WITTENSTEIN

Using the MPU with an RTOS to Enhance System Safety and Security

By

Stephen Ridley

10 December, 2016

WITTENSTEIN

WITTENSTEIN high integrity systems: A World Leading RTOS Ecosystem

FreeRTOS™ is a market leader, with over 100,000 downloads per year

freeRTOS
From Real Time Engineers Ltd

We added **commercial licensing**, **middleware**, **support** & **indemnification**

OPENRTOS®

We rebuilt for the **safety sector**, adding multiple **certification standards**.

SAFERTOS®

We provide a leading RTOS ecosystem that's achieved global recognition

WITTENSTEIN

Objectives

- Safety Systems, Functional Safety, Software Components and Partitioning.
- Multi Core and Multi Processor Architectures as a means of Partitioning.
- Using the MPU to Partition and Protect.

Safety, Security, Software and Systems

- In embedded systems, when we talk about “Safety”, we usually mean ‘Functional Safety’. Every industry sector has its own set of standards setting out the development steps and verification activities required for a product to achieve a given Safety Integrity Level (SIL).
- When we are talking about “Security”, we mean guaranteeing integrity, preventing intrusion and unauthorized use of resources.
- For both of these aims, using an MPU can help by preventing code from operating or accessing data outside of its assigned bounds.
- Does not matter which industry sector or standards we need to be compliant to, using an MPU can help to prove that the system is operating within its assigned parameters.
- Finally, “Safety” and “Security” apply to “Systems” and can encompass multiple devices featuring hardware and software; however we are primarily concerned with software within this talk.

Software Components and System Partitioning

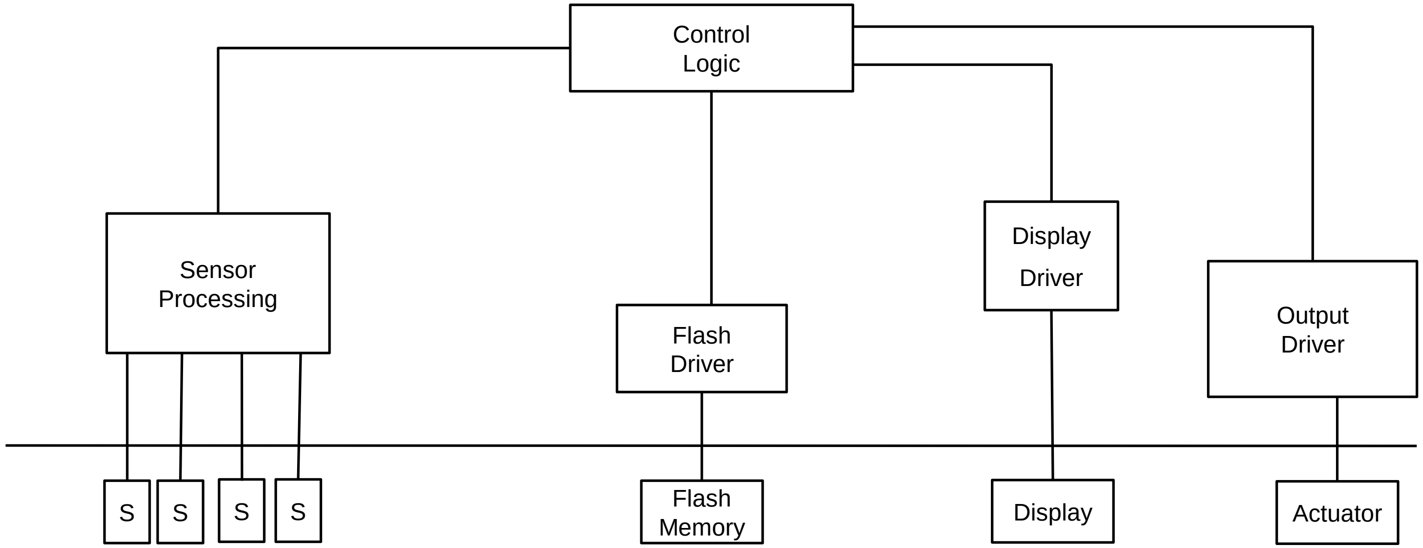
Another use for an MPU is to enforce partitioning. This is just part of the wider subject of mixed SIL programming, where critical parts of the system are developed to the required SIL level. However, we may also use commercial grade software (COTS or SOUP) as well as lower grade software developed in house.

