Certification-First Compute Infrastructure for Spaceflight Systems

A Constrained SDK, Deterministic Hardware Platform, and Integrated Test Stack

Kronus
Version 0.1

Technical Abstract

Spaceflight systems impose uniquely stringent requirements on computational infrastructure. Launch vehicles, spacecraft, and orbital platforms operate in environments where failure is catastrophic, intervention is impossible, and system behaviour must remain predictable across long mission lifetimes. At the same time, the coming decade of space activity will be characterised by increasing onboard autonomy and inference, higher launch cadence, and more software-defined spacecraft.

Despite substantial advances in computational capability, the development of flight-critical electronic systems remains bottlenecked not by physics or algorithms, but by the difficulty of producing convincing safety assurance. In practice, this has driven teams to rely on conservative, inefficient, and often outdated technical solutions whose primary value lies in their accumulated flight heritage rather than their intrinsic suitability for modern missions.

Kronus is developing a vertically integrated compute platform purpose-built for spaceflight. The platform comprises a constrained software development kit (SDK), a deterministic hardware execution target, and tightly coupled simulation, testing, documentation, and evidence-generation tools. Together, these components are designed to demonstrate safety properties more directly, more transparently, and with less incidental complexity than existing approaches.

Certification is treated not as a post-hoc activity, but as an emergent property of system architecture and workflow. By improving how safety assurance is demonstrated, the platform enables both higher efficiency and broader participation, while remaining aligned with regulator expectations. While designed first for space systems, the underlying principles extend naturally to other safety-critical domains.

1. Problem Definition: Spaceflight Development Is Constrained by Assurance

1.1 Nature of Spaceflight Systems

Spaceflight systems are characterised by tight coupling between:

- Temporal behaviour (timing, deadlines, jitter)
- Resource usage (compute, memory, power, and I/O bandwidth)
- Execution order and state transitions
- Environmental constraints (radiation effects, thermal variation, vacuum)

Correctness in space systems is not solely functional. It is behavioural, temporal, and environmental. Small deviations in timing, resource contention, or fault-handling behaviour can result in mission-ending failures even when functional logic is correct.

1.2 Heritage-Driven Development Reality

To manage these risks, spaceflight teams have historically prioritised designs with extensive flight heritage. In many cases, this has led to the continued use of computational architectures, hardware platforms, and development practices that are conservative, inefficient, or poorly suited to modern mission requirements, but whose behaviour is well understood and defensible during certification.

The constraint is not a lack of better technical solutions, but the difficulty of convincingly demonstrating safety and predictability for newer, more efficient approaches. As a result, development effort is often spent working around tooling and architectural limitations rather than improving mission capability.

1.3 Consequences

This assurance-driven conservatism leads to:

- Slower development and iteration cycles
- Reduced onboard capability per unit mass, power, or cost
- High dependence on a small pool of experienced personnel
- Barriers to entry for new organisations and emerging spacefaring nations

Much of this complexity is incidental, arising from how safety is demonstrated rather than from unavoidable constraints imposed by spaceflight physics.

2. SDK as the Primary System Abstraction

2.1 Design Objective

The Kronus SDK is designed to encode spaceflight and safety-critical constraints directly into the software development model. Rather than relying on informal process, institutional

knowledge, or conservative overdesign, the SDK provides explicit constructs for expressing timing, execution structure, and resource usage.

This approach reflects the needs of modern space systems, which increasingly depend on software-defined functionality, autonomy, and adaptive behaviour that cannot be supported efficiently by legacy development models.

2.2 What the SDK Abstracts

The SDK abstracts away sources of incidental complexity that are difficult to analyse and defend during certification, including:

- Ad-hoc scheduling mechanisms
- Hardware-specific arbitration details
- Unbounded concurrency patterns
- Informal timing management

By doing so, it allows teams to focus on what the system must do, rather than on how to defensively constrain general-purpose tooling.

2.3 What the SDK Makes Explicit

In place of implicit behaviour, the SDK requires explicit declaration of:

- Execution boundaries and task lifecycles
- Data-flow relationships
- State transitions
- Temporal constraints and execution rates

Only behaviours that can be bounded, analysed, and demonstrated are expressible within the SDK.

2.4 AI Integration

Because the SDK makes unsafe behaviour inexpressible, forces explicit structure, and guarantees analysability, AI agents can safely operate within SDK constraints. These agents implement, fix, and optimise components from structured plans, and their output remains statically analysable + evidence-generating

2.5 Resulting Properties

Software developed using the SDK is:

- Statically analysable by default
- Predictable under defined load and fault conditions
- Structurally constrained against unsafe interaction patterns

This reduces reliance on conservative assumptions and enables more capable and efficient designs without sacrificing assurance.

