Core Avionics & Industrial Inc.
400 North Tampa Street
Suite 2850
Tampa, Florida 33602

T: 888-330-5376
F: 866-485-3199
www.coreavi.com

# Principles of Safety Critical Software Design:

# Applying Avionics Safety Design to Automotive and Industrial Markets

## Abstract

This white paper will provide the history and current state of the art in safety critical design for commercial and military avionics display systems. These principles will then be extrapolated to address evolving safety considerations in markets such as autonomous vehicles, commercial UAVs and cloud edge autonomous machines.

## Principles of Safety Critical Design

To understand the principles of safety critical software design, it is helpful to examine a typical software stack in embedded safety critical designs common to avionics and aerospace applications. The application layer, typically tied to a user interface or HMI, is generally a complex software layer that executes a set function or responds to a set of inputs. The application programs are generally called through a real time operating system (RTOS) that itself is designed to guarantee latency and provide deterministic operation. The RTOS and the application program are usually interfaced to a hardware driver library that is constructed to a safety critical design standard such as OpenGL® SC2. The driver layer is standardized to assure latency and determinism through the construction of application programming interfaces (APIs) that effectively confine the functionality of application program calls. The design of the APIs, using safety critical constraints, sets the foundation for higher level software and serves as a system design constraint set to ensure safety critical operation. Without a set of safety critical APIs, the system design challenge is multiplied, and system verification is made more complex and risky.

In concert with safety critical standard driver libraries, the underlying hardware is often designed to allow for monitoring and checking of command execution. These monitor activities form the feedback system to the application for error detection and notification. The system application software can take prescribed action based on the nature and severity of the error condition. These actions span from simply reporting the fault to system reconfiguration or shutdown. The interaction of these software layers and the need for safety constraints and functional error reporting are depicted in Figure 1 below.

Core Avionics & Industrial Inc.
400 North Tampa Street
Suite 2850
Tampa, Florida 33602

T: 888-330-5376
F: 866-485-3199
www.coreavi.com

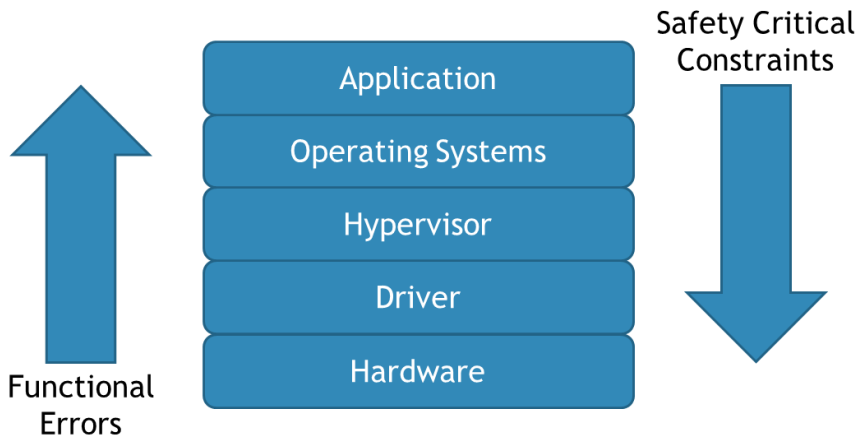


*Figure 1: Safety Critical Stacking*

The Hypervisor layer is a resource management layer that allows for prioritization and management of multiple applications with differing levels of safety critical function. For example, the safety constraints and functional error overhead for an entertainment application is dramatically different than that required by a console display with embedded safety information.

A system design challenge that pervades all safety critical embedded functions is how to maximize functionality and performance within the boundaries of safety critical constraints while also including robust functional error checking and management. Historically, this challenge was resolved through the deployment of individual, independent systems that were constrained to single or limited common functions. The boundaries of these systems were typically hardware boundaries with independent system design flows and goals. Moving forward, system designers endeavor to consolidate more functions onto few, or single, CPU and GPU complexes and to isolate functions through software design. This provides for lower overall cost while allowing the sharing of resources between dissimilar safety level applications.

## Design Constraints

Regardless of approach, safety critical designs must adhere to certain rules of safe software construction. Code must be constructed in a non-blocking fashion, precluding the use of semaphores and mutexes for resource allocation. Second, the use of interrupts must be either excluded or skillfully controlled to ensure execution priority and latency. Third, error conditions must be carefully defined and controlled to prevent unintended system states. A listing of software execution details is shown in Figure 2.

Core Avionics & Industrial Inc.
400 North Tampa Street
Suite 2850
Tampa, Florida 33602

T: 888-330-5376
F: 866-485-3199
www.coreavi.com

**ISO 26262-6: Ten software unit design/implementation principles.**

1. One entry and one exit point in subprograms and functions
2. No dynamic objects or variables, or else online test during their creation
3. Initialization of variables
4. No multiple use of variable names
5. Avoid global variables or else justify their usage
6. Limited use of pointers
7. No implicit type conversions
8. No hidden data flow or control flow
9. No unconditional jumps
10. No recursions

*Figure 2: Software Safety Critical Design Rules*

The more complex the software routine and the greater the number of functional requirements, the more difficult it becomes to adhere to these basic rules of construction. Therefore, applying safety constraints at the proper software interface level allows higher level software to be designed with known safe operation. The specification of software design principles and methods is not enough to assure safe operation. Today's avionics systems are designed to a known safe interface standard and use software that has been proven through multiple system deployments to function in a deterministic manner.

