Actuator Control Design for Safety-Critical Applications

Vince Socci, Principal
On Target Technology Development, LLC
1701 North Street, Bldg 40-1
Endicott, NY 13760 USA
vsocci@ontargettechnology.com
(607) 755-4990

Abstract - Designers of safety-critical actuator control applications face rigorous requirements and standards to assess safety requirements, develop system architectures, and design component hardware and software. This paper demonstrates integrated techniques of safety-critical development with examples from various actuator applications. Strategies for safety analysis, engineering design and application of lifecycle guidelines are discussed. Methods of developing actuator controls with robust fault tolerance and testability are highlighted. The secrets of effective safety-critical development are revealed.

I. FUNDAMENTAL CONCEPTS OF SAFETY-CRITICAL SYSTEMS

Imagine sitting comfortably on an airplane, enjoying a new issue of your favorite magazine. All of a sudden, as you fly over the Equator, the plane does a fast 180-degree roll, and you find yourself in an inverted flight. The pilot announces over the loudspeaker "Hmmm… that doesn't seem right. Are there any systems engineers onboard?"

This scenario may seem far-fetched, but it's not. The software development for the F-16 fighter plane experienced this exact failure mode. Fortunately, it occurred during simulation flight testing and was resolved in the fielded design. If this failure occurred in real-life, it would have resulted in the loss of aircraft and pilot.

Was this a weird anomaly? Perhaps. But consider other problems discovered during the comprehensive simulation testing of the F-16 [1,2]:

- When the computer was commanded to raise the landing gear while the aircraft was standing still on the runway, the computer complied and "turfed" the aircraft.
- The aircraft system also complied with commands to jettison missiles, bombs, fuel tanks, etc. while the plane was upside-down, resulting in them falling on and damaging the wings.
- When the F-16 went into a spin, the software did not give the pilot enough control authority to recover. The pilot had to eject.

The F-16 problems are not unique. The A320 Airbus has been involved in at least four fatal accidents to date. These are at least in part attributed to software failures.

Failures like this put lives and property at risk. We can't let them happen – period.

Safety-related systems are those in which a failure during operation can have serious or irreversible effects – such as loss of life or limb, severe property damage or large financial losses. Safety-critical systems are safety-related systems that present a direct threat to human life. They can include aircraft control systems, automotive x-by-wire systems, medical instrumentation, railway signaling, nuclear reactor control systems and many other applications.

Safety-critical systems include all of the components that work together to achieve the safety-critical mission. These may include input sensors, digital data devices, hardware, peripherals, drivers, actuators, the controlling software, and other interfaces. Their development requires rigorous analysis and comprehensive design and test.

For example, consider the brake-by-wire system of Figure 1.

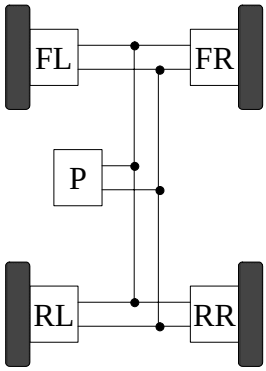

Figure 1: Brake-by-Wire Architecture [3]

A brake actuator interfaces with each wheel (FL, FR, RL, and RR). These actuators are controlled using the brake pedal input (P). Dual-redundant signal paths are used to prevent single-point failures in the wiring.

Brake-by-wire systems are safety critical systems. If the brakes fail to operate correctly, the vehicle will not be controllable. Imagine what would happen if you were driving 120 kph and the brakes would not actuate at all? What if you were driving that fast and the brakes suddenly locked on without command?

Safety-related problems can occur in development (design errors) or in operation (failures). The design may be weak or not robust enough for the operating environment. Design standards and practices, including rigorous requirements management, engineering analyses, design reviews and testing can support validation of the design.

Failures in operation are unavoidable. Stuff breaks! Whether it's a wire in a harness or a worn-out relay – hardware will

eventually fail. Safety-critical systems are designed such that the effects of these hardware failures are predictable (i.e. deterministic) and not catastrophic. Architectures are designed to consistently maintain and control the safety of the system when failures occur.