For full mixed SIL programming to work, we need to:

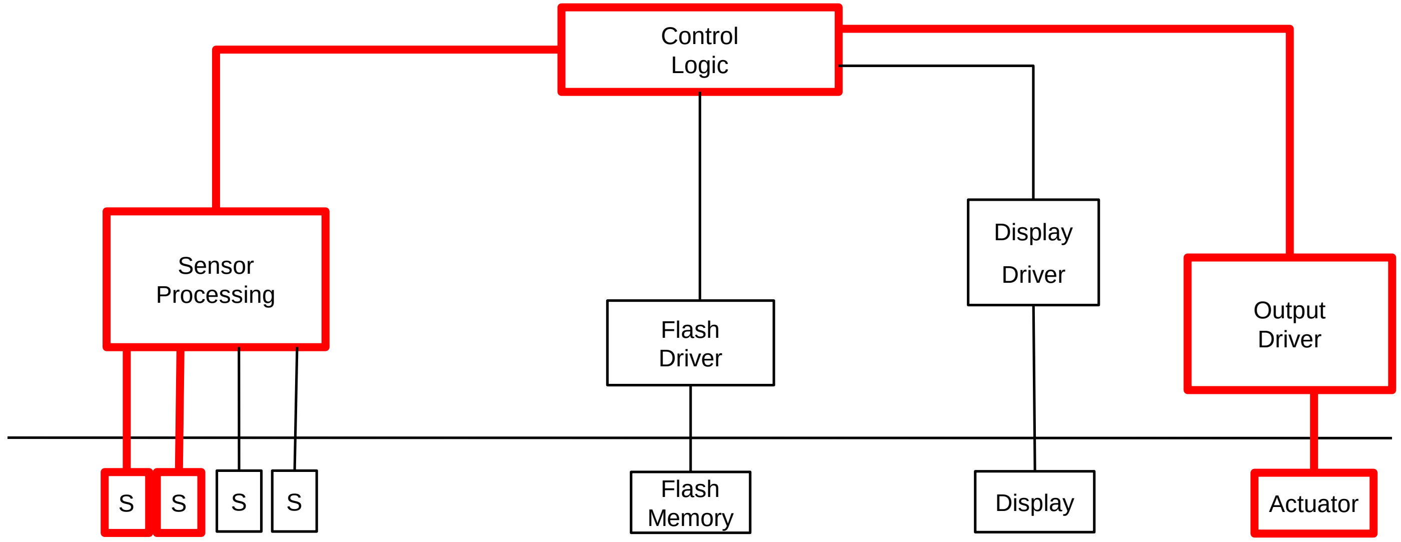
1) Identify safety functions and use case requirements.
2) Ensure that the system architecture can supply the necessary spatial separation (i.e. prohibit memory accesses by non-safety code that could compromise the operation of the safety software).
3) Ensure that the system architecture can supply the necessary temporal separation (i.e. prove that non-safety software cannot prevent the safety software having sufficient run time to achieve its purpose).
4) Be able to prove that the requirements have been correctly implemented

Use Case – An Embedded System

Standard practice says that all software must be developed to the highest SIL level required.

