devices are fully supported by the Self-Test Library.

4.2.2. STM32Cube

STM32Cube includes STM32CubeMX, a graphical software configuration tool that allows the generation of C initialization code using graphical wizards and ready-to-use project for Keil, IAR and System Workbench toolchains.

It also embeds comprehensive STM32Cube MCU Packages, delivered per STM32 microcontroller Series (such as STM32CubeF4 for STM32F4 Series). These packages include the STM32Cube HAL (an STM32 abstraction layer embedded software ensuring maximized portability across the STM32 portfolio), the STM32Cube LL (low-layer APIs, a fast, light-weight, expert-oriented layer), plus a consistent set of middleware components such as RTOS, USB, TCP/IP and graphics. All the embedded software utilities are delivered with a full set of examples.

The software Self-Test Library is packaged as an STM32Cube Expansion Package and is easily integrated in a STM32Cube project.

4.3. SAFE**RTOS**®

The RTOS performs an important role in the creation of a safe embedded platform. It must control access to the processor's resources, schedule the STL, and support the relevant safety standards.

SAFE**RTOS** is a safety critical, highly deterministic, embedded RTOS with an accompanying Design Assurance Pack that provides an easy route to achieving certification of SAFE**RTOS** once integrated into a Safety Product.

With an imperceptible boot time, SAFE**RTOS** provides a priority-based, pre-emptive, Real Time Operating System. The number of priority levels is user-defined. The highly deterministic scheduling algorithm ensures that the highest priority Task that is able to run, is the Task which is selected to run.

Inter-Task communication and synchronisation is achieved using features including Semaphores, Mutexes, Queues, Event Groups and Task Notification. SAFE**RTOS** also supports a timer feature.

In addition to the Task Separation and Isolation feature, SAFE**RTOS** contains a range of other safety features.

SAFE**RTOS** performs thorough API input validity checking (as far as practically possible) in order to mitigate the risk of misuse by the host application. If the value of an API parameter is found to be invalid, the API function will not perform any action other than returning an error code indicative of the error encountered.

SAFE**RTOS** also performs the following run time integrity checking with the intention of facilitating the detection of data corruption:

- The execution context of a Task that is not in the Running state is stored on the stack allocated to the Task. The context of a Task will only be saved onto the stack of the Task if there is sufficient stack space remaining to hold the entire stack.
- A check is performed to ensure that the Task Control Block associated with the Task selected to enter Running state is valid. This is achieved by checking key data parameters against their inverted mirror copies.
- Prior to implementing the tick count value, a check is performed to test that the current tick count value remains at the last written and therefore expected value.
- Verification that the MPU modes and privileges are restored correctly.

A failure in any of these integrity checks will result in a call to the application Error Hook function.

Some features that are present in other operating systems have been removed from SAFE**RTOS** due to safety concerns. For example, SAFE**RTOS** does not perform any dynamic memory allocation operations, but instead requires the application to allocate a block of memory for SAFE**RTOS** during the initialisation sequence. Reference to this memory block is passed to SAFE**RTOS** via the API during the initialisation phase. Application designers are still able to use dynamic memory allocation within their designs.

4.3.1 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 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 the opportunity to 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 minimise the 'Priority Inversion' that has already occurred.

Priority inheritance does not cure priority inversion, it just minimises its effects. It's best to ensure real time applications avoid priority inversion issues by design.

The preferred option is to use a Gatekeeper Task to control access to the resource. A basic Gatekeeper Task is shown in Figure 4-1, consisting of a Task that controls the 'resource', a queue for receiving data/command, and a callback function.

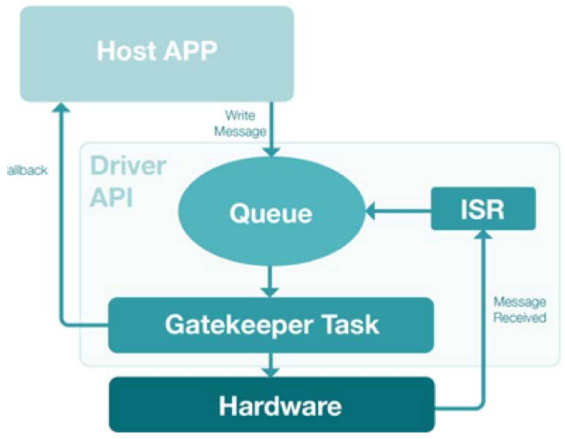

Figure 4-2 *A Basic Gatekeeper Task*

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 (say it's a new network message) which is placed in the Queue, the Gatekeeper Task will then execute the registered call back function to pass back the new data to the Application Task.

A basic but effective alternative to avoid Priority Inversion is to enter a critical section. This is where the RTOS scheduling algorithm is suspended during the time a Priority Inversion could occur. Although an appropriate solution in some systems, it may affect the responsiveness of the system.