Robust Design + Controlled Operation = Safe Mission

Remember that safe actuator control is guaranteed only through robust design and robust operation. Design robustness is achieved through robust processes and engineering design practices. Controlled operation is achieved by thorough monitoring, fault detection and mitigation strategies.

II. ACTUATOR CONTROL DESIGN FROM A SYSTEMS PERSPECTIVE

This paper focuses on actuator control systems because these applications provide a relevant mixture of system, hardware and software functions that integrate the broad requirements of safety-critical systems. Consider the following system block diagram of an actuator control system.

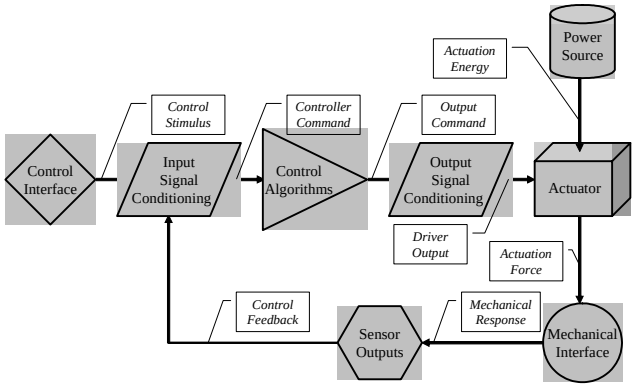

Figure 2: Actuator Control System Block Diagram

An operator (pilot, driver, etc.) uses the control interface (brake pedal, throttle, flight control stick, etc.) to stimulate the control system. The control algorithms drive an output command to the actuator, which applies force to the appropriate mechanical interface (brake, aileron, etc.). A feedback response of the mechanical interface is collected through feedback sensors. Signal conditioning is applied to inputs and outputs to convert raw signals to usable forms.

Most of these control system functional blocks are hardware functions, implemented by some mechanical or electrical components. The control algorithms are typically software functions, implemented by a computing program installed to and operating on a processing device. Signal conditioning can occur in both hardware and software.

Controllers generally use some form of built-in-test (BIT) to validate interfaces and control variables. BIT is designed to detect failures in any of the system elements. BIT is the primary means to verify the integrity and operation of system and hardware components before, during and after the mission.

Systems engineers typically perform a BIT analysis as part of the design process of a safety-critical system. This BIT analysis considers potential failures throughout the systems and verifies that these failures will be detected and mitigated by the BIT functions. For instance, if the power source of Figure 2 fails (e.g. loss of power), BIT should detect the failure and respond by ensuring that the system responds in a safe manner.

As safety-critical systems are becoming more complex and computer-controlled, the role of software is becoming more dominant for both control and monitoring. BIT functionality can be implemented in hardware or software form. Software implementation of BIT functionality is preferred. Hardware that implements BIT functionality can itself be subject to failure, resulting in false positive or false negative failure detection. Engineering design practices suggest that hardware that implements BIT functions should not reduce reliability or system safety.

Now let's consider a simple actuator system from the perspective of controller implementation. Figure 3 provides a general block diagram of controller implementation. The arrows represent a simplified functional flow for the actuator. Actual data flow can be much more complex. Refer to the figure for the following discussion.

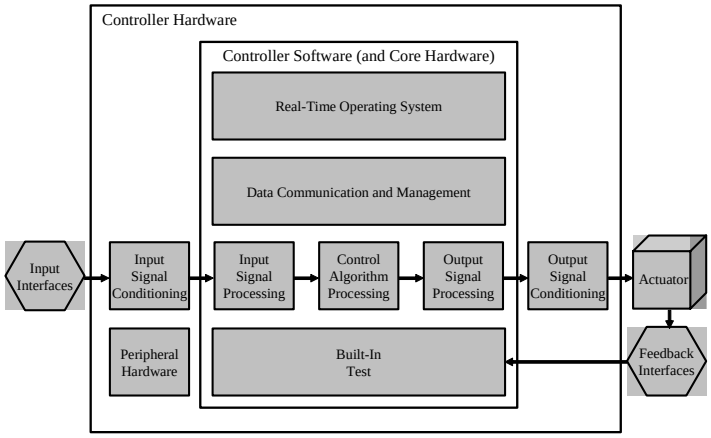

Figure 3: Actuator System Implementation Block Diagram [4]

The actuator, input interfaces and feedback interfaces are external to the controller hardware. The interfaces could be physically near the controller or could be quite far away and connected through wire harnesses.

