Using Pre-Existing Open-Source Software under ISO 26262: Lessons from Zephyr RTOS and the Xen Hypervisor

Roberto Bagnara[1,2] Nicola Vetrini[2]
Abramo Bagnara[2] Stefano Stabellini[3]

[1]University of Parma, Italy [2]BUGSENG, Italy [3]AMD, USA

24th Workshop on Automotive Software & Systems
Bologna, Italy
May 21, 2026

bugseng

Disclaimer

I am a member of the *MISRA C Working Group*, the *MISRA SQM Working Group*, and of ISO/IEC JTC1/SC22/WG14, a.k.a. the *C Standardization Working Group*

however

the views expressed in this presentation are mine and my coauthors' and should not be taken to represent the views of such working groups

bugSeng

Functional-Safety Standard Compliance of Existing Software

Greatest benefits can be obtained when the process is embraced since the very beginning of the project

This is sometimes not possible, for various well-known reasons

One that is becoming more and more common is the incorporation into safety critical systems of open-source software (OSS)

bugSeng

Why Incorporating OSS in Critical Systems?

Access to innovative technologies and products

Avoidance of vendor lock-in

Leverage of larger ecosystems allowing for cost reduction and shorter development cycles

Lower layers in a typical software stack are particularly suited to this move:

- Trusted Firmware Arm, Google, ST, Renesas, NXP, Linaro, ...
- Xen Project Hypervisor AMD, Arm, AWS, EPAM, Linux Foundation, ...
- Zephyr RTOS Analog Devices, Google, Intel, Meta, NXP, Linux Foundation, ...
- ELISA Boeing, Red Hat, Arm, Bosch, Huawei, Linux Foundation, ...

bugSeng

Challenges in the Safety-Qualification of OSS

Notable issues are, typically, with:

Processes Some processes are there, but do not comply with functional-safety standards (e.g., technical safety concept is skipped, formal inspections are not conducted)

Artifacts In most cases they are largely missing (e.g., traceable requirements and tests)

Governance Based on volunteer work, only part of the community may care about safety qualification: hence, liability left to end users

bugSeng

Challenges in the Safety-Qualification of OSS (cont'd)

High configurability Want to safety-qualify only a small subset: how is that defined and how to deal with common code?

Pace of development Many OSS projects move too fast: maintaining all safety artifacts up-to-date is hard

Virtuoso programming Truly talented programmers take pride in writing clever, hard-to-understand code, but safety-critical code has to be boring!

bugSeng

Approaches to the Safety-Qualification of OSS

Some possibilities, which are not mutually exclusive:

Retrofitting safety Treat the OSS component as a PSAE; create/adapt requirements; make them traceable to the implementation and tests; adopt and enforce a suitable coding standard

Fork Not suitable if code is not fit for purpose as is, but might be inevitable in the final phases of certification

Refrain from safety qualification E.g., qualify only a small component that monitors the correct operation of the rest

bugSeng

ISO/PAS 8926:2024: The PSAE View

ISO/PAS 8926:2024 addresses a pre-existing software architectural element (PSAE): already available COTS or custom software, not specifically built-to-order, and not developed to conform with ISO 26262

The argument is for a specific version and configuration of the PSAE integrated into a target software architectural design developed under ISO 26262

The PSAE is safety-related when:

- software safety requirements are allocated to it
- errors in its functions or properties can violate safety requirements

For OSS, this shifts the question from "Is the project certified?" to "Which configuration are we integrating, in which architecture, and under which assumptions?"

In this talk, ISO/PAS 8926 is a general reference framework; it is not presented as the certification basis of the Xen work discussed later

ISO/PAS 8926:2024: Integration Argument

The framework starts with an impact analysis: allocated ASILs and software safety requirements, interfaces, target environment, resources, timing/concurrency, configuration data, and existing evidence

Then the PSAE is classified by two uncertainties:

Provenance process evidence: from standard-based development, to acceptable gaps, to insufficient evidence

Complexity from no high complexity, to high but manageable complexity, to not manageable

The result drives the work:

- Class I: software-component qualification under ISO 26262-8
- Class II: tailored activities: retrospective design analysis, refined requirements, safety-oriented analysis, freedom from interference, and verification of the integrated use

bugSeng

MISRA Compliance and Functional Safety

Safety-critical software has to be:

- traceable to documented requirements
- verifiable and verified by means of peer review, static and dynamic analyses, and testing

Use of C and C++ is allowed provided that:

1. they are standardized
2. programs have a well-defined semantics: no undefined or unspecified behaviors
3. translators are qualified
4. programs refrain from using error-prone constructs
5. program units (e.g., functions) are of limited complexity

MISRA C and MISRA C++ address these concerns

bugSeng

MISRA Compliance of Existing Code

Legacy code comes in two flavors: adopted code and simply existing code

Adopted code is fit-for-purpose as is and needs not to be read understood or modified: simplified MISRA compliance

Simply existing code will be adapted and modified: standard MISRA compliance

bugSeng

MISRA Guidelines Tailoring

Let us take it for granted that existing code that is candidate to inclusion in safety-critical system is of very high quality

MISRA makes it very clear that code quality always comes first

Tailoring of the MISRA guidelines is essential:

- selection of the (non-mandatory) guidelines to comply with
- global deviations for required guidelines where justifiable
- partial deviations for required guidelines to adapt them to the project

bugSeng

Tailoring of MISRA Guidelines: E.g. Rule 10.1

Multiple rationales: unspecified and undefined behavior, as well as implementation-defined behavior and developer confusion

But the following are safe:

- value-preserving conversions of integer constants
- shifting non-negative integers to the right, if the shift count is non-negative and not too large
- shifting non-negative integers to the left, if the shift count is as above and if the result is still non-negative
- bitwise logical operations on non-negative integers, even if the operands are of signed type
- implicit conversion to Boolean for logical operator arguments
- two's complement behavior of bitwise ops on signed integers can be assumed to be known by all developers

. . .

bugSeng

Conditional Compilation via Kconfig

Several OSS projects (including Linux, Zephyr and Xen) use the Kconfig DSL for managing quite complex configurations

Preprocessor-based conditional compilation is explicitly not recommended:

```
switch(x) {
  case 1:
    f();
    break;
#if IS_ENABLED(CONFIG_FOO)
  case 2:
    foo_enabled();
    break;
#endif
  default:
  break;
}
```

bugSeng

Conditional Compilation via Kconfig (cont'd)

Instead, this is what is recommended, because:

- it is easier to write and read, especially in the presence of nesting
- syntax is always checked, also in excluded parts

```
switch(x) {
  case 1:
    f();
    break;
if(IS_ENABLED(CONFIG_FOO)) {
  case 2:
    foo_enabled();
    break;
}
  default:
  break;
}
```

bugSeng

Reachability and Decidable Guidelines

MISRA view: excluded code should be removed by preprocessing

Linux view: nothing changes if excluded code is removed by later compiler translation phases

Note that decidable guidelines apply to all code that is present after preprocessing, including code guarded by `if(0)`

A compromise is possible provided that:

1. rules are deviated taking into account the following points
2. the compiler does indeed eliminate code that is guarded by `if(0)` unless jumping inside it is possible
3. no jumps inside code excluded by `if(0)` are possible from outside: true if a bunch of other MISRA guidelines are complied with!

bugSeng

The Xen Hypervisor

Xen is a type-1 hypervisor with a microkernel design

Xen is a open source community project under Linux Foundation, license GPLv2

ARM64 and x86-64 fully supported, RISC-V in progress

Xen embedded features highlights:

- Real-time and cache isolation in software (cache coloring)
- Fast parallel VMs booting and static partitioning (dom0less)
- Virtio and Xen PV Drivers support with freedom from interference
- Cortex-R52 and R82 support, Xen on MPU systems

bugSeng

Qualification of the Xen Hypervisor

Xen runs independently at a higher privileged mode, separately from any operating system: everything else runs on top of Xen as a Virtual Machine

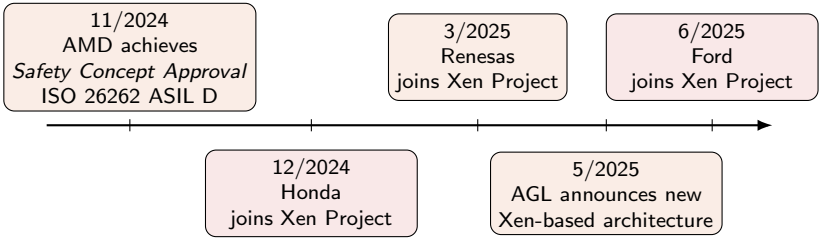
Xen is instrumental in achieving isolation/independence/freedom-from-interference among software components running on the same hardware platform

Being one of the most critical components in a system, it needs to be qualified for safety-related use

BUGSENG was selected by AMD to work with the Xen community toward MISRA compliance of selected Xen configurations for AMD's x86-64 and ARM64 architectures (99% complete!)

bugSeng

Xen Momentum in Automotive



Automotive and embedded stakeholders are actively and increasingly investing in Xen

bugSeng

Xen: From IVI to Vision Hub and Industrial

In-Vehicle Infotainment (IVI)

- graphics
- Virtio & PV devices
- richer configurations

Vision Hub & Industrial

- real-time
- static partitioning
- simpler configurations

Xen supports the full spectrum: rich mixed-criticality deployments at one end, and static, low-footprint configurations at the other

bugSeng

Deployment Capabilities Across the Spectrum

Richer IVI-style configurations

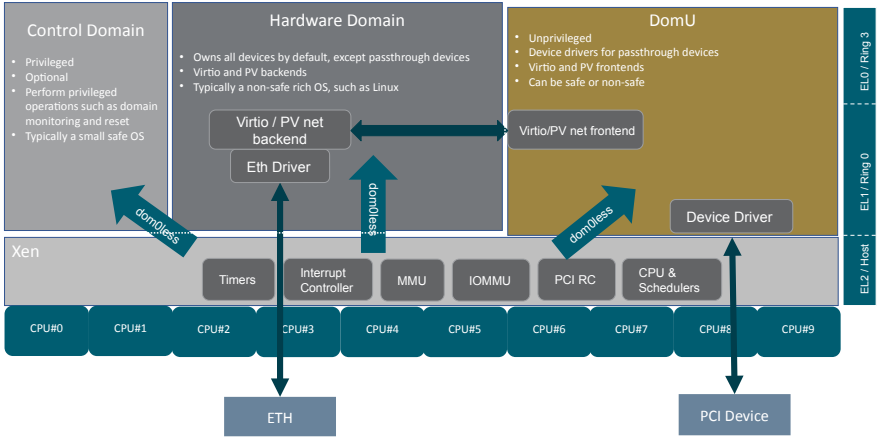
- Dom0less/Hyperlaunch for faster boot times
- Several Virtio protocols supported: Virtio-NPU, Virtio-net, Virtio-block, Virtio-gpu, Virtio-sound, Virtio-media, Virtio-tee, Virtio-gpio, Virtio-i2c, Virtio-input
- Passthrough available

Static/safety-oriented configurations

- Dom0less/Hyperlaunch for static partitioning
- Hardware passthrough to domains
- Fully dedicated pCPUs to vCPUs: no scheduling
- 3 μs interrupt latency under interference
- Minimal Xen footprint at runtime
- Xen on microcontrollers without MMU, such as Cortex-R52 and R82
- Virtio available

bugSeng

Modern Xen Architecture for Safety



Safety-Certifying the Xen Hypervisor

AMD safety certification work is near completion:

- IEC 61508 SIL 3 and ISO 26262 ASIL D
- both ARM64 and AMD x86-64
- Phase I completed; Phase II nearing completion
- based on upstream community processes and the upstream codebase

Already fully public and Open Source:

- MISRA C rules adopted and documented
- all Xen code changes are public
- safety requirements being upstreamed
- test harness and examples: requirement-based tests, QEMU fault injection, and fuzzing
- Xen Project GitLab CI open to community members

bugSeng

AMD Certification Work: Status

Phase I – completed, November 2024

- Configuration, change, and document management plans accepted
- Custom V-model lifecycle established
- Software safety requirements: concept, examples, traceability
- Software architecture specification: concept, examples, traceability
- V&V plan at high level
- Software validation and verification plans
- Software tools classification
- *Safety Concept Approval* achieved

Phase II – nearing completion

- MISRA C: 99% complete and upstream
- Requirements and architecture specifications
- Validation and verification tests
- 100% code coverage
- Safety case and safety manual
- Final assessment and certification

bugseng

Xen Project Community Collaboration around Safety

Active contributors

- AMD
- EPAM
- Renesas
- BUGSENG
- Boeing
- Resiltech

Growing interest from automotive, embedded, and robotics

Safety Pilot

- AMD, Renesas, EPAM
- joint effort to accelerate safety work
- precursor to a broader Xen Safety Initiative

A large share of the safety artifacts is in the pilot

Safety artifacts

- safety requirements
- architecture specifications
- fault injection tests
- fuzzing
- FMEA
- tool qualification

bugSeng

Why Open Collaboration Matters

- Certification tracks the same code everyone uses: no private safety fork to maintain
- Updating the certification remains a bounded effort because upstream changes flow back into the safety baseline
- Community review increases assurance through an independent panel of maintainers across vendors
- Downstream adopters start from a public, reviewed, MISRA-clean codebase
- Shared effort keeps a single, current source of truth, lowering per-company cost and making timelines more predictable

bugSeng

Conclusion

OSS is not disqualified by its history, but the safety case must be honest about what was not developed under ISO 26262

ISO/PAS 8926:2024 gives a useful structure: argue about a specific PSAE version/configuration in a target architecture, not about an OSS project in the abstract

The hard work is the impact analysis: allocated requirements, interfaces, resources, timing/concurrency, unused or unintended functionality, assumptions, and limitations

MISRA compliance is valuable because it reduces complexity and uncertainty, but it is only one part of the broader classification, suitability, and verification argument

Open collaboration can make that argument stronger and cheaper, provided the evidence follows the upstream code that adopters actually use

bugseng