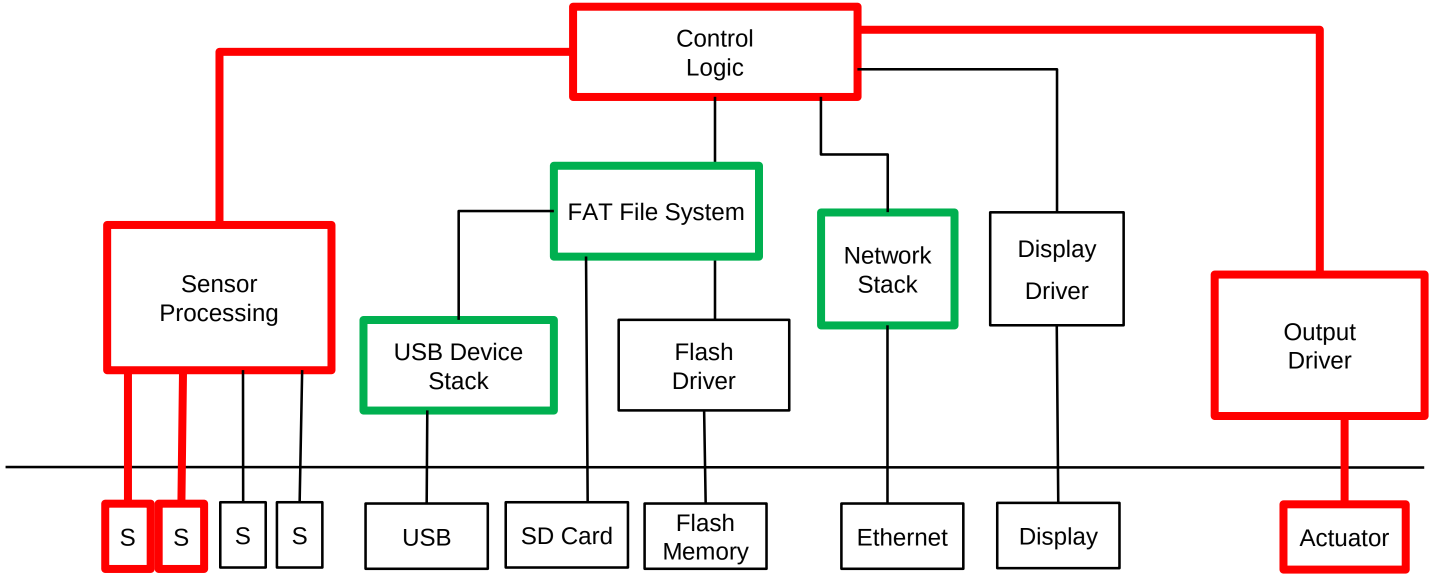


Use Case – An Embedded System



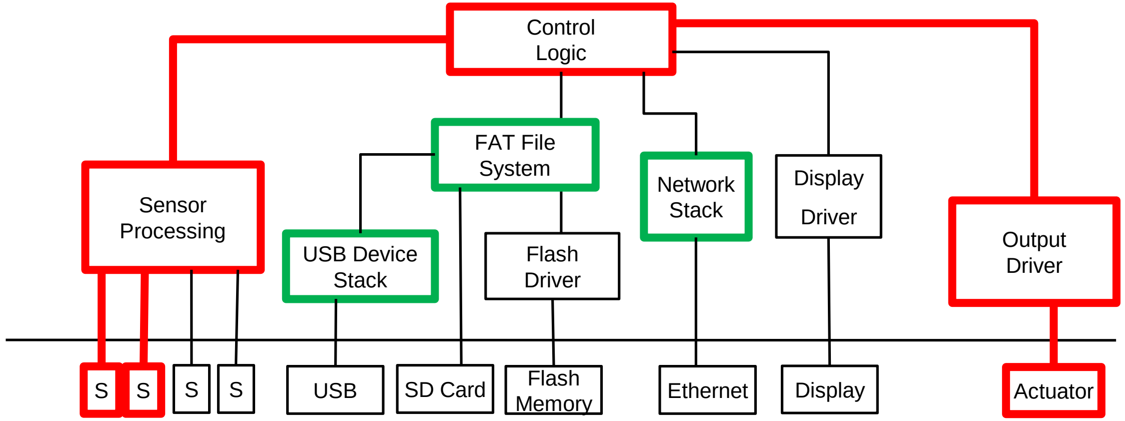
WITTENSTEIN

Use Case – Marketing and Ambitious Engineers Now Involved



WITTENSTEIN

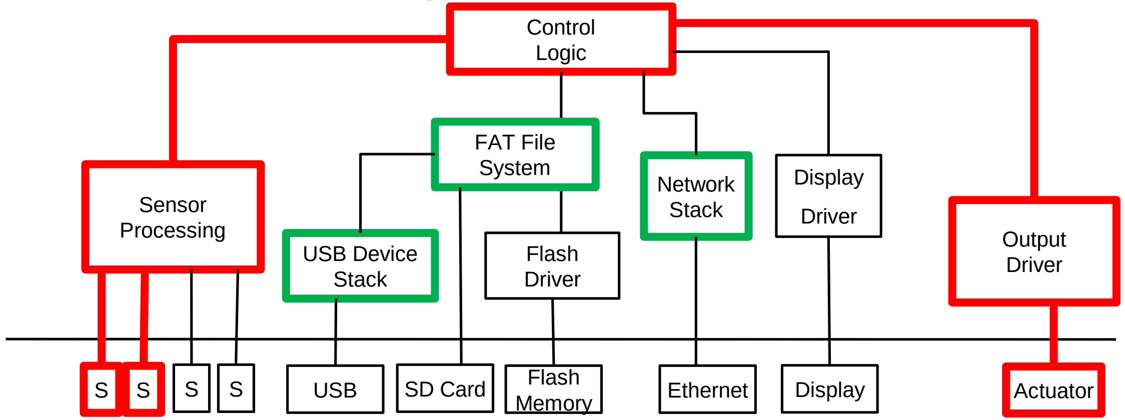
Use Case – An Embedded System



We now have:

- Critical safety software.
- Commercial third party software that we have no control over.
- Other software not developed to the required SIL.