3. Capability Expansion Through Demonstrable Assurance

3.1 Assurance-Limited Development Model

In conventional spaceflight development, safety assurance is achieved primarily through heritage, conservative margins, and manual reasoning. This limits who can build flight systems and encourages reuse of known solutions even when they are suboptimal.

3.2 Demonstration-First Model

Kronus adopts a demonstration-first approach:

- Safety-relevant behaviour is explicit and inspectable
- Invalid or unprovable designs are unrepresentable
- Assurance is derived from system structure rather than accumulated history

Constraint is used not to limit capability, but to make safety properties easier to demonstrate.

3.3 Practical Impact

This approach enables:

- More efficient use of compute, power, and mass while maintaining safety
- Faster iteration during development and ground testing
- Participation by teams without decades of accumulated flight heritage

Capability is expanded by improving how safety is shown, allowing newer and better technical solutions to be adopted with confidence.

4. Deterministic Hardware Execution Platform

4.1 Motivation

Many modern hardware platforms prioritise flexibility and peak performance, but do not make safety-relevant behaviour easy to bound or explain. In spaceflight contexts, this pushes teams toward conservative configurations or older platforms with well-understood behaviour.

4.2 Hardware Design Principles

The Kronus hardware platform is designed to make safety properties directly observable and defensible. Core principles include:

- Explicit and bounded execution timing
- Clear resource partitioning
- Elimination of implicit or hidden execution behaviour

The objective is not novelty, but to provide a hardware substrate whose behaviour can be demonstrated clearly and efficiently.

4.3 Co-Design with the SDK

The hardware platform enforces the assumptions made by the SDK, while the SDK exposes the guarantees provided by the hardware. This alignment reduces the need for conservative overdesign and narrows the gap between intended and observed behaviour across simulation, testing, and flight.

5. Integrated Simulation, Testing, Documentation, and Evidence Generation

5.1 Continuous Validation Model

Simulation, testing, and documentation are embedded into the development lifecycle rather than treated as downstream activities. The platform supports:

- Software-in-the-loop execution
- Timing- and behaviour-accurate simulated hardware
- Hardware-in-the-loop testing
- Deterministic replay of execution scenarios

This enables early validation of flight behaviour and assurance arguments before physical hardware is finalised.

5.2 Automatically Generated Artefacts

As systems are developed, simulated, and tested, the platform automatically generates:

- Execution traces and timing envelopes
- Configuration and build records
- Change impact summaries
- Supporting technical documentation

These artefacts are reproducible, comparable across iterations, and suitable for independent review.

5.3 Review, Transparency, and Local Capability

Because evidence and documentation are generated continuously, system behaviour can be reviewed at any stage by internal teams, regulators, or government stakeholders. Building these capabilities domestically strengthens supply-chain resilience, reduces dependence on external vendors, and enables closer collaboration with regulators on how safety is demonstrated.

6. Certification as an Emergent and Transparent Property

6.1 Traditional Spaceflight Certification

Spaceflight certification has historically relied on manual evidence reconstruction, human-maintained traceability, and concentrated review phases late in the program lifecycle.

6.2 Certification-First Architecture

Within the Kronus platform:

- Evidence exists because the system executes under representative and simulated constraints
- Assurance artefacts are generated as part of normal development workflows
- Reviews can occur incrementally throughout the lifecycle

Certification is supported by system structure rather than imposed retroactively.

6.3 Public and Repeatable Assurance Frameworks

By producing evidence through a consistent and publishable framework, the platform supports clearer communication between developers and regulators and reduces reliance on opaque, organisation-specific processes.

6.4 Implications for Sovereign Capability

By improving how safety is demonstrated rather than relying on inherited heritage, the platform supports the development of efficient, modern, and sovereign spaceflight capability. Teams can adopt better technical solutions while maintaining confidence that safety expectations are met.

7. Scope, Boundaries, and Non-Goals

The Kronus platform is not:

- A general-purpose operating system
- A replacement for human judgement
- A universal solution for legacy systems

It is best suited for:

- New space system developments
- Incrementally introduced flight components
- Teams seeking to improve efficiency while maintaining high assurance

8. Conclusion

Spaceflight development has long traded efficiency and capability for demonstrable safety through heritage. As space activity accelerates, this trade-off becomes increasingly limiting.

By improving how safety properties are expressed, demonstrated, and reviewed, Kronus enables more efficient hardware and software to be adopted without compromising assurance. Certification becomes a natural outcome of system behaviour rather than a justification exercise, supporting faster development, higher capability, and the growth of resilient sovereign spaceflight programs.