4.3.2 Mixed Safety Integrity Levels

SAFERTOS supports the definition and manipulation of MPU regions on a per task basis. This feature provides the tools allowing developers to add a degree of spatial separation between tasks, which used effectively, can help prevent tasks directly making unintentional or accidental access to incorrect memory regions.

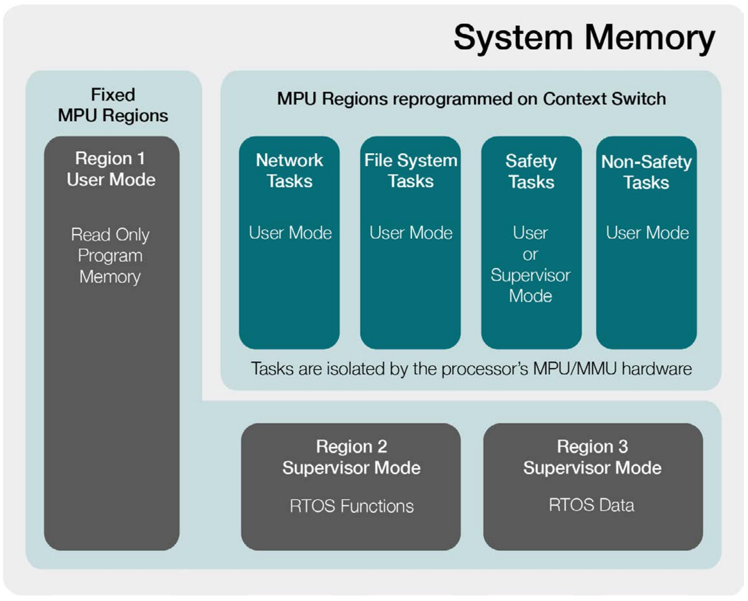

Figure 4-3 *Memory Protection Unit*

The MPU implementation is tightly coupled to the STM32 core and provides 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. This can be used to set up an environment where only kernel or system code can access certain hardware registers or sections of code.

The kernel code and data regions are allocated to memory with privileged rights. The remaining MPU regions are allocated to Tasks, and these can be set at the appropriate access level (user or privileged). MPU memory regions relating to Tasks are reprogrammed at each context switch, enabling each Task to have an individual memory access profile.

Once the memory regions are assigned, the MPU and the processors Memory Manage Fault handler is enabled. An access violation of a memory region will cause a Memory Manage Fault, and the processor fault handler will be activated.

4.3.3. Scheduling of the STL

The STM32 STL's built-in Software-Test library will operate within a series of Tasks. The priority of the Tasks needs to be carefully chosen to ensure that the responsiveness of the system is not affected, but that the required tests are completed within the required time frame.

The set of constraints for STL time execution depends on the characteristics of the end application safety concept, and on the selected safety standard.

4.3.4. Safety Support

SAFE**RTOS** and its Design Assurance Pack (DAP) are available pre-certified by TÜV SÜD to IEC 61508 SIL 3 (Industrial), the highest level possible for a software-only component, and ISO 26262 ASIL D (Automotive). SAFE**RTOS** has been independently certified by TÜV SÜD, with the initial certification achieved in 2007, and has been used across a wide range of safety critical industries.

SAFE**RTOS** is certifiable to IEC 62304 Class C and FDA 510(k) (Medical), and to DO178C (Aerospace), and supports all STM32 devices for a wide range of compilers.

4.4 Board Support Package

The basic STM32 board support package supplied with SAFE**RTOS** is based on the drivers generated by the STM32Cube. WHIS checks that the drivers are thread safe, and updates the code to conform with the WHIS safety critical coding standards. WHIS documents the functional requirements for the basic driver, and creates a set of safety requirements for the driver, using the SAFE**RTOS** HAZOP process. The resulting functional and safety requirements are placed through the SAFE**RTOS** safety critical design life cycle, to create the safety critical driver and its Design Assurance Package.

If a more advanced driver is required, the additional functional requirements will be added to the requirement set.

CHAPTER 5 Conclusion

This paper has examined the parts necessary to form a safety certified embedded platform, made up of a microcontroller, an associated Self-Test Library, RTOS, drivers, and Board Support Packages, using the STM32 microcontroller family and SAFE**RTOS**.

A safety critical microprocessor environment can be created very simply by following the instructions detailed within the relevant STM32 safety manual.

We have seen how the RTOS performs an important role in the creation of a functional safe Embedded Platform. It must control access to the processor's resources, schedule the test library, and support the relevant safety standards.

Following the accompanying safety manuals allows the developer to demonstrate completeness of design to a certification body, and show that the embedded platform meets all the rigorous demands of a wide range of international safety standards.

This method greatly reduces complexity, aiding in the design of a safety critical device.

Contact Information

User feedback is essential to the continued maintenance and development of SAFE**RTOS**. 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.High**Integrity**Systems.com

All Trademarks acknowledged.