The controller hardware provides I/O signal conditioning, peripheral interfaces and a core processing platform. Signal conditioning converts the input signals into forms that can be interpreted by the core processor, and translates processor outputs into driver signals for the actuator. These driver signals may be high current signals that directly drive the actuator or they can provide indirect control of an external power source, as shown in Figure 2. Peripheral hardware may include watchdog monitors, data communication devices, power converters, protective devices and filters, configuration hardware and other peripheral hardware. The core processing platform includes the processor, memory, and any other devices needed to support processing operation.

Controller software, which resides and operates on the core hardware, has a real-time operating system (RTOS) that performs all executive management and task scheduling, as well as other operating utilities. This may be an ad-hoc operating system; or it can be a commercial off-the-shelf RTOS, such as

Integrity, VxWorks or LynxOS, that is certifiable to the safety criticality level of the application.

Input signal processing software reads raw data inputs from the hardware and massages them into usable form for the software algorithms. This massaging includes software filters (e.g. lag and lead filters), discrete debouncing, and range detection. Output signal processing converts the results of the control algorithms into physical signals to drive to hardware.

Control algorithms provide the heart of the actuator control functionality. This is where the plant models, control loops and decisions are processed. Control data and variables are read and real-time algorithms are processed to deterministically drive the appropriate real-time response of the actuator.

The data communication and management functions manipulate the data associated with the inputs and outputs, including digital communication to external components. These functions also provide any processing associated with redundancy or cross-channel communication. For example, a quad-redundant actuator system would have a complete set of data for each of the four channels. Some level of data synchronization is needed to maintain coordinated control of the four channels. Each channel may perform BIT on itself and one or more of the other channels. Depending on the intricacy of data communication, the amount of data that must be managed can become huge and coordination can become complex.

Actuator controls include BIT functions to detect faults and isolate them to individual components or small groups of them. Failure detection is usually accomplished by conducting a series of tests during operation and comparing results to expected values. Isolation logic determines if the fault occurred inside the controller or was the result of external interfaces. The designer's goal is to test any potential fault that has an effect on the safe operation of the mission in the appropriate mode of BIT.

BIT functionality can be categorized in the following modes:

1) *Power-up BIT (PBIT)* – This is also known as Start-up BIT. It executes automatically upon the application of power and provides a rapid verification of the ability of the actuator control to operate. For example, if the core processing platform is not working on power-up, the mission can be halted before operator safety is jeopardized. Also, if actuator feedback interfaces are detected to be failed, the actuator may not be controllable, and the mission would be aborted. The scope of PBIT testing depends on application powerup requirements. Designers must trade-off PBIT test coverage with PBIT timing constraints.

2) *Continuous BIT (CBIT)* – This is also known as Periodic BIT. CBIT runs throughout operation to provide continuous monitoring of all system components. In safety-critical systems, it becomes important to test as many system components as possible throughout operation in order to reduce the exposure to failures that become active during the mission. For example, a current feedback sensor on an aircraft surface actuator can help detect if the flight surface gets mechanically stuck during operation. When safety-related failures occur during operation, CBIT helps the operator detect and mitigate them. CBIT completion time must be considered when the maximum failure exposure time, which is the maximum time between when a fault occurs and when the fault is detected and mitigated, is critical.

3) *Initiated BIT (IBIT)* – This may be called Maintenance BIT, or other alternative names. IBIT is an extensive set of tests, initiated by the operator, which occurs when the system is in a stable, known environment. The environment must be controlled because control of the actuator is given to the BIT functions rather than the normal control algorithms. An example is a range test on an actuator. Suppose an actuator moves a swing-arm from 0^o to 180^o. The system cannot test the full range of motion during operation, unless the application guarantees to move the swing-arm over its full range of motion during the mission. The operator can use a special controlled test in IBIT to verify that the swing-arm can operate over its full range of motion.