Use Case – An Embedded System

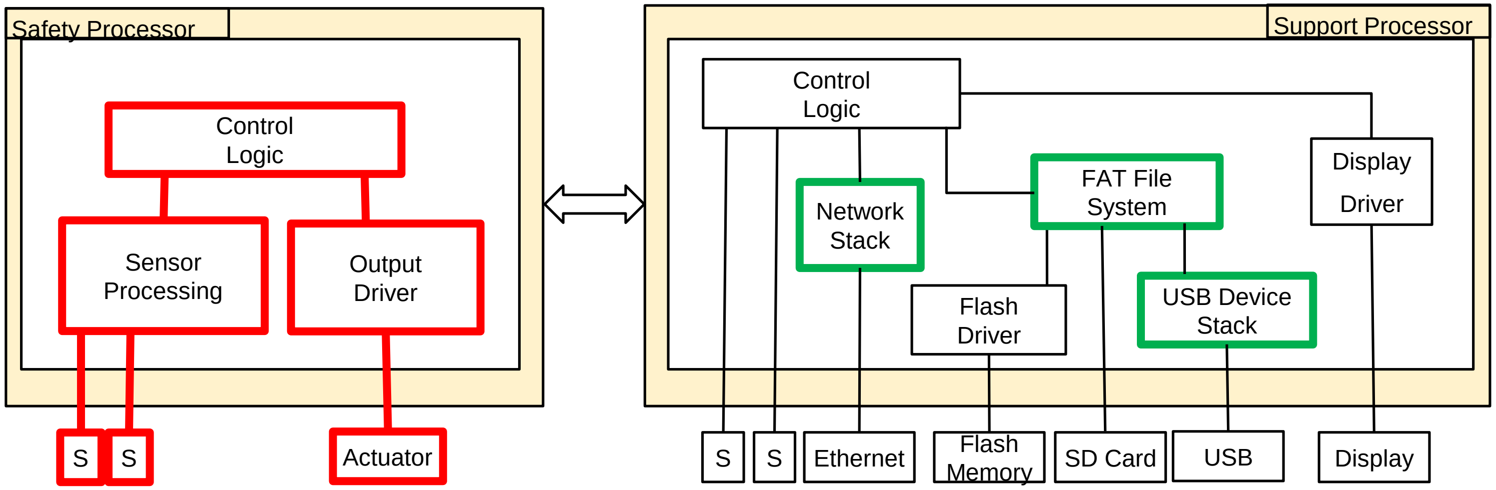


For this system to be implementable in an acceptable manner for a safety system, we need to be able to demonstrate:

- Spatial separation between the safety code/data and the non-safety code/data.
- Temporal separation between the safety code and the non-safety code.
- Data passed through non-safe components is either not safety related or protected by other protocols.

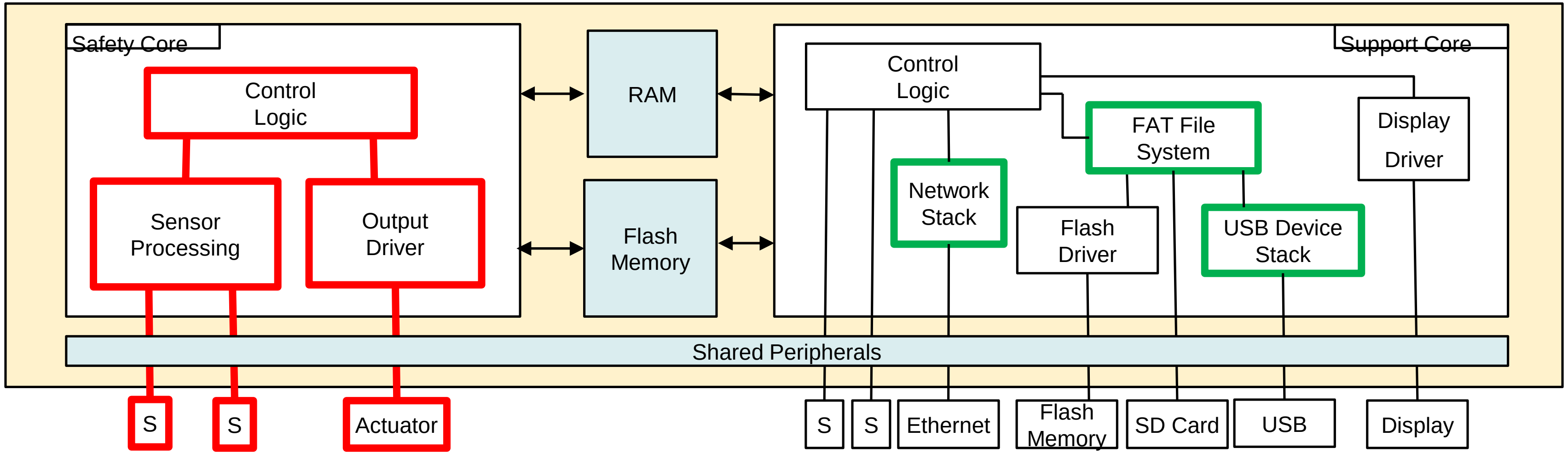
Using Multiple Processors to Achieve Separation

- Relatively easy from a software perspective in that there is clear isolation between safety and non-safety components.
- More expensive and complicated hardware (not optimal in low cost embedded systems).