## What is Safe, Very Safe, and Safe Enough?

How safe does software need to be and how should safety be determined? The design of safe software has been practiced for many years in the aerospace industry. The first digital jet engine controller was designed in 1968 while the first fly by wire system debuted on the F-16 in 1978. The first glass cockpit design appeared in 1988 and is standard on all airframes today. This experience, accumulated by the avionics industry over the years, has led to a program management framework that assures every new safety system meets aerospace operational demands. Some of these considerations are as follows:

- Management and communication of risk
- Safety critical systems demand "fit for purpose" considerations
- Standards and certification practices make this easier and demonstrable
- Goal in safety critical implementations:
    - Efficient
    - Effective
    - Risk Reducing
    - Verifiable

Core Avionics & Industrial Inc.
400 North Tampa Street
Suite 2850
Tampa, Florida 33602

T: 888-330-5376
F: 866-485-3199
www.coreavi.com

These principles have created an industry where assurance of safety is a framework rather than a program element. The first, last and primary consideration for these systems is operational safety. But the question remains - what is safe enough? The answer is not simple. What we do know is that the knowledge and methodology accumulated over decades of designing in a "cannot fail" environment has led to commercial jet travel being safer than crossing the street, and this is safe enough.

## New Safety Critical Applications

The reality of automated machines is inevitable. Even today, robotics applications run in fully autonomous modes while more and more work is being focused on adding mobility to autonomous operation. The question is when, not whether, the world will be changed by these machines. A simple set of top level design requirements can be defined to ensure safe system design of these new machines.

- Standards-based
- Verified and proven test cases for system verification
- Coordinated two-way communication between machines and the environment
- Self-contained safety systems that cannot be reliant on external data interfaces
- Established error handling methodology
- Acceptable worst-case failure scenarios

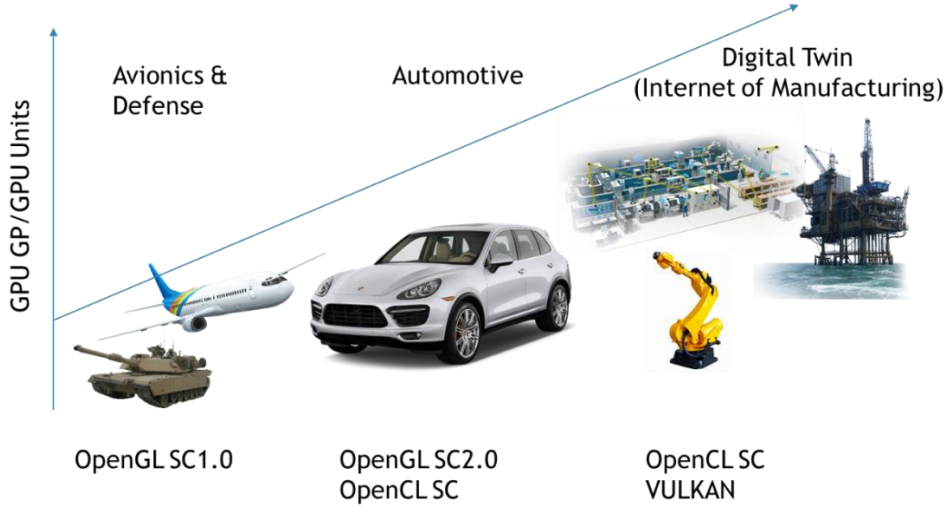


*Figure 3: Evolution of Autonomous Machines and Standards*

The rate we accept new autonomous machines into everyday life will partially be a function of our ability to design them for 100% failsafe operation and prove that these machines will operate safely.

Core Avionics & Industrial Inc.
400 North Tampa Street
Suite 2850
Tampa, Florida 33602

T: 888-330-5376
F: 866-485-3199
www.coreavi.com

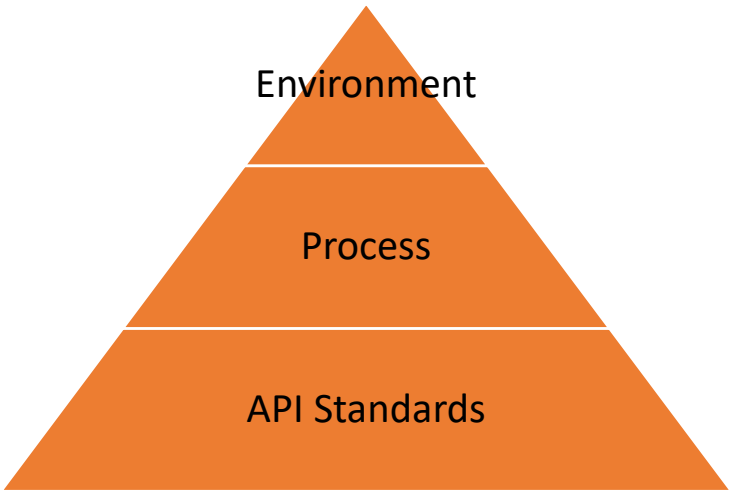


*Figure 4: Safety Critical Software Development Pyramid*

Successful creation of safety critical software is dependent on many factors. Figure 4 illustrates there is a hierarchy of constraint necessary within the development organization. The top of the pyramid represents the culture of a safety critical environment. It is not enough to assign safety critical development on a project by project basis - it must be in the fabric of the development organization and team. In other words, it must be the first priority addressed and the last thing verified. Linked to this factor is the creation of a software development process that guides the development team and assures verification of the product. These process standards, such as DO-178C and ISO 26262 should be the minimum, and therefore only, standards within the software development organization. Lastly, the software product itself must be designed to a safety critical API standard such as Vulkan or OpenGL SC2. The use of API standards provides design constraints and error reporting that ensure higher level, more complex, software will function in a safe manner.

For more information on CoreAVI's ArgusCore OpenGL safety certifiable graphics drivers, visit our website.