Engineers perform a BIT analysis by listing all the potential safety-related faults and assessing whether each fault would be detected and mitigated by one or more categories of BIT. The analysis also determines the exposure time of a fault. Failures should be mitigated before the loss of safety.

Redundancy is a common design practice that is often used to maintain functionality after a failure occurs. For instance, consider the following quad-redundant system:

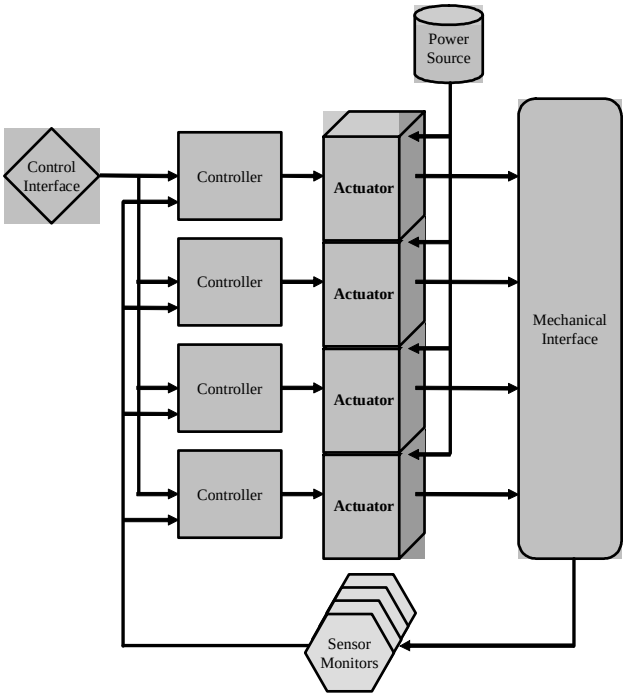

Figure 4: Quad-redundant Actuator System

In this case, functionality is maintained by four redundant operating channels. Failures that occur in a controller, actuator or mechanical monitor can be mitigated by the other operating channels, perhaps with a degraded level of performance. The effects of failures in non-redundant components, such as the

control interface and power source in this case, cannot be alleviated.

Actuator control systems encompass hardware, software and redundant system interfaces. There are many failure modes, but system designers have developed approaches to mitigate them. Various response categories have been developed to describe mitigation of these system failures:

1) *Fail-Operational* – From an operator perspective, the system behaves normally. The failure is reported, but system operation is unaffected. The system components that remain functional typically take over the functions of the failed components. *Example*: Quad-redundant flight control surface actuator systems typically can meet performance requirements with only three channels. When one channel fails, the system continues to perform using the remaining three channels.

2) *Fail-Passive* – The system outputs a predetermined desirable state, such as a power disconnect to the actuators. The failure is reported, and operation is typically reverted to a backup mechanism. Intervention may be required, but the system generally remains under control, although usually in some degraded fashion. *Example*: Many marine actuator control applications provide a mechanical backup that is automatically engaged when automated systems fail, thereby enabling the pilot to "limp home". The vehicle may not meet performance requirements, but it will be able to travel to a safe environment.

3) *Fail-Safe* – This approach is used to provide a safe response when normal, predictable control is not possible. In this category, which is sometimes considered a subcategory of fail-passive, the actuators are not controllable in a manner to meet the performance specification. However, the system employs mechanisms to force the actuator in a known state that maintains a safe operational state. *Example*: Hybrid-electric vehicles (HEV) have experienced failures that cause "runaway motors" where motors speed up out of control and transfer uncommanded power to the drive axles. A fail-safe mitigation would disconnect power from the motor, usually through high-current relays, when a failure occurs. The vehicle may not be drivable, but the passengers and cargo remain safe.

4) *Fail-Active* – This category is highly undesirable and strikes fear into the hearts of safety-critical systems designers. It occurs when mitigation into one of the other categories is not possible. It cannot be prevented completely, so applications typically allow a small probability, such as 10^{-9}. In a fail-active state, actuators drive the system in an uncontrollable and unpredictable (nondeterministic) manner. Safety-critical systems are designed to minimize fail-active scenarios. *Example*: Automotive steer-by-wire applications have a unique challenge in that typical steering wheel interfaces have common mode failures that could inhibit steering control.