Using a Multi-Core Processor to Achieve Separation

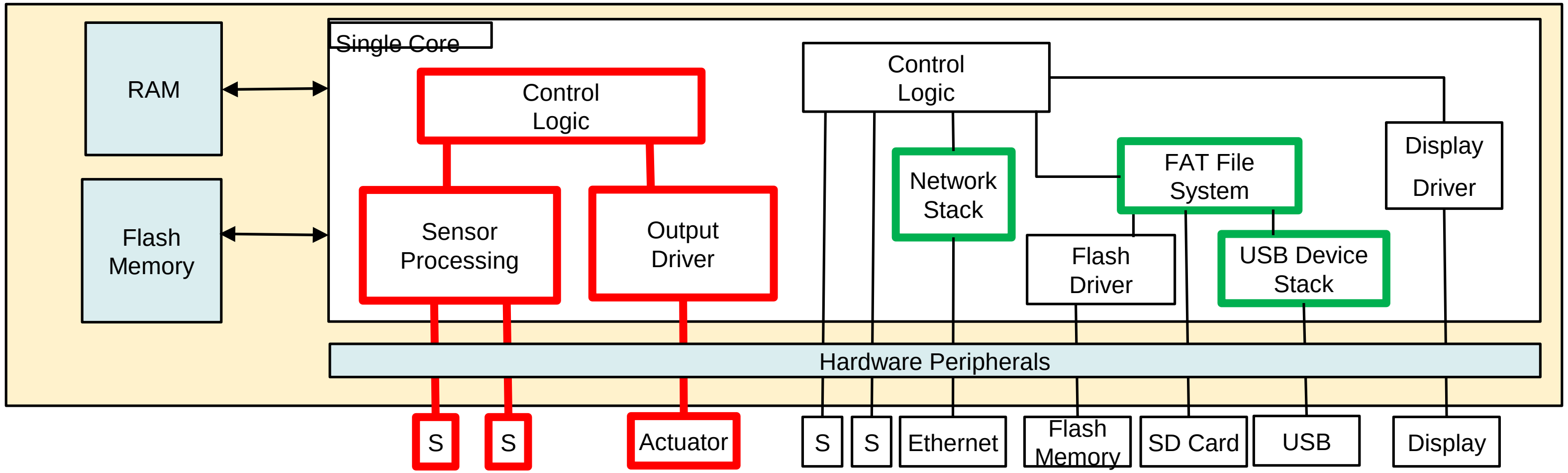
- Easier hardware design, only one set of peripheral components.
- As both cores typically share access to memory and peripherals it is difficult to claim spatial separation without using an MMU or MPU.



WITTENSTEIN

A Single Core System has no Inherent Separation.

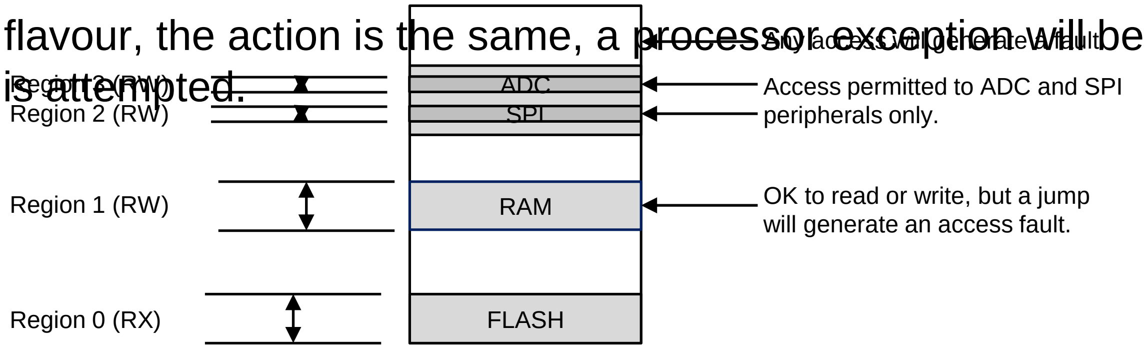
- Different approach needed to achieve spatial separation.
- An MPU or MMU can be used to achieve spatial separation.



