Arcturus Software Architecture Whitepaper

This document was generated for investor information only, and is not for public release

Patrick Bellamy, Jack Ulbrich-Baker

1 Overview

Arcturus is a vertically-integrated compute platform engineered for deterministic, certifiable, safety-critical workloads across spacecraft, UAVs, defence avionics, and autonomous systems. It integrates a deterministic hardware stack, a closed SDK, and an automated verification pipeline that produces certification-ready artefacts by default.

Across aerospace and other safety-critical domains, multiple studies and industry reports observe that achieving compliance with standards such as DO-178C, DO-254, ECSS, ISO 26262, and IEC 61508 often consumes a substantial share of programme cost and schedule, frequently in the range of 30–50% of total engineering effort once rework and documentation are included.[16, 17, 13, 1, 6] This burden is driven not only by testing, but by the need to demonstrate traceability, structural coverage, configuration control, and process assurance.[20, 19] Arcturus is designed explicitly to attack this cost structure by embedding certification activities into the development flow rather than treating them as a separate phase.[14]

This document describes the high-level software architecture, design principles, workflows, certification mapping, safety model, and roadmap. It is explicitly designed for external use: proprietary design elements such as RTL, firmware internals, scheduling algorithms, and low-level pipeline implementations are intentionally *not* disclosed.

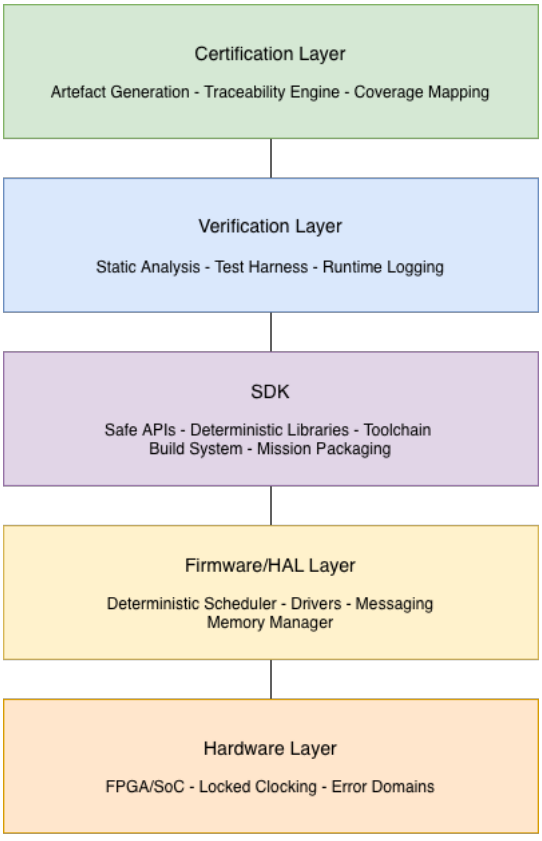
Key investor anchors:

- Arcturus turns 12–18 months of manual certification work into days of automated evidence generation by systematising activities that are today performed by hand.[13, 26, 27]
- Determinism is enforced by construction, not retrofitted after the fact, following principles from hard real-time and time-triggered systems.[9, 10, 34]
- The platform is designed as reusable infrastructure, not one-off consulting, aligning with trends in hardware-accelerated and tool-driven assurance frameworks.[36, 38, 51]

2 Architecture Summary

Arcturus unifies four tightly-coupled layers into a single controlled compute environment:

1. **Deterministic Hardware Layer** – FPGA prototype today and future custom SoC, engineered for predictable execution and safety partitions.[35, 36, 37]
2. **Firmware and HAL Layer** – Minimal, closed firmware enforcing deterministic scheduling, disciplined memory use, and controlled I/O, consistent with partitioned real-time architectures such as ARINC 653.[11, 12]
3. **Arcturus SDK** – A safe, constrained development environment that prevents nondeterminism at compile time and embeds verification hooks into the development workflow, drawing on best practice in hard real-time programming and high-level synthesis flows.[9, 38, 39]
4. **Verification & Certification Pipeline** – Automated tooling that transforms runtime artefacts, test results, traceability maps, and build metadata into DO-178C/DO-254-aligned evidence sets, inspired by continuous certification research and modern V&V tooling.[14, 13, 19]



This structure ensures that developers interact with a clean SDK boundary, while determinism, safety, and evidence generation are enforced by the underlying stack. This mirrors layered architectures used in certified avionics and space software, where clear separation of application logic, runtime, and hardware abstraction is essential for compositional assurance.[1, 5]

3 Design Principles

These principles provide the intellectual backbone of the platform and demonstrate depth without exposing implementation-level IP.

3.1 Determinism by Construction

Arcturus treats determinism as a first-class invariant, not an afterthought. In line with established real-time theory, predictable behaviour is achieved by controlling time, resources, and interference rather than trying to measure or bound emergent complexity after the fact.[9, 10] The goal is to achieve reliable computation while maintaining performance.

