IMPLICIT SAFETY

GENERIC SAFETY SOFTWARE APPROACH

DR. DAVID BACA
OCTOBER 23, 2019

7TH SCANDINAVIAN CONFERENCE ON SYSTEM & SOFTWARE SAFETY

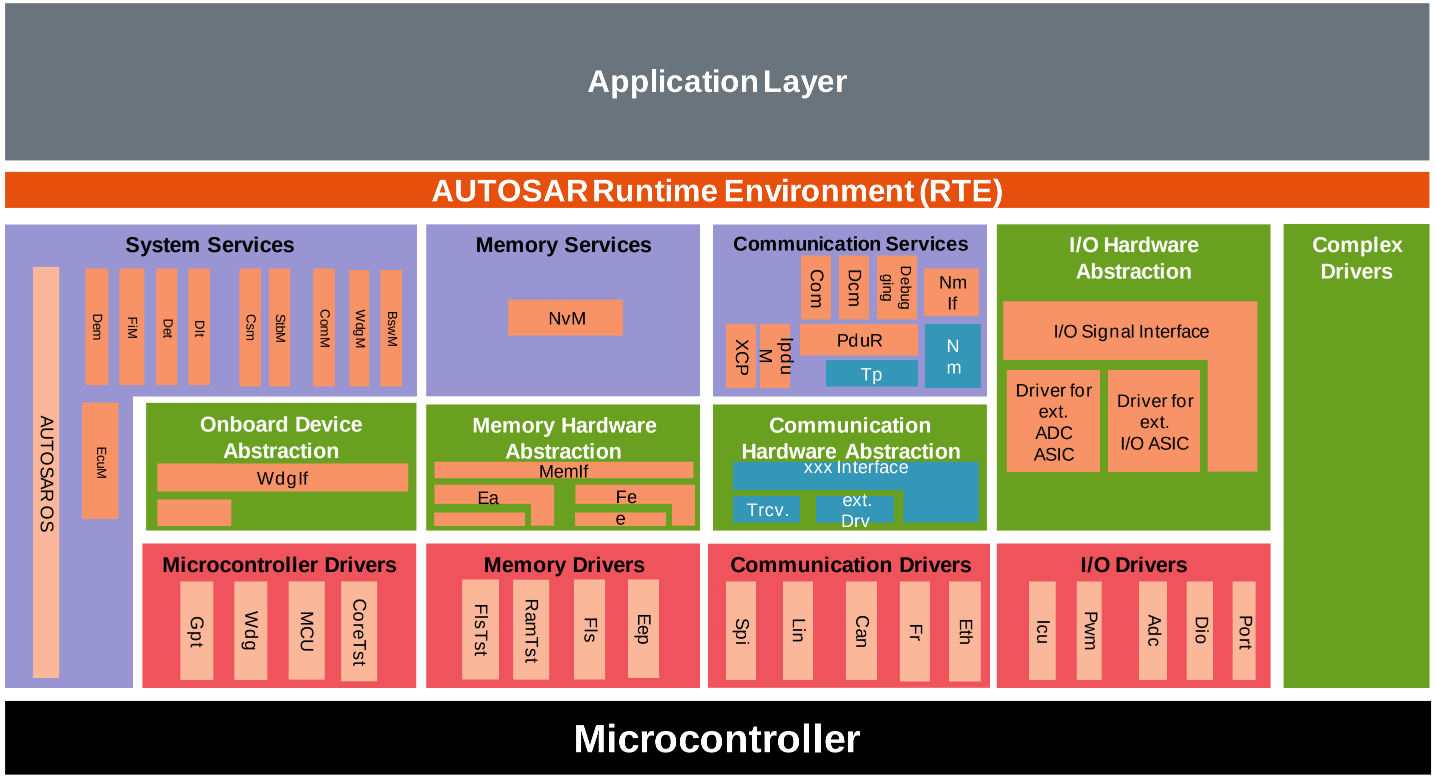
PUBLIC

SECURE CONNECTIONS FOR A SMARTER WORLD

Safe Generic SW – Safety Elements out of Context in ISO26262

SEooCs:

- generic SW elements deployed to different applications and also to different customers.
- not developed in a context of a particular system (item).
- safety requirements are assumed
- integration requirements imposed
- an example are AUTOSAR Basic SW components



AUTOSAR Layer Architecture

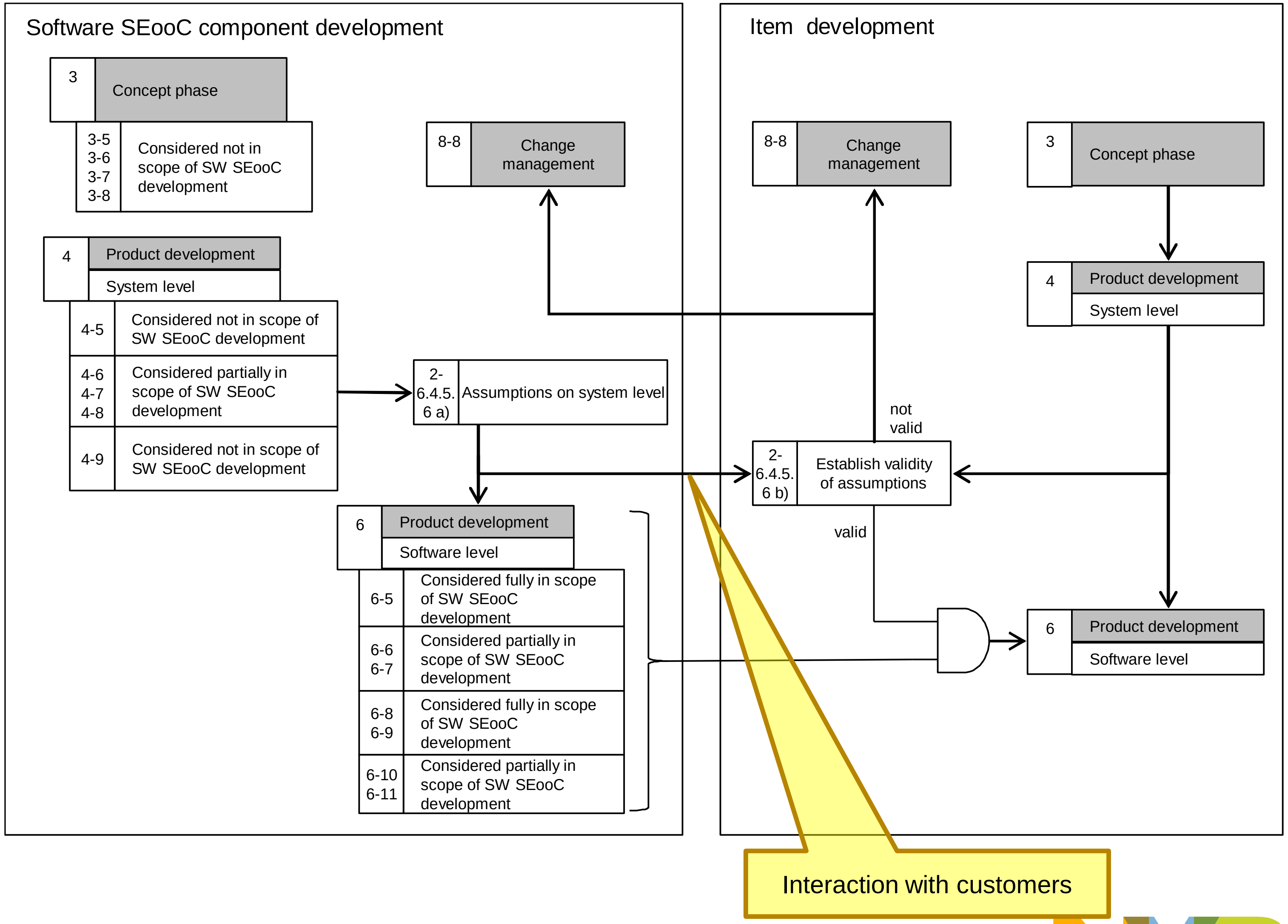
SW SEooC development in ISO26262

SEooC definition:

- assumed purpose and role
- assumed surrounding architecture
- assumed integration environment
- assumed higher level safety requirements
- derived safety requirement for SEooC

SEooC integration:

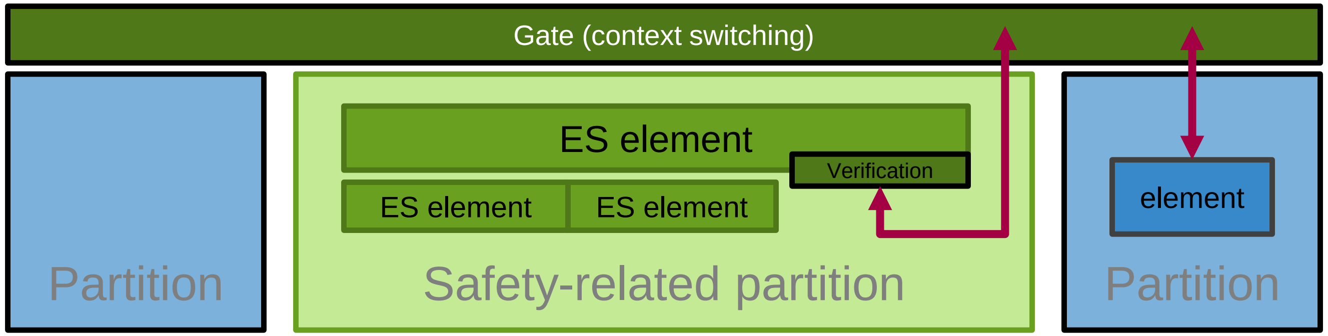
- check validity of assumptions
- perform impact analysis if assumptions do not fit



Copyright ISO26262 Part 10

Motivation

- Development of generic SW elements that are on different safety paths but that are not responsible for safety.
- Enable integration of SW elements into safety partitions to avoid gated calls.



- Generic approach for any SW element.
- Easy integration into any SW safety architecture.

Faults Considered by Safety SW

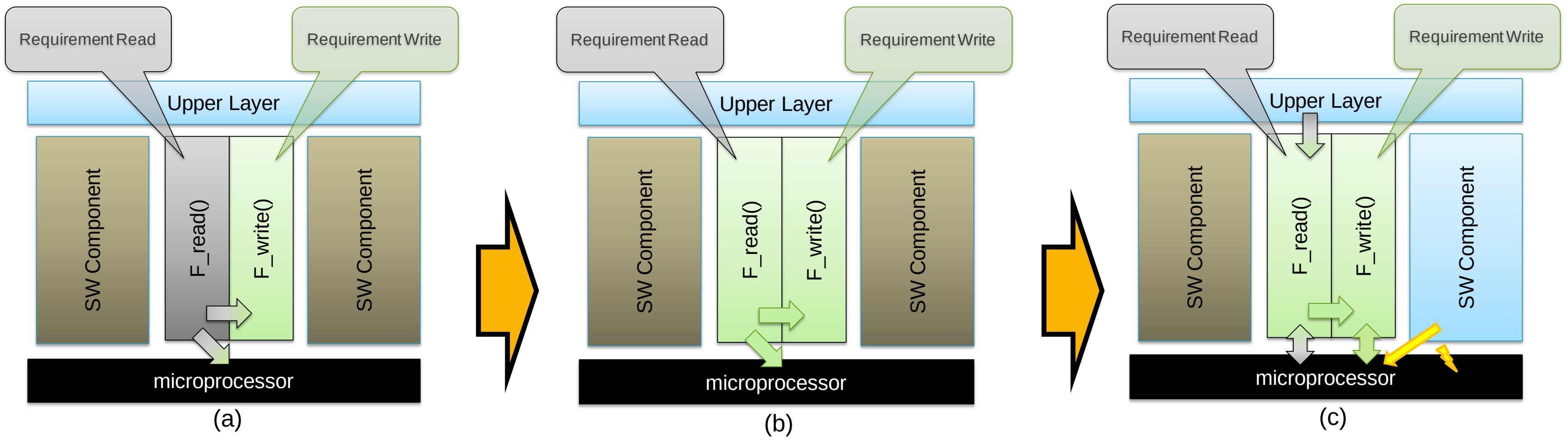
Faults	Causes	Measures
Hardware random faults	Caused by transient or permanent HW failures.	• HW-based detection if exists • **SW-based fault detection**
Software systematic faults	Caused by mistakes in SW development process	• Compliant ISO 26262 process • **Architectural measures**
Software interference	Fault propagation from lower ASIL SW component. Caused by HW random faults or SW systematic faults.	• **Interference prevention** • **Interference detection**

Implicit Safety Derivation

The derivation of implicit safety by example:

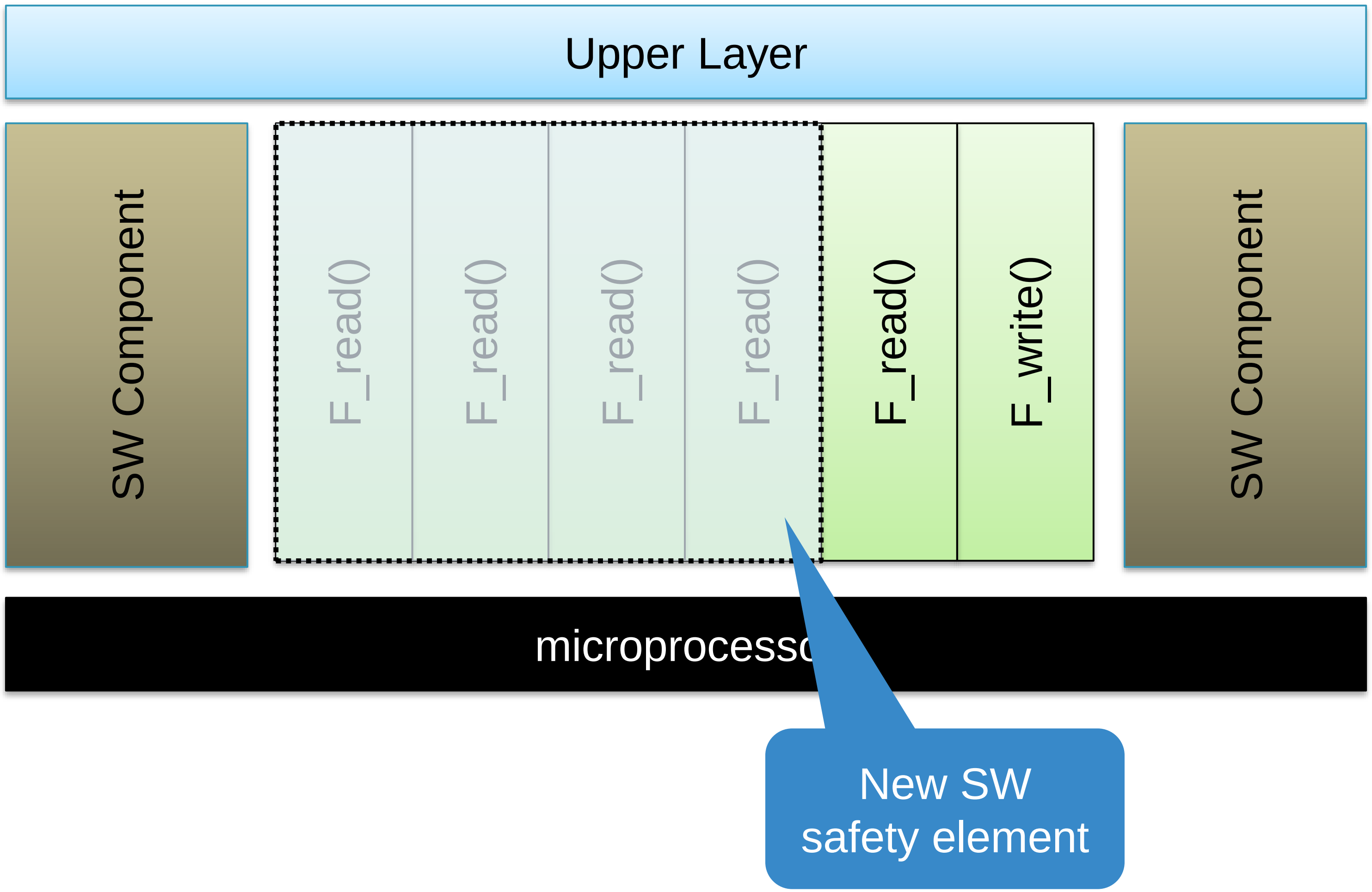
Assume a simple element with functions F_read and F_write where each is defined by a single respective requirement.
The requirement for F_write is safety related. See Figure (a) where green denotes the safety-related part and gray non-safety-related.

F_read can affect F_write including the registers F_write writes to; therefore it has to be developed according to ISO 26262 as well – Figure (b). HW faults and SW interference can impact integrity of F_read which can then corrupt F_write.
As a result, F_read needs to be resistant to such faults – Figure (c). However, not all HW faults are of concern; faults that impact only the correctness of values returned by F_read do not have to be detected as long as those values are valid. Invalid returned values can corrupt upper layers.



IS element consists of functions such as F_read only. Since F_read functions do not corrupt each other and they also do not corrupt any other function they coexist with in the same partition, an IS element can coexist with any other safety-related element of the same ASIL. The IS requirement states that neither inner integrity corruption nor external integrity corruption occurs.

Forming and Implicit Safety Element



Implicit Safety

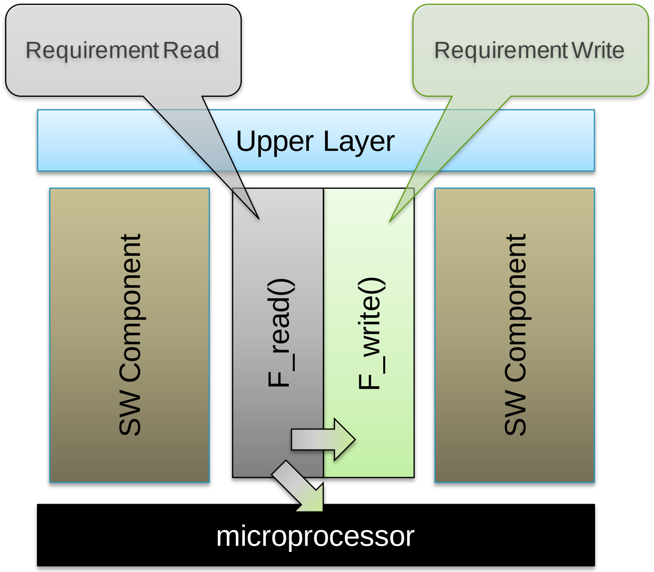
The **Implicit Safety (IS) requirement** is defined as follows:

A safety-related element shall not corrupt its own integrity and the integrity of other elements – ASIL-D.

Element's integrity is defined as the element being in a valid state

Implicit Safety element is a safety-related SW element that is allocated the IS requirement and the IS requirement is the only safety-related requirement allocated to the element

Explicit Safety element is an element that is allocated one or more safety-related requirements but not IS.



FMEA – rule-based

Applicable	Fault Effects	Failure Mode	FM explanation	Causes	Measures (Requirements)
Yes	Out of range results corrupting the environment. Crash Memory corruption	Register out of range	Register contains unexpected value	Register fault IP fault	Mask register value Use default value
Yes	Out of context calculation corrupting the application. Memory corruption	Interrupt out of order	Hardware Interrupts triggered outside normal conditions	Spurious interrupt Odd-behaving IP	Check interrupt conditions Check driver status
No	Deadlock	Peripheral status frozen	IP does not complete the operation or is unable to signal its completion	IP or register faults	Protect waiting loops with maximal iteration counts
…	…	…	…	…	…
…	…	…	…	…	…

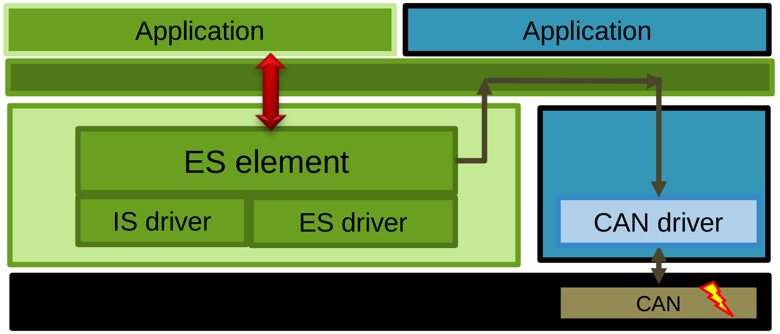
Implicit Safety (IS) Advantages

IS CAN driver example

Non-safety solution

No safety mechanisms in CAN driver
Safety mechanisms in the interface

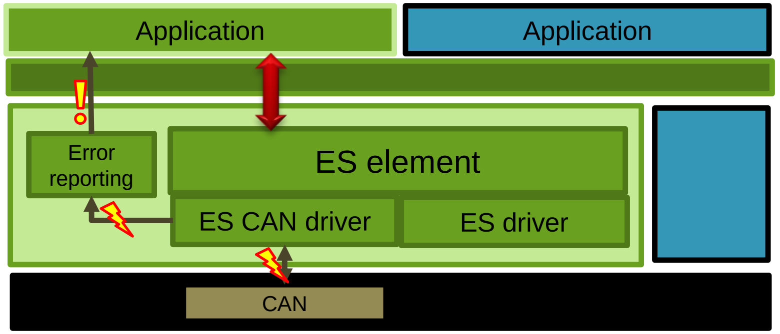
- Context switching
- Gate checks during context switching.
- Data exchange verification
- Timeouts



Standard Solution

Safety mechanisms (detect&report):

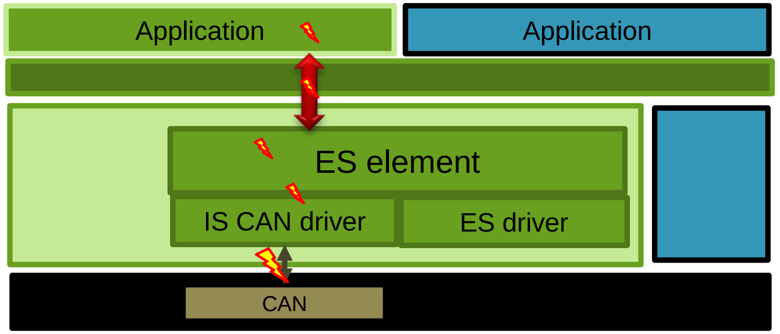
- Config Register read-back
- Config Register periodic check
- Invalid or inconsistent values check
- Stuck-at faults in status registers check
- Spurious Interrupts check
- Inconsistent behavior at HW/SW interface check



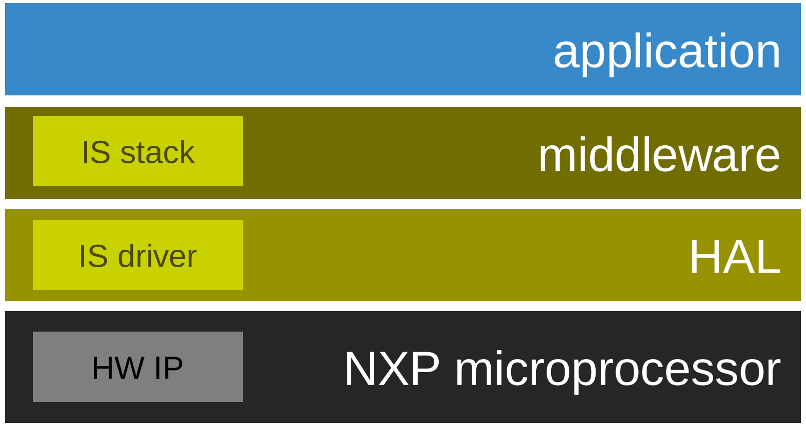
Implicit Safety Solution

Blocking mechanisms, resistance to:

- Invalid HW values (including stuck-at)
- Stuck-at faults in status registers
- Spurious interrupts



IS examples

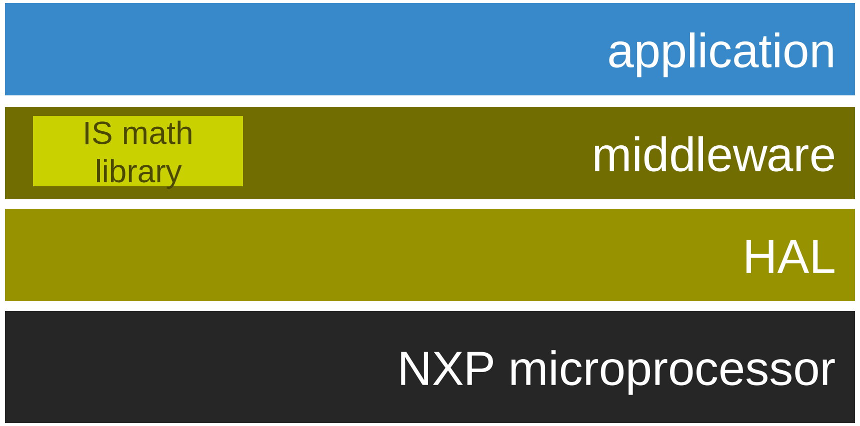


IS SW does the following:

- blocks HW faults, responses:
  - default value
  - timeout response
- ensures valid data exchange
  - by blocking HW faults
  - development process

Application performs:

- control flow monitoring (FfI)
- evaluation of responses and system reaction to failures



IS math library does the following:

- nothing special
- safety manual lists integration requirements

Application performs:

- control flow monitoring (FfI)
- plausibility checks if needed (safety analysis)

Conclusions

The **Implicit Safety (IS) requirement** is defined as follows:

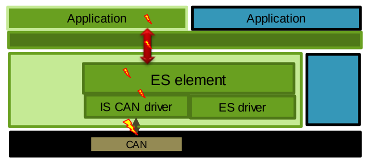
A safety-related element shall not corrupt its own integrity and the integrity of other elements – ASIL-D.

Implicit Safety

- provides a safety concept that allows development of any SW element as a safety-related element.
- does not compromise the generic aspects of the SW element.
- enables efficient integration of safety SW elements into any safety application architecture.
- uses simple FMEA that makes the safety analysis easy.
- incurs very small execution and code size overhead
  - robustness measures in the SW element
  - application monitoring but that is usually in place anyway

Implicit Safety Solution

Blocking mechanisms, resistance to:
- Invalid HW values (including stuck-at)
- Stuck-at faults in status registers
- Spurious interrupts



SECURE CONNECTIONS
FOR A SMARTER WORLD