WITTENSTEIN

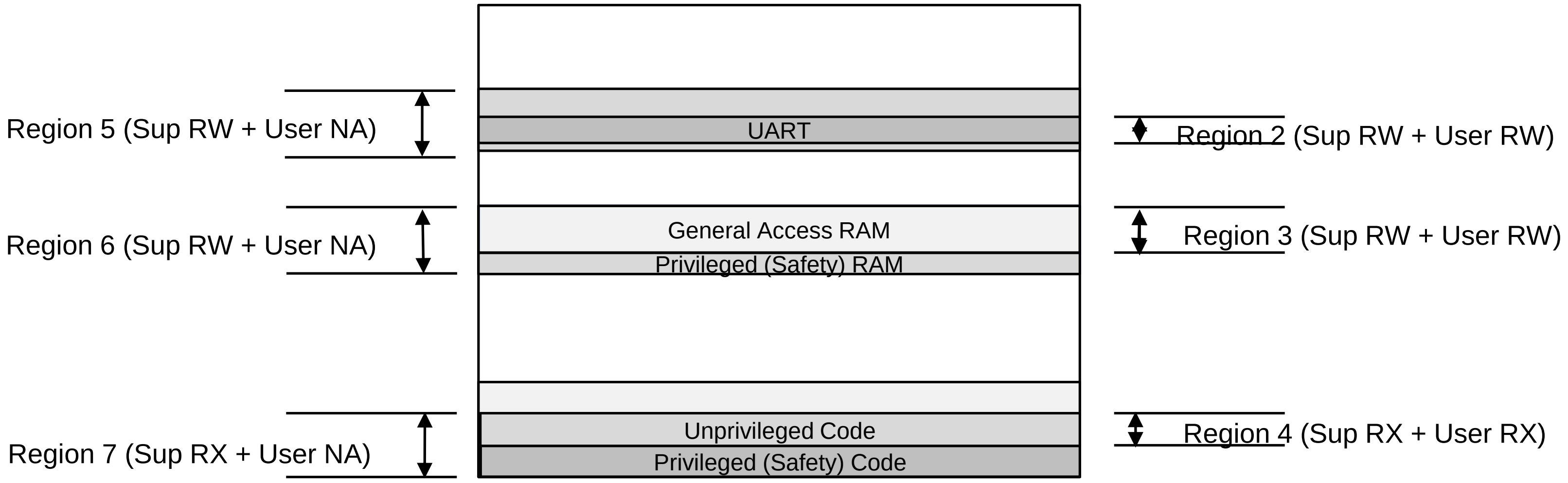
Using a Memory Protection Unit (MPU)

- The MPU is used to prevent access to unauthorised regions within the memory map.
- Microprocessors that have an MPU typically allow a number of “memory regions” to be defined. This is usually a memory range and associated access permissions.
- Some differences depending on the silicon manufacturer (e.g. ARC processors feature prioritised MPU regions whereas others are additive with respect to granting of permissions).
- Whatever the flavour, the action is the same, a processor exception will be generated if an illegal access is attempted.


Any access will generate a fault.
Region 3 (RW)
ADC
Region 2 (RW)
SPI
Access permitted to ADC and SPI peripherals only.
Region 1 (RW)
RAM
OK to read or write, but a jump will generate an access fault.
Region 0 (RX)
FLASH

Using a Memory Protection Unit (MPU)

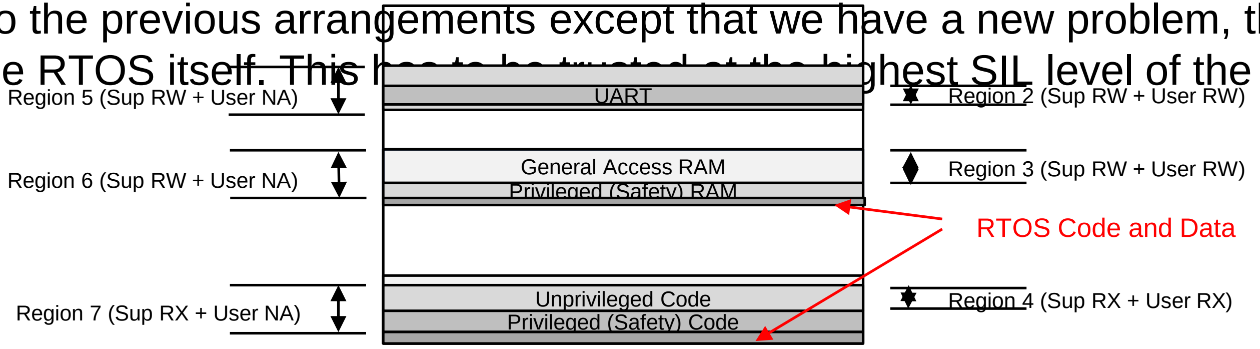
- Use processor privilege modes. Different access permissions can be granted depending on whether the processor is running in privileged mode or not.


Region 5 (Sup RW + User NA)
UART
Region 2 (Sup RW + User RW)
Region 6 (Sup RW + User NA)
General Access RAM
Privileged (Safety) RAM
Region 3 (Sup RW + User RW)
Region 7 (Sup RX + User NA)
Unprivileged Code
Privileged (Safety) Code
Region 4 (Sup RX + User RX)

MPU and a Real Time Operating System (RTOS)

When using an RTOS, the application is broken up into tasks or threads. Communication between tasks is frequently accomplished with Queues or Events that are managed by the RTOS.

- This gives much more opportunity for enforcing partitioning at a fundamental level if the RTOS provides native support for the MPU.
- If the RTOS in use does not provide native support for the MPU, then we have a similar scenario to the previous arrangements except that we have a new problem, the code and data for the RTOS itself. This has to be trusted at the highest SIL level of the application.

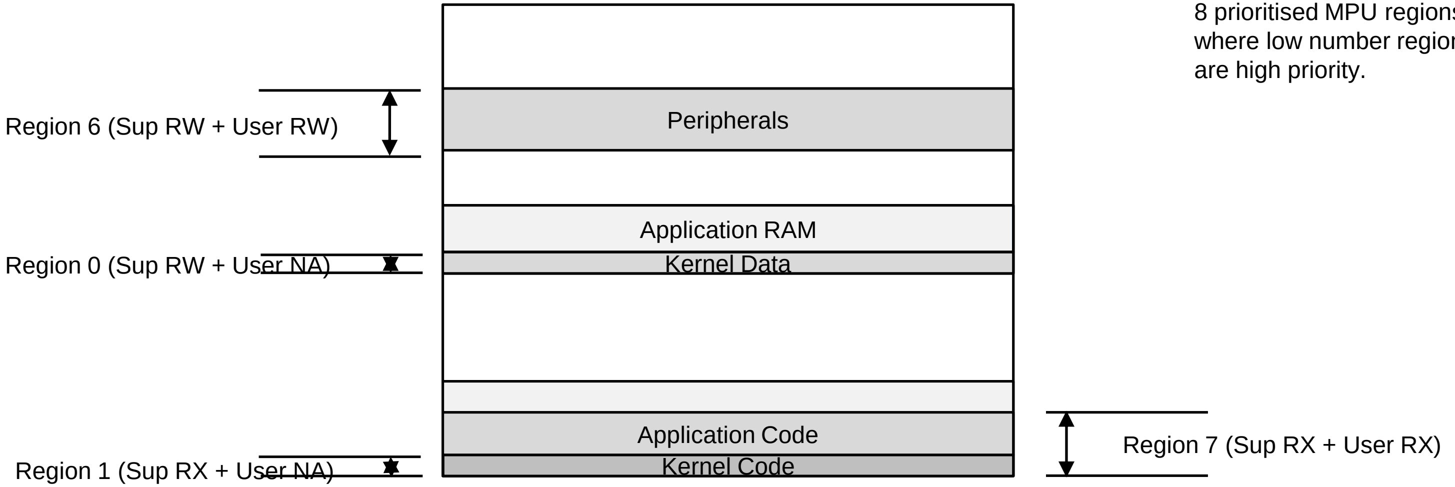

Region 5 (Sup RW + User NA)
UART
Region 2 (Sup RW + User RW)
Region 6 (Sup RW + User NA)
General Access RAM
Privileged (Safety) RAM
Region 3 (Sup RW + User RW)
RTOS Code and Data
Region 7 (Sup RX + User NA)
Unprivileged Code
Privileged (Safety) Code
Region 4 (Sup RX + User RX)

Using an RTOS with Integrated MPU Support (e.g. SAFE**RTOS**)

Integrated MPU Support allows us to:

- Protect the RTOS Kernel Code and Data from unauthorised access using fixed regions that are setup during system initialisation.

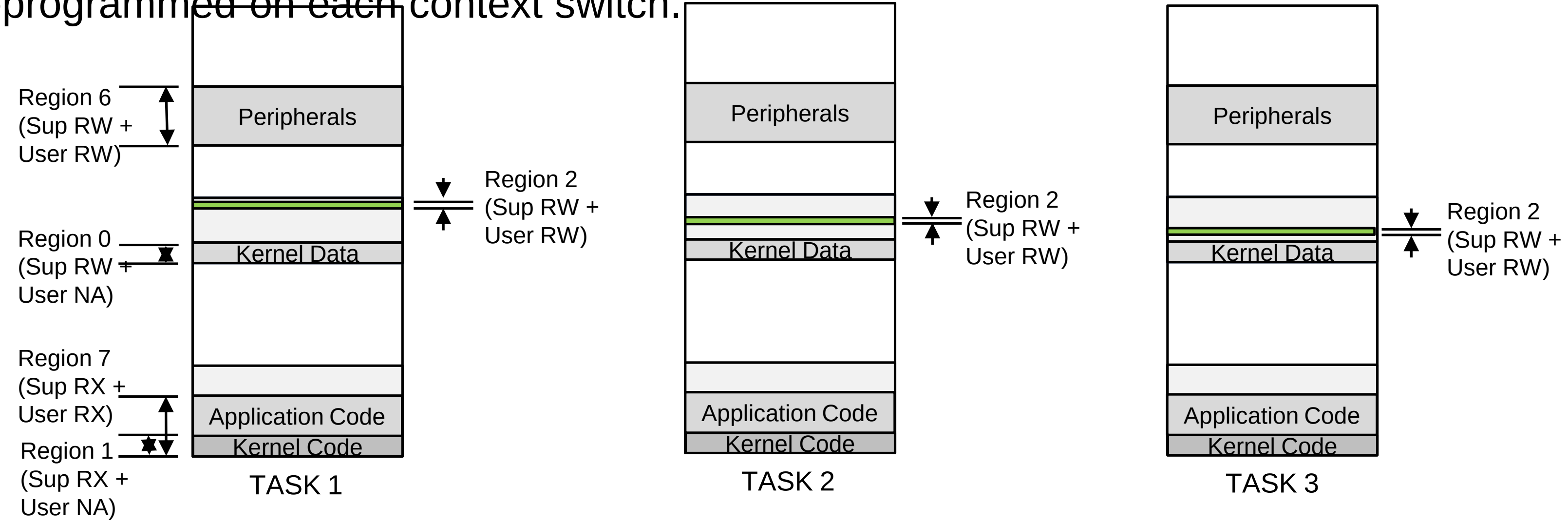
Note: This example assumes 8 prioritised MPU regions where low number regions are high priority.


Region 6 (Sup RW + User RW)
Peripherals
Application RAM
Region 0 (Sup RW + User NA)
Kernel Data
Application Code
Region 7 (Sup RX + User RX)
Region 1 (Sup RX + User NA)
Kernel Code

Using an RTOS with Integrated MPU Support

Integrated MPU Support allows us to:

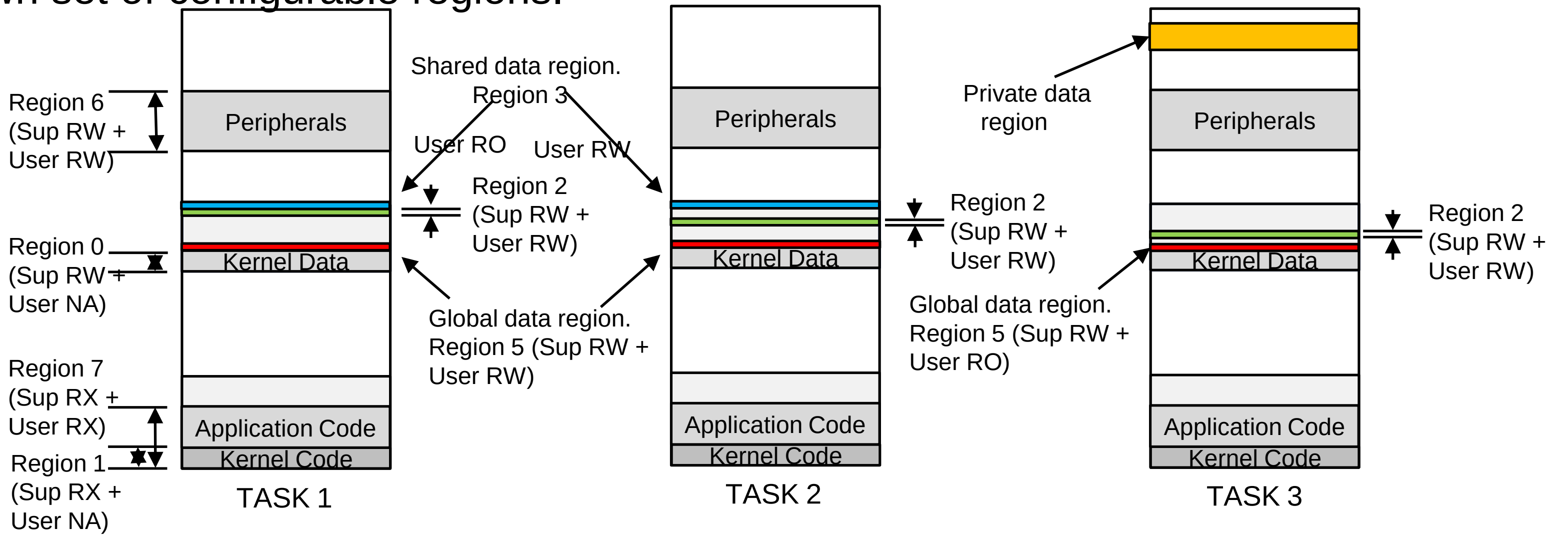
- Provide a degree of task isolation by protecting user task stacks. Some MPU regions are reprogrammed on each context switch.



Using an RTOS with Integrated MPU Support

Integrated MPU Support allows us to:

- Reprogram the MPU during each context switch and therefore allow each task to have its own set of configurable regions.



Common Problems with Using the MPU

- Human nature – designing the memory map, managing the linker script etc. are non trivial especially when the development spans multiple teams. The tendency to allocate everything to shared data or declare the task privileged has to be managed.
- Not enough regions – even when everybody is on-board, most microprocessors do not offer enough memory regions to really control the memory space.
- Interrupt handlers – will always be privileged and therefore typically are not constrained by the MPU. Or if they are constrained then they will run with permissions in force when the exception occurs.
- What to do when an error is detected – in a microprocessor based system, managing to recover from a major fault condition such as a hard fault or MPU fault is difficult.

Summary

- Why an MPU can be helpful in systems that need to be certified as safe or secure.
- Multi Core and Multi Processor Architectures as a means of partitioning and how the MPU is still useful.
- What we can do with the MPU to partition and protect.
- How an RTOS that natively supports the MPU gives greater scope with regards to partitioning.