- Static or bounded memory allocation; no unbounded dynamic allocation, reflecting guidance in standards such as DO-178C and ISO 26262 for avoiding unbounded resource usage in high-criticality software.[1, 8]
- Restricted control-flow constructs and runtime features that can introduce nondeterminism (unbounded recursion, uncontrolled dynamic dispatch, unrestricted threading).
- Fixed-frequency, controlled clock domains; no opportunistic DVFS or non-deterministic timing features, enabling classical worst-case execution time (WCET) reasoning.[9]
- Strict clock control, with no dynamic frequency control or voltage scaling.[2]
- A deterministic DMA controller with fixed-latency arbitration for deterministic memory management.
- No OS-style pre-emptive scheduling noise; execution is governed by a deterministic runtime inspired by time-triggered and cyclic-executive designs.[10, 11]
- A reproducible build pipeline: identical inputs produce identical binaries and evidence, easing configuration management obligations in DO-178C and IEC 61508.[21, 6]
- CPU core memory isolation options for error detection, prevention, and containment, while providing options for performance boosting for non-mission-critical code.

Investor anchor: predictable timing and behaviour reduce certification complexity by an order of magnitude because analysis can be performed on design-time artefacts instead of ad hoc measurements, as supported by both theory[9, 10] and regulatory guidance.[1, 32]

3.2 High performance hardware matched with high reliability software

In the new AI era, customers are looking for high performance hardware that is able to run complex AI networks on-board their spacecraft, rover, rocket, lander, satellite, and similar platforms in order to leverage the latest advances in artificial intelligence for space exploration and earth monitoring.[53]

The NASA requirements for hardware push manufacturers to improve their current space compute products significantly and include the following properties:

- Fault-tolerant heterogeneous RISC-V cores with high performance per watt.
- Radiation hard by design.

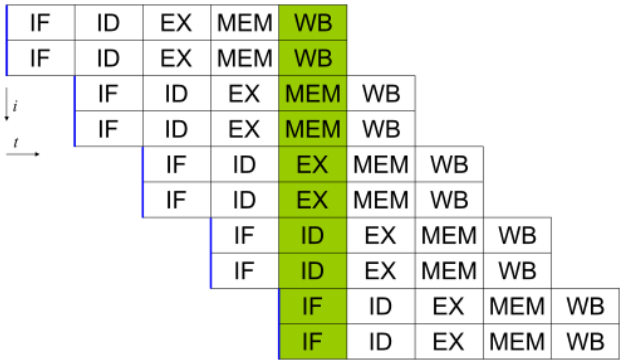


Figure 1: Superscalar instruction graph

- Fault tolerance for mission-critical applications.
- Post-quantum security.
- High connectivity:
  - Ethernet (10 GbE),
  - RDMA,
  - PCIe,
  - SpaceWire.

Investor anchor: building high performance hardware that is paired with deterministic code, a powerful SDK, and a quality developer and customer experience will empower legacy space companies to achieve more and grant access to smaller entities that might not previously have been able to access space.[53]

3.3 CPU Design

The core processing unit in Arcturus will be a dual-core lockstep control cluster used for safety-critical operations such as GNC, ADCS, and health monitoring.

- Based on a small, deterministic RISC-V ISA subset and dual-core lockstep (DCLS) with cycle-for-cycle comparison.
- Configurable split-mode for parallel verification or redundancy.
- Multiple instructions are executed concurrently in a pipeline without speculation to ensure timing determinism (see Figure 1).
- Hard real-time interrupt latency bounds are fully characterised.

3.4 Traceability-First Design

Traceability is designed into the platform. Safety standards consistently emphasise complete bidirectional traceability between requirements, design, implementation, and verification.[1, 6, 8] Empirical studies in safety-critical projects show that traceability gaps are a major source of late-stage rework and certification risk.[23, 24]

- Requirements, test cases, and code are linked via embedded metadata, enabling automated generation of traceability matrices similar to those used in regulated avionics and automotive programmes.[25]
- Build artefacts are tagged with provenance, configuration, and version information to satisfy configuration management and change control requirements.[20, 6]
- Execution traces and coverage outputs can be mapped back to requirements and design elements, supporting requirements-based testing and structural coverage arguments expected under DO-178C.[22, 19]

Investor anchor: this is the foundation of "certify as you build". Artefacts come out of the pipeline ready for audit, rather than being reverse-engineered from logs and code at the end of a programme.[14, 13]

3.5 Closed Hardware–Software Boundary

Developers do not program the hardware directly. They interact with the Arcturus SDK.

- The HAL and firmware interfaces are closed and controlled, following the pattern of partitioned real-time operating systems and separation kernels.[11, 12]
- The SDK exposes stable, safe abstractions rather than low-level registers or timing hacks, similar in spirit to domain-specific platforms for FPGA acceleration where template architectures and controlled APIs improve predictability.[38, 41]
- This containment reduces the certification audit surface and enforces consistent patterns across teams, aligning with guidance that favours pre-qualified components and well-defined interfaces.[1, 2, 5]

Investor anchor: this is a platform strategy, not a custom contracting model. The same constrained boundary and toolchain can be reused across missions and programmes, enabling economies of scale on certification artefacts.[13, 51]

3.6 Verification Embedded in the Workflow

Testing and certification are integrated into the development loop, mirroring proposals for continuous certification in avionics and other safety-critical domains.[14]

- Static analysis and deterministic checks run as part of each build, similar to how modern DO-178C toolchains integrate coding rules, flow analysis, and data-coupling checks.[20, 19]
- SITL (Software-in-the-Loop) and HITL (Hardware-in-the-Loop) testing are built into the toolchain, reflecting widely adopted practice in control, automotive, and aerospace V&V.[28, 30, 29]
- Artefacts for DO-178C/DO-254 mapping are produced automatically from existing build and run data, aligning with model-based and tool-centric approaches to regulatory documentation.[13, 27]

Investor anchor: every commit moves the system closer to certification, rather than accumulating debt that must be paid down later in a dedicated certification phase.[14, 26]

4 Software Stack Details

4.1 Firmware & HAL Layer

Responsibilities

- Provide a deterministic task execution model compatible with schedulability analysis for high-criticality workloads.[9]
- Implement a fixed-priority or table-driven scheduler (implementation details withheld) similar to those used in certified RTOSes and cyclic executives.[10, 11]
- Enforce controlled memory regions and safety partitions, following partitioning principles found in ARINC 653 and IEC 61508.[12, 6]
- Provide pre-validated, controlled driver interfaces for I/O and peripherals.
- Implement fault isolation domains and watchdog mechanisms to support fail-safe and fail-operational strategies.[6]
- Offer logging primitives and hooks for runtime instrumentation without violating determinism guarantees.

Visible Concepts (External)

- Philosophy of determinism and safety, grounded in real-time systems literature and functional safety standards.[9, 10, 6]
- Hardware/software "cocoon" model: applications live inside a controlled runtime that mediates access to time, memory, and devices, similar to separation kernel approaches.
- Strict separation between mission logic and low-level device management, echoing the layering found in DO-178C-compliant avionics architectures.[1, 31]

Hidden Details (Proprietary)

- Exact scheduling algorithms and policies.
- Memory layout, partitioning, and isolation mechanisms.
- Firmware implementation details and driver code.
- Interrupt handling strategies and low-level state machines.

4.2 Arcturus SDK

The SDK sits above the HAL and shapes how developers interact with the platform. It is informed by both real-time programming patterns and high-level synthesis frameworks for FPGAs, which demonstrate how carefully constrained abstractions can deliver high performance while remaining analysable.[38, 39, 41]

Components

- **Mission Framework:** declarative configuration for mission logic, tasks, and interactions, making the intended behaviour explicit and traceable.[23, 24]
- **Safe APIs:** deterministic math libraries, timing primitives, messaging, and I/O interfaces designed to avoid non-deterministic features (for example, unbounded queues or uncontrolled blocking).
- **Toolchain:** reproducible builds, static linking, version pinning, and deterministic linker layouts, consistent with configuration management recommendations in DO-178C and DO-254.[1, 2, 21]
- **Developer Constraints:** enforcement of coding patterns that are compatible with deterministic, certifiable execution, analogous to the restricted subsets used in high-criticality profiles of C, Ada, and MISRA-guided environments.[20, 8]
- **Artefact Tags:** automatic embedding of requirements and test identifiers into binaries and logs, enabling straightforward generation of traceability matrices.[25]

Developer Experience

From a developer's perspective:

1. Purchase off-the-shelf hardware from Kesslr, or a custom hardware option with our processor.
2. Write mission code using the Arcturus SDK.
3. Build with the Arcturus toolchain (including static analysis and constraint checks).
4. Run SITL/HITL and retrieve automatically generated logs and reports.
5. Export certification-ready artefacts when needed.

The SDK transforms complex certification constraints into a constrained but productive programming model, similar in spirit to domain-specific accelerator frameworks that hide low-level details while preserving performance and analysability.[36, 42, 44]

4.3 Verification & Evidence Engine

This layer is the core of Arcturus' certification value proposition. It reflects both regulatory expectations for evidence[1, 2, 6] and industrial experience that systematic, tool-driven verification improves both coverage and cost-efficiency.[26, 27]

Components

- **Static Analysis:** checks conformance to deterministic and safety rules, similar to the static verification used in DO-178C toolchains.[20, 19]
- **SITL Test Runner:** executes mission logic in a simulated environment for rapid feedback.[28]
- **HITL Test Runner:** executes on the physical Arcturus board, capturing real timing and coverage data and aligning with HIL practices in automotive and aerospace.[30, 29]

- **Runtime Instrumentation:** controlled hooks into execution for logging and coverage without breaking determinism guarantees.
- **Log Collector and Coverage Mapper:** transforms raw logs into structured coverage and behavioural reports, supporting MC/DC-style coverage analysis where required.[22, 1]
- **Traceability Engine:** builds trace matrices linking requirements, code, tests, and results.[23, 25]
- **Artefact Generator:** packages evidence into DO-178C/DO-254-aligned bundles and can be extended to ECSS, ISO 26262, and IEC 61508 contexts.[5, 8, 6]

Outputs

- Requirements-to-code and requirements-to-test traceability matrices.
- Test execution reports and pass/fail summaries.
- Structural coverage summaries (statement, decision, and where needed MC/DC).[22, 19]
- Timing analysis summaries at a task or function level.
- Configuration management information, including build provenance.
- Exportable evidence bundles for certification authorities and auditors.

The exact internal algorithms and data schemas used by this engine remain proprietary, but are designed to align with typical certification data packages observed in regulated projects.[21, 13]

5 Workflow: Certify-as-You-Build

The Arcturus development workflow is built around continuous evidence creation. This mirrors academic and industrial proposals for continuous or incremental certification, where development and assurance activities are interleaved rather than strictly sequential.[14, 15]

Step 1 – Mission Development

Developers implement mission logic using the Arcturus SDK and safe APIs. Design and requirements can be expressed in a way that is natively traceable (for example, annotations and configuration structures), supporting the creation of traceability artefacts without manual bookkeeping.[23, 24]

Step 2 – Deterministic Build & Static Checks

The build system:

- Runs static analysis and deterministic rule checks.[20]
- Validates constraints such as banned language features or constructs, echoing the restricted subsets used in safety profiles of C, C++, and Ada.[8, 1]
- Produces reproducible binaries with embedded metadata, including build identifiers and versions, references to requirements and test entities, and configuration parameters.

Step 3 – SITL Execution

The mission is executed in a software-in-the-loop environment:

- Rapid iteration on logic without hardware dependencies, following the MIL/SIL/HIL progression familiar from control and automotive toolchains.[28]
- Injection of fault scenarios and edge cases.
- Generation of early logs, coverage estimates, and behavioural traces.

Step 4 – HITL Execution

Once SITL is stable, the same mission is deployed to the Arcturus board.

- Real hardware execution under deterministic conditions.
- Collection of timing, coverage, and fault-handling data.
- Validation of integration-level behaviours, similar to hardware-assisted verification flows used in complex SoC development.[48, 52, 51]

Step 5 – Evidence Generation

The verification engine aggregates build, SITL, and HITL data.

- Automatic construction of trace matrices.[25]
- Structural coverage reports aligned with certification expectations.[1, 22]
- Summary documents, logs, and configuration manifests suitable for inclusion in certification data packs.[21, 27]

Step 6 – Certification Bundle Export

Finally, the system exports a certification bundle with:

- traceability evidence,
- coverage and test reports,
- configuration and build provenance,
- structured documentation that can feed into a DO-178C/DO-254 process and be adapted to ECSS, ISO 26262, and IEC 61508.[1, 2, 5, 8, 6]

Investor anchor: this pipeline compresses months of manual document creation into automated, repeatable operations, as advocated by recent work on digital certification and model-based regulatory documentation.[13, 14]

6 Mapping to Certification Standards

Arcturus is intentionally aligned with major safety-critical standards. Rather than reinterpreting each standard from scratch, the architecture is designed to sit on top of the established objectives for software, hardware, and functional safety.[1, 2, 5, 8, 6, 33]

6.1 DO-178C (Software)

Objective Area	Arcturus Coverage (Conceptual)
Levels A/B/C	Deterministic execution, structured testing (SITL/HITL), and traceable mission logic support high criticality levels, reflecting DO-178C expectations for high Design Assurance Levels.[1, 32]
Structural Coverage	Built-in runtime instrumentation and test harnesses generate coverage data compatible with required evidence levels (up to MC/DC for DAL A).[1, 22, 20]
Verification Independence	Automated, tool-driven workflows can be configured to align with independence requirements (for example, separation of roles and environments), as described in FAA and EASA guidance.[32, 31]
Requirements Linking	Requirements identifiers are embedded in code, configurations, and tests, enabling automated traceability, consistent with DO-178C and common industry practice.[23, 25]
Configuration Management	Reproducible builds and artefact tagging provide traceability from binary back to source, configuration, and requirements, satisfying CM objectives.[21, 6]

6.2 DO-254 (Hardware)

While DO-254 focuses on hardware, Arcturus supports hardware certification by:

- fixing and documenting hardware–software boundaries, which DO-254 and DO-178C both treat as critical for clear responsibility and evidence partitioning,[2, 1]
- providing reproducible firmware and HAL behaviour that can be treated as a controlled hardware-dependent layer,[4, 12]
- generating artefacts that link hardware configurations to software tests and outcomes, aiding DO-254 verification and validation activities,[13]
- allowing consistent reuse of hardware behaviours across missions and programmes via controlled platform configurations.[35, 36]

6.3 Other Standards (ECSS, ISO 26262, IEC 61508)

The principles embedded in Arcturus (determinism, partitioning, controlled interfaces, and strong traceability) are compatible with:

- ECSS for space products, which mirrors many DO-178C objectives in a space context,[5, 33]

- ISO 26262 for automotive functional safety, which requires traceable safety requirements, analysed architectures, and evidence for ASIL-rated software and hardware,[8]
- IEC 61508 as a generic functional safety framework underpinning many sector-specific standards and emphasising systematic capabilities and lifecycle-based assurance.[6]

Investor anchor: Arcturus is a horizontal certification engine, not limited to a single niche standard. The same architectural principles and tooling apply across multiple regulated domains.[13, 27]

7 Reliability & Safety Model

7.1 Fault Tolerance Philosophy

Conceptually, Arcturus is designed to support:

- **Fault domains** that isolate failures in specific components, following fault containment region concepts in real-time and safety literature,[10, 6]
- watchdogs and safe-state fallbacks for mission-critical functions,
- deterministic rollback or recovery procedures where feasible,
- clean reporting of faults through instrumentation and logging, enabling effective diagnosis and safety case argumentation.

7.2 Safety Enforcements

The platform enforces safety via:

- exclusion of non-deterministic features (for example, unconstrained threads and dynamic memory), in line with recommendations for high-criticality software in DO-178C, ISO 26262, and IEC 61508,[1, 8, 6]
- bounded resource usage to avoid overload conditions, which is a prerequisite for meaningful worst-case analysis,[9]
- restricted access to time and randomness sources,
- controlled communication patterns between subsystems, supporting arguments about freedom from interference and information flow.[11, 12]

7.3 Observability

Observability is built into the runtime and verification layers.

- Per-task or per-function timing summaries.
- Structured logs suitable for both debugging and evidence generation, aligned with best practices reported in industrial surveys.[26, 27]
- Execution signatures that allow detection of unexpected behaviour, including subtle timing or sequencing deviations that may impact safety or mission objectives.

8 Roadmap: Hardware, Software, & Certification

8.1 Phase 1: FPGA Prototype (Now–2026)

- Completing and hardening the deterministic runtime and HAL.
- Maturing the SDK and developer experience.
- Stabilising the verification engine and artefact generator.
- Delivering early pilots with university and startup teams.
- Integrating with certification consultants and auditors for real-world feedback.[13, 27]

8.2 Phase 2: Arcturus SoC V1 (2026–2027)

- Transition from FPGA prototypes to a custom SoC with:
  - real-time cores tuned for deterministic workloads,[9, 10]
  - hardware features that accelerate timing analysis and trace collection, inspired by on-chip monitoring and trace modules in modern SoCs and FPGA platforms,[48, 35]
  - on-chip support for error detection and partitioning.
- Extending the verification toolchain to exploit new hardware primitives.
- Building long-term partnerships with aerospace, defence, and autonomy customers.

8.3 Phase 3: Platform Expansion (2027+)

- Introducing a safe domain-specific language (DSL) for mission configuration and logic, leveraging experience from DSLs and constrained languages used in safety-critical systems.
- Building a reusable artefact repository, enabling cross-mission evidence reuse, analogous to template-based approaches to regulatory documentation.[13]
- Expanding into deterministic AI kernels that remain certifiable and bounded, drawing on the rapidly growing body of work in FPGA-based neural network accelerators and their energy/performance benefits.[42, 43, 44, 46, 47]
- Developing an enterprise certification portal for programme-level governance.

Investor anchor: Arcturus is not a single dev board. It is a multi-phase platform aimed at becoming the default infrastructure for safety-critical compute, with a roadmap that closely follows proven paths in accelerators and safety tooling.[34, 36]

9 Evidence of Execution

Without disclosing proprietary details, the following milestones will demonstrate execution, aligned with best practices observed in industrial surveys and vendor guidance for safety-critical software and hardware.[26, 27, 51]

- Operational firmware layer on FPGA prototypes.
- Deterministic runtime and task scheduling skeleton implemented.
- Early version of the SDK with safe APIs available for pilot users.
- Reproducible build system integrated with traceability metadata.
- Functional SITL and HITL harnesses for early adopters.[28, 30]
- Prototype artefact generator producing trace matrices and basic coverage reports.
- Live mission demos with early university and startup teams.
- Significant early interest: over 100 teams in the testing pipeline, with multiple EOIs and MOUs in progress.

10 Summary for Investors

Arcturus delivers:

- A deterministic compute platform intentionally designed for certification, grounded in real-time and functional safety theory.[9, 10, 6]
- An SDK that prevents nondeterminism and embeds verification by construction.[20, 19]
- A verification engine that automates the bulk of certification artefact creation, in line with emerging continuous-certification approaches.[14, 13]
- A clear roadmap from FPGA to custom SoC and beyond, consistent with the evolution of FPGA-based platforms into full heterogeneous computing systems.[34, 35]
- A horizontal platform that can address multiple standards and domains.[1, 2, 5, 8, 6]

Moat and Defensibility

- Tight hardware–software coupling around determinism and certification, analogous to domain-specific accelerator stacks where hardware and software co-evolve.[36, 42]
- Proprietary deterministic runtime and verification strategies.
- A growing corpus of reusable certification artefacts and patterns.
- Relationships with teams and programmes for whom certification is the primary bottleneck.

Thesis: Arcturus changes the unit economics of building safety-critical systems by making certification a continuous, automated outcome of normal development, rather than a separate multi-year effort.[14, 13]

For further information or extended technical appendices, please contact the Kesslr Labs team.

References

[1] RTCA. *DO-178C: Software Considerations in Airborne Systems and Equipment Certification*. RTCA, 2011.
Available at https://www.rtca.org/do-178/.

[2] RTCA. *DO-254: Design Assurance Guidance for Airborne Electronic Hardware*. RTCA, 2000.
Product page: https://my.rtca.org/productdetails?id=a1B36000001IcjTEAS.

[3] FAA. AC 20-152 / 20-152A – *RTCA, Inc., Document RTCA/DO-254, Design Assurance Guidance for Airborne Electronic Hardware*.
Information at https://www.faa.gov/regulations_policies/advisory_circulars/index.cfm/go/document.information/documentid/22211.

[4] Cadence. *DO-254 Explained*. Whitepaper.
Available at https://www.cadence.com/content/dam/cadence-www/global/en_US/documents/solutions/aerospace-and-defense/do-254-explained-wp.pdf.

[5] ECSS. *ECSS-E-ST-40C Rev.1: Software*. 30 April 2025.
Overview: https://ecss.nl/standard/ecss-e-st-40c-rev-1-software-30-april-2025/.
PDF:
https://ecss.nl/wp-content/uploads/2025/05/ECSS-E-ST-40C-Rev.1%2830April2025%29.pdf.

[6] IEC. *Functional Safety and IEC 61508*. IEC overview.
Available at https://www.iec.ch/functional-safety.

[7] IEC. *Overview of IEC 61508 & Functional Safety*.
PDF at
https://assets.iec.ch/public/acos/IEC%2061508%20%26%20Functional%20Safety-2022.pdf.

[8] International Organization for Standardization.
ISO 26262: Road Vehicles – Functional Safety. ISO, 2018.
Overview: https://www.iso.org/standard/68383.html.

[9] Buttazzo, G. C. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*. Springer.
Springer page: https://link.springer.com/book/10.1007/978-3-031-45410-3.

[10] Kopetz, H., and Steiner, W. *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Springer.
Springer page: https://link.springer.com/book/10.1007/978-3-031-11992-7.

[11] Wind River. *ARINC 653 Compliant Safety-Critical Applications* – concept overview.
Available at https://www.windriver.com/solutions/learning/arinc-653-compliant-safety-critical-applications.

[12] Cheptsov, V., Khoroshilov, A. *Robust Resource Partitioning Approach for ARINC 653 RTOS*. 2023.
arXiv: https://arxiv.org/pdf/2312.01436.
Semantic Scholar: https://www.semanticscholar.org/paper/Robust-Resource-Partitioning-Approach-for-ARINC-653-Cheptsov-Khoroshilov/0cfc72a31728ae7e0ab98cdb03e4599f20d1cf75.

[13] Cartile, A., Marsden, C., Liscouët-Hanke, S. *Digital Transformation in Aircraft Design and Certification: Developing Requirements for Modeling Regulatory Documentation*.
Aerospace, 12(8), 724, 2025.
MDPI: https://www.mdpi.com/2226-4310/12/8/724.

[14] Baron, C., Louis, V. *Towards a Continuous Certification of Safety-Critical Avionics Software.* *Computers in Industry*, 125, 103382, 2021.
ScienceDirect: https://www.sciencedirect.com/science/article/pii/S0166361520306163.

[15] Zakrzewski, P., et al.
Safety-Critical Software Development Methodologies in Avionics. Transactions on Aerospace Research, 2020.
Overview: https://sciendo.com/article/10.2478/tar-2020-0004.

[16] Hilderman, V. *Understanding DO-178C Software Certification: Costs vs. Benefits.*
Semantic Scholar entry: https://www.semanticscholar.org/paper/Understanding-DO-178C-Software-Certification%3A-Costs-Hilderman/49fc1d3a2e2315385ee412c5a1792a3be0c64bef.

[17] AFuzion. *DO-178C Costs Versus Benefits.* Whitepaper.
Available at https://afuzion.com/do-178c-costs-versus-benefits/.
UAV-specific variant: https://afuzion.com/uavs-applying-178c-costs-vs-benefits/.

[18] ConsuNova. *DO-178C Explained.* 2024.
Available at https://consunova.com/do-178c-explained/.

[19] Parasoft. *DO-178C Software Compliance for Aerospace and Defense.* Online guide.
Overview page: https://www.parasoft.com/learning-center/do-178c/overview/.

[20] AdaCore. *Compliance with DO-178C/ED-12C Guidance: Analysis.* In *AdaCore Technologies for Airborne Software.*
Online chapter: https://learn.adacore.com/booklets/adacore-technologies-for-airborne-software/chapters/analysis.html.

[21] AdaCore. *The DO-178C/ED-12C Standards Suite.*
Online chapter: https://learn.adacore.com/booklets/adacore-technologies-for-airborne-software/chapters/standards.html.

[22] Rapita Systems. *An Introduction to Modified Condition/Decision Coverage (MC/DC).* 2010.
Available at https://www.rapitasystems.com/blog/introduction-modified-conditiondecision-coverage-mcdc.

[23] Peraldi-Frati, M.-A., and Albinet, A. *Requirement Traceability in Safety Critical Systems.* CARS 2010.
ResearchGate: https://www.researchgate.net/publication/221246521_Requirement_traceability_in_safety_critical_systems.

[24] Ramesh, B., Jarke, M. *Toward Reference Models for Requirements Traceability. IEEE Transactions on Software Engineering*, 27(1):58–93, 2001.
Overview via Cleland-Huang et al.:
https://link.springer.com/chapter/10.1007/978-3-642-28714-5_16.

[25] Pachiana, G., Grunwald, M., Markwirth, C., Sohrmann, C. *Automated Traceability of Requirements in the Design and Verification Process of Safety-Critical Mixed-Signal Systems.* DVCon Europe, 2021.
Proceedings page:
https://dvcon-proceedings.org/document/automated-traceability-of-requirements-in-the-design-and-verification-process-of-safety-critical-mixed-signal-systems/.
Direct PDF: https://dvcon-proceedings.org/wp-content/uploads/automated-traceability-of-requirements-in-the-design-and-verification-process-of-safety-critical-mixed-signal-systems.pdf.

[26] Kassab, M. *Testing Practices of Software in Safety Critical Systems: Industrial Survey*. ICEIS 2018. SCITEPRESS PDF: https://www.scitepress.org/papers/2018/67970/67970.pdf.

[27] National Instruments. *Best Practices for Embedded Software Testing of Safety Compliant Systems*. 2025.
Available at https://www.ni.com/en/solutions/transportation/best-practices-for-embedded-software-testing-of-safety-compliant.html.

[28] MathWorks. *What Is Hardware-in-the-Loop (HIL)?*. Discovery page.
Available at https://au.mathworks.com/discovery/hardware-in-the-loop-hil.html.

[29] Ansys. *What is Hardware-in-the-Loop Testing?*. 2025.
Available at
https://www.ansys.com/simulation-topics/what-is-hardware-in-the-loop-testing.

[30] National Instruments. *What Is Hardware-in-the-Loop (HIL)?*. 2024.
Available at https://www.ni.com/en/solutions/transportation/hardware-in-the-loop/what-is-hardware-in-the-loop-.html.

[31] Civil Aviation Safety Authority (CASA). AC 21-50 v1.0 – *Approval of Software and Electronic Hardware Parts*.
Info at https://www.casa.gov.au/approval-software-and-electronic-hardware-parts.

[32] FAA. AC 00-69 – *Best Practices for Airborne Software Development Assurance Using EUROCAE ED-12( ) and RTCA DO-178( )*.
Linked from https://en.wikipedia.org/wiki/AC_00-69.

[33] SYSGO. *Space DO-178C vs. ECSS-E-ST-40C*. 2024.
Available at https://www.sysgo.com/blog/article/space-do-178c-vs-ecss-e-st-40c.

[34] Hartenstein, R. "A Decade of Reconfigurable Computing: A Visionary Retrospective." In *Proceedings of Design, Automation and Test in Europe (DATE)*, 2001.
PDF: https://websrv.cecs.uci.edu/~papers/compendium94-03/papers/2001/date01/pdffiles/09a_1.pdf.

[35] Trimberger, S. "Three Ages of FPGAs: A Retrospective on the First Thirty Years of FPGA Technology." *Proceedings of the IEEE*, vol. 103, no. 3, 2015.
Info: https://www.semanticscholar.org/paper/Three-Ages-of-FPGAs%3A-A-Retrospective-on-the-First-Trimberger/e4b8b984927c197f4fc0d876d0015b6d4b2b9302.

[36] Peccerillo, B., Prinetto, P., Trotta, P. "A Survey on Hardware Accelerators: Taxonomy, Trends, and Challenges." *Journal of Systems Architecture*, 125, 102453, 2022.
ScienceDirect: https://www.sciencedirect.com/science/article/pii/S1383762122001138.

[37] Tibaldi, M., Pilato, C. "A Survey of FPGA Optimization Methods for Data Center Energy Efficiency." *IEEE Transactions on Sustainable Computing*, 2023 (early access).
arXiv PDF: https://arxiv.org/pdf/2309.12884.

[38] Nane, R., Sima, V., Pilato, C., et al. "A Survey and Evaluation of FPGA High-Level Synthesis Tools." *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 35(10):1591–1604, 2016.
PDF: https://janders.eecg.utoronto.ca/pdfs/tcad_hls.pdf.

[39] Huang, L., Cong, J., Xu, Z., et al. "A Survey on Performance Optimization of High-Level Synthesis for FPGAs." *Journal of Computer Science and Technology*, 35, 2020.
DOI: https://doi.org/10.1007/s11390-020-9414-8.

[40] Lahti, S. "High-Level Synthesis for FPGAs – A Hardware Engineer's Perspective." Master's Thesis, Tampere University, 2025.
PDF: https://trepo.tuni.fi/bitstream/handle/10024/212180/High-Level_Synthesis_for_FPGAsA_Hardware_Engineers_Perspective.pdf.

[41] Molina, R., et al. "High-Level Synthesis Hardware Design for FPGA-Based Systems." *Microprocessors and Microsystems*, 2022.
PDF: https://ieeexplore.ieee.org/document/9864576.

[42] Wu, R., et al. "Accelerating Neural Network Inference on FPGA-Based Platforms: A Survey." *Electronics*, 10(9):1025, 2021.
MDPI: https://www.mdpi.com/2079-9292/10/9/1025.

[43] Liu, F., et al. "Review of Neural Network Model Acceleration Techniques Based on FPGA." *Neurocomputing*, 2024.
ScienceDirect: https://www.sciencedirect.com/science/article/abs/pii/S0925231224012827.

[44] Chowdhury, M. R. Z., et al. "Towards Next-Generation FPGA-Accelerated Vision-Based AI Systems." *AI*, 6(4):53, 2025.
MDPI: https://www.mdpi.com/2624-6120/6/4/53.

[45] Siddiqui, F., et al. "FPGA-Based Processor Acceleration for Image Processing Applications." *Electronics*, 8(8):910, 2019.
PMC: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8320866/.

[46] Li, X., et al. "A Survey of FPGA Based CNNs Accelerators." Preprint, 2020.
EasyChair: https://easychair.org/publications/preprint/GGnJ.

[47] Zhang, C., et al. "FPGA-based Acceleration for Convolutional Neural Networks: A Comprehensive Review." Preprint, 2025.
arXiv: https://arxiv.org/abs/2505.13461.

[48] Synopsys. "FPGA-Based Prototyping: HAPS Solution Datasheet."
PDF: https://www.synopsys.com/content/dam/synopsys/services/datasheets/fpga-based-prototyping-ds.pdf.

[49] Synopsys. "What is HAV Prototyping? – Hardware-Assisted Verification Glossary Entry." 2025.
https://www.synopsys.com/glossary/what-is-hav-prototyping.html.

[50] EETimes / Synopsys. "The Case for Hardware-Assisted Verification in Complex SoCs." 2025.
https://www.eetimes.com/synopsys-speeds-verification-with-hardware-assisted-systems/.

[51] S2C Inc. "Prototypical: The Practice of FPGA-Based Prototyping for SoC Design." Ebook, 2022.
PDF: https://www.s2cinc.com/resources/lit/en/ebooks/s2c-prototypical-the-practice-of-fpga-based-prototyping-for-soc-design.pdf.

[52] FirstEDA. "Hardware Assisted Verification (HW-V)."
https://firsteda.com/solutions/hw-v/.

[53] Th'ea–Martine Gauthier and the HPSC Core Technical Team, *High Performance Spaceflight Computing (HPSC) White Paper*, NASA / Jet Propulsion Laboratory, June 2024.
https://www.nasa.gov/wp-content/uploads/2024/07/hpsc-white-paper-tmg-26jun2024-final.pdf.