Application performance specifications may require system architectures to apply these degradation categories through multiple failure modes. For instance, after the first failure in a quad-redundant flight actuator system, the system may be required to remain fail-operational. After the second failure in the system, the application may require fail-passive operation. After three failures, the application may require fail-safe operation. System designers must be aware of failure response requirements and select architectures and components accordingly. In this example, the actuator system must be able to adequately move the aircraft surfaces with only two operating channels. Therefore, the actuators must be sized to enable operation with only two functioning actuators.

Now that we have discussed what it means to have a safety-critical actuator control application, let's consider some tools to help us on our journey to a successful design.

III. INDUSTRY SPECIFICATIONS AND STANDARDS

We claimed earlier that safety-critical applications are guaranteed only through robust designs and controlled operation. Industry specifications and standards provide generally accepted engineering practices and guidelines to support safety-critical application development. Development programs will identify applicable industry specifications and standards early in the life-cycle, as compliance to these standards can have a significant influence on development cost and schedule.

Standards can contain requirements, guidelines or both. Requirements are processes or design practices that must be implemented. Guidelines provide recommended practices and methods for satisfying requirements. They can be applied to development processes and/or design implementation, as categorized in Figure 5.

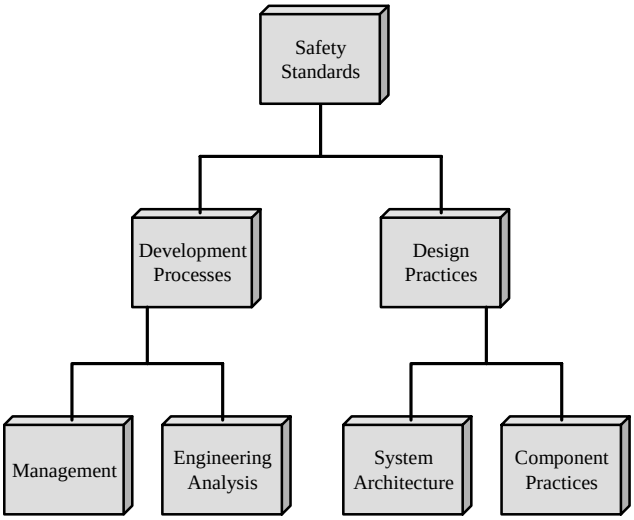

Figure 5: Safety Standards Categories

Examples of development processes are software development plans, quality management processes and verification plans. Engineering analysis standards include reliability analysis and other performance analyses. Design practices are very specific to the application industry. For instance, the Federal Motor

Vehicle Safety Standards (FMVSS) provide design practices for motor vehicles.

Although the scope of this paper is not to provide a tutorial of safety standards, it is relevant to identify some standards and their applicable industries and applications. Table 1 highlights some common standards.

Table 1: Safety Standards and Applicability

Standard	Applicability
MIL-STD-882	Safety systems for US DoD
ARP 4761	Airborne systems safety assessment
ARP 4754	Aircraft systems certification considerations
RTCA/DO-254	Aerospace and Aviation Hardware
RTCA/DO-178B	Aerospace and Aviation Software
IEC 61508	Generic 'Programmable Systems'
MOD DEF STAN 00-56	Automotive system safety processes
MOD DEF STAN 00-55	Automotive software safety processes
UL-1998	Software design requirements (non-process)
IEC 880	Nuclear Industry
SAE J2344	Guidelines for Electric Vehicle Safety
SAE J1938	Vehicle Electronic Systems
SAE J1739	Automotive FMEA
IEC 601	Medical Equipment
EN 50128	Railway Industry
FMVSS	Motor vehicle systems and components (US)
CMVSS	Motor vehicle systems and components (CAN)
MISRA / MISRA C	Automotive Industry Software

Many of the standards listed above classify systems into five System Integrity Levels (SILs) from 4 (highest safety) to 0 (not safety related). DO-178B uses A (highest safety) to E (lowest safety). The SIL of the development project drives the processes for that project. For instance, to test software in a DO-178B Level A project, such as a flight control system with no mechanical backup, all code statements must be executed, all code branches must be followed, and actual target hardware must be used.

To steal a quote from the movie *Pirates of the Caribbean*, many of these standards are "more what you'd call 'guidelines' than actual rules". The designer's real mission is to achieve safety certification. Deviations can be taken, if they are coordinated with and approved by the application's certification representative.

IV. DEVELOPMENT STRATEGIES

Systems that are well-planned, designed, tested and certified can be trusted with human life. The design goal of safety-critical system development is to eliminate failure modes that could result in catastrophic failures, or if that is not feasible, minimize the risk such that there is a very low probability of catastrophic failure.

Safety is proven not only through testing, but also through safe planning and development processes. Testing is a necessary, but not sufficient, element of safety-critical development – it only proves the presence of errors, not their absence. We haven't yet found a way to "test safety" into the system.

Safety must be built-in throughout the development life-cycle using safety-critical development plans and procedures, independent reviews, and comprehensive test cases.

This section highlights some basic development strategies that have proven to be important in many of the author's designs. It is by no means a comprehensive list, but it does capture some thought processes that must be exercised.

a. System Architecture

i. Analyses and Processes

The industry standards described earlier stress formal planning, development and test processes. Systems engineering for safety-critical systems emphasizes formal development processes for system requirements management, iterative design and analysis, and comprehensive verification. These guidelines, coupled with robust design practices, provide safe system architectures.

A safety analysis, such as the system safety assessment process described by guidelines like ARP4761 [5], is performed on the system architecture to quantify safety performance and ascertain what safety-critical design practices should be implemented. The safety analysis (including hazard analysis) and system requirements iteratively evolve together throughout development.

Safety analyses for embedded systems evaluate not only static failures, but also dynamic failures. For example, embedded controllers can exhibit various system safety failures including but not limited to [6]:

- A required event that does not occur
- An undesirable event that does occur
- Order of events sequence failures
- Timing failures in event sequences (maximum and minimum time constraints)

Without making claims of the relevance of Murphy's Law in engineering designs, a systems engineer for a safety-critical application must assume all possible failure modes will occur. Through robust design processes, inspection, analysis, design or test, the engineer must quantify that the probability of catastrophic failure meets application requirements (e.g. 10^{-9}).

ii. Partitioning and Modularity

The system architect must determine which functional components are safety-related, and what safety integrity level should be applied to each of them. The requirements of the system are allocated and distributed to the functional components. The safety integrity level of the system is the minimum safety integrity level for any functional component that performs a safety-critical function of the system. Selection of the safety integrity level is important because it determines which development guidelines, processes and standards will be required in development.

Functional modularity of the system architecture may be implemented using the ARINC-653 [7] open system applications programming interface. ARINC-653 provides brick-wall time and space partitioning of separately loadable software applications. These individually loaded programs can

be of different levels of criticality, which may result in greatly reduced test and certification costs and increased portability and reuse.

Safe partitioning may be used at the system, software or hardware level to isolate failures and prevent them from permeating to other parts of the system. For example, ARINC-653 was developed as a standard for time and space partitioning in software.

As systems become more complex, the trend is to make system architectures more modular. This enables individual functional components to be certified in applications, and then reused in subsequent applications. When those components are integrated into a larger system, their aspects of certification can be integrated with those of the larger system.

Certification authorities, such as the FAA, will typically only certify systems at the application level. For example, a flight control system is certified for an aircraft, not as a stand-alone application. However, safety-critical standards and the certification authorities have made accommodations for reuse of individual components. DO-178B includes section 12.1 to consider the use of previously developed software. [8]

iii. System Redundancy

Redundancy is the primary means for making fault tolerant systems of components. Simple examples of redundant systems are shown in Figure 1 and Figure 4. Consider these two example architectures in the following discussion.

The first example is a brake-by-wire system. If one brake actuator fails in an inactive state (i.e. the brake cannot be applied), the brake actuators on the other three wheels will likely stop the vehicle. Redundancy allows the system to perform, albeit in a degraded mode, when failures occur. If a brake fails in an active state (i.e. the brake is applied when not commanded), the safety of the operator may be jeopardized, especially if it occurs while the vehicle is traveling at high speeds when the failure occurs. The Federal Motor Vehicle Safety Standards (FMVSS) provide guidelines to consider when developing this type of system. Depending on your application, the fail-safe position of brake actuators may be full-active, partially active, or full inactive.

The second example shows a quad actuator system whereby all actuators control the same mechanical interface. Failures in the controllers, actuators and feedback sensors are mitigated by redundant components. However, failures in the control interface, power source and even the mechanical interface cannot be mitigated. These are called single-point failure modes, in which a failure at a single point will cause the system to be inoperable. Good system architectures minimize single-point failure modes, and ideally have none.

Redundancy practices are not only concerned with hardware redundancy, but also data and software redundancy. Quad-redundant systems usually share their data among the channels. Each channel understands what the other channels are reading as inputs and trying to control as outputs. Redundant systems use cross channel data voting to converge on consistent control parameters. For instance, the four feedback sensor readings may be averaged to determine a consistent variable value to be used by all channels. Cross-channel data that has critical time constraints will synchronize the channels when acquiring the data in order to maintain real-time integrity. The preferred strategies of redundancy management, as this practice is called, are as opinionated as your favorite pizza. Engineering organizations use the redundancy management strategies that best fit their development processes.

Redundancy is one means of improving the system architecture reliability. There are other design practices that are described in detail throughout the industry specifications and standards.

iv. Isolation and Independence

Whereas redundancy is concerned with adding redundant components to maintain performance; isolation and independence go even further by mitigating common mode failures, which can affect all channels of a redundant system. Isolation ensures that any failed components are completely isolated from the system. Architecture independence strategies use completely independent technologies and components to mitigate system failures. An example is an aircraft with an electronic flight control system and an independent mechanical backup flight control system. Even if the entire flight control system is disabled by a common mode failure, the pilot will be able to operate the aircraft with the mechanical backup system.

An example of an architecture that uses both isolation and independence is a multi-source power control system for hybrid electric vehicles (HEV). The system uses both batteries and ultra-capacitors as supplementary power sources. The sources are completely independent and isolated from one another. This prevents propagation of failures from one power source to the other.

v. BIT Analysis

Another important element of architecture development is the BIT analysis discussed earlier. When engineers evaluate the failure modes, they must ascertain which BIT mode(s) (PBIT, CBIT, or IBIT) is most feasible for detecting and reporting the failure. For instance, actuator power sources can be tested before the mission is begun. It is more practical to test failures before the mission begins (PBIT) than to try to mitigate the failure in operation. Powerup timing requirements may constrain the amount of testing performed during PBIT. Some failure testing may require the mission to be underway (CBIT) or for the actuator system to be in a controlled environment (IBIT). The combination of these BIT modes should perform comprehensive system testing in order to prevent operation with active catastrophic failures.

b. Hardware Design

Many of the development strategies for system components also apply to hardware components. For example, hardware designers may choose to employ buffers to isolate signals or external watchdog monitors to provide independent processor integrity validation.

Hardware independence must also consider the allocation of components to functional blocks. For instance, a quad op-amp package should not be used for both the control and monitor of the same interface. In this case, a component failure will affect both the control and the monitor interfaces. Similarly, it should not be used to process a signal interface for all four channels in a

quad redundant system. Designers must ensure that faults cannot propagate between functional blocks.

Reliability, maintainability and safety analyses are prevalent in safety-critical hardware design. MIL-STD-882C defines activities needed to identify possible hazards and analyze/reduce the risk of occurrence in use. These analyses could include safety program plans; Failure Modes and Effects Analysis (FMEA); Failure Modes, Effects and Criticality Analysis (FMECA); Fault Tree Analysis (FTA); hazard analysis and other safety and certification analyses. Using these analysis techniques and the prediction processes of MIL-HDBK-217, reliability engineers consider all safety-related failure modes and quantify parameters such as mean time between failures (MTBF), which is the mean time to system failure (MTTF) plus the mean time to repair (MTTR).

All electronic hardware systems fail at one time or another, so designers must understand what will happen when a malfunction occurs. A loