Runtime redundancy and monitoring for critical subsystem/components

Presenters

Gabriele Paoloni
Senior Principal Software Engineer
Functional Safety
In-vehicle OS

Red Hat

Daniel Bristot
Principal Software Engineer
real-time/trace/verification
and other things in kernel

Red Hat

Agenda:

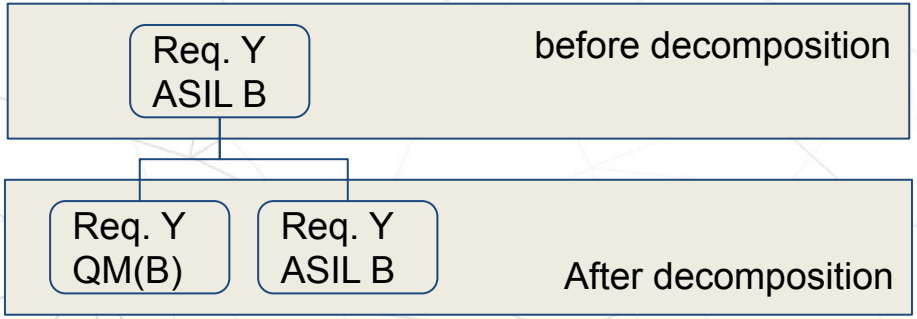
- ASIL Decomposition: what it is, when and how to use it
- Is it applicable to Linux drivers/subsytem?
- When it is worth applying it?
- How?...Runtime Verification Monitors
- What is the performance penalty?

ASIL Decomposition: what it is, when and how to use it

The ASIL decomposition is the methodology of decomposing a safety requirement into redundant safety requirements and allocating such safety requirements to sufficiently independent design elements.

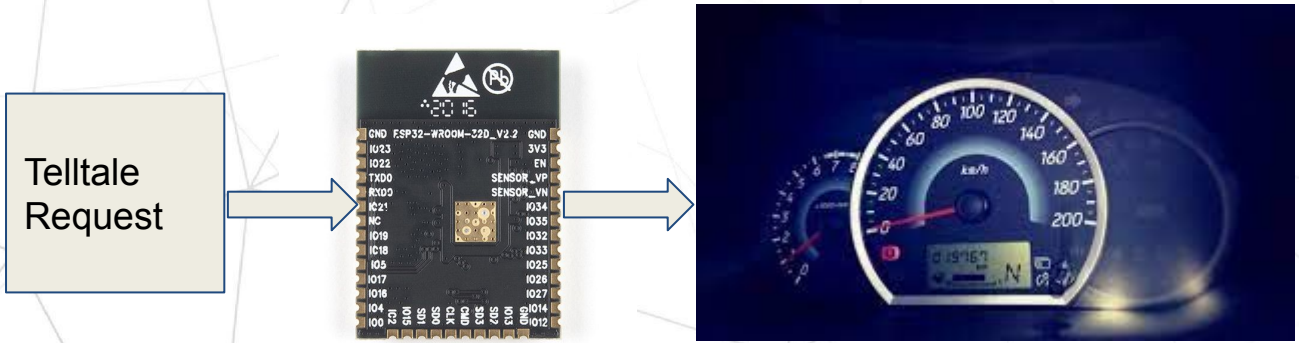
In the process of decomposition the ASIL allocated the each independent element can be lower than the ASIL allocated to the top level one as long as the sum of the ASIL allocated to the decomposed requirements equals the ASIL of the parent requirement.

E.g.



ASIL Decomposition: what it is, when and how to use it.
>> Example

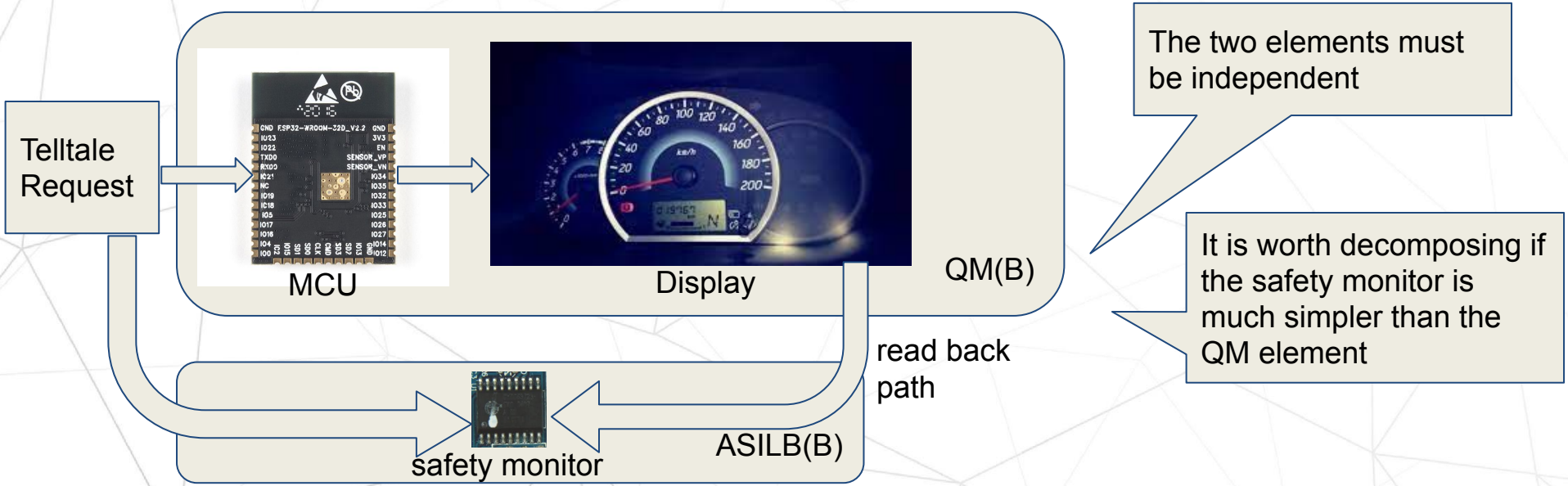
Top level requirement: Following car failure the display system shall project the respective correct telltale within 100msec (ASILB)



ASIL Decomposition: what it is, when and how to use it.
>> Example

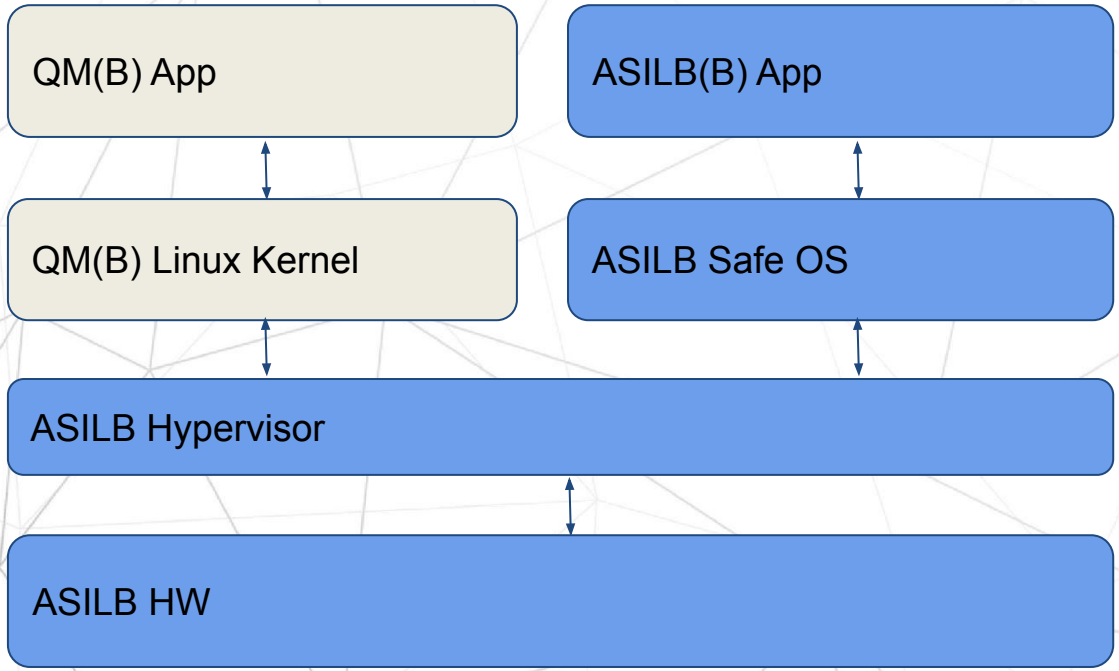
Decomposed requirements:

- Element A: following a car failure the display system shall project the respective correct telltale within 100msec (QM(B))
- Element B: Following a car failure the dashboard display should be monitored to check the requested telltale to be projected within 100msec (ASILB(B))



ASIL Decomposition in the Linux Kernel

External (to Linux) independent monitor



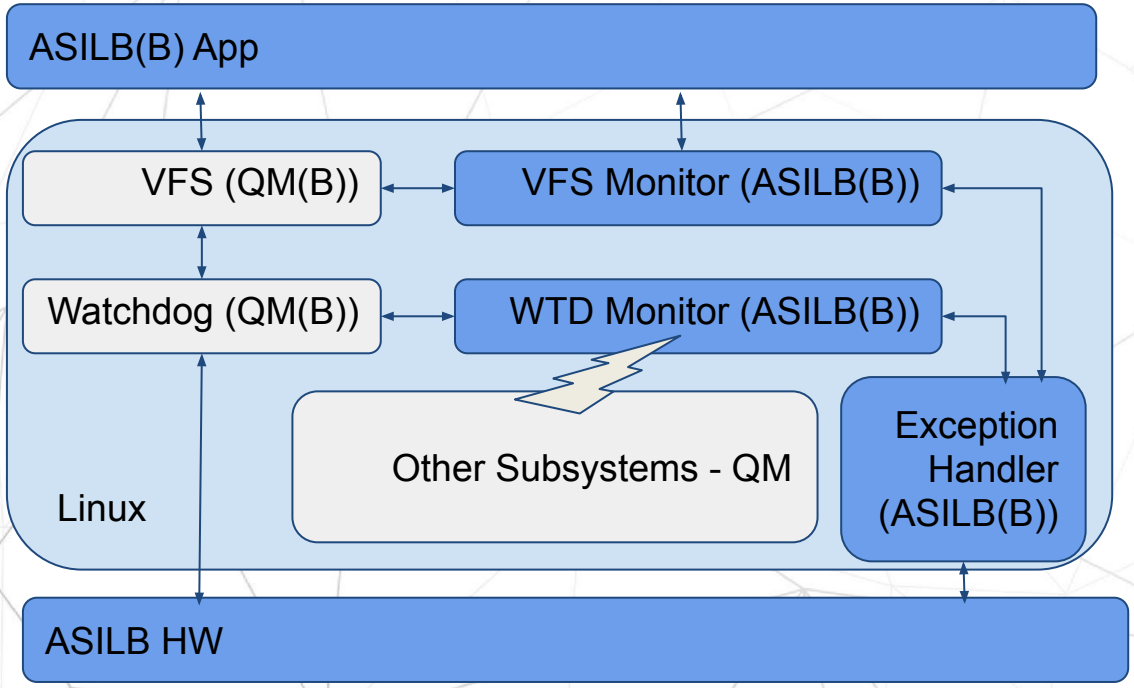
This is a common way of using Linux systems in Functional Safety.

In this example we need an hypervisor, 2 operating systems and respective independent applications.

So the BOM is significant

ASIL Decomposition in the Linux Kernel

What if we decompose inside the Kernel ?



- Each subsystem allocated with a safety requirement is monitored by a safety monitor running inside the Kernel. So…
  - how to design the monitors?
  - how to make sure the monitors are independent WRT the monitored subsystems/drivers?
  - how monitors react to possible interference coming from any QM subsystem/driver?

Discussion Points

- How to make sure the monitors are independent WRT the monitored subsystems/drivers?
- How monitors react to possible interference coming from any QM subsystem/driver?
  - Temporal Interference: monitors can regularly pet an external wtd
  - Communication interference: monitors by design only communicate with the monitored subsystems (QM(x)) and with the exception handler (ASIL(x))
  - Spatial interference: The table of the states to be monitored can be read-only (hence we leverage the HW MMU to raise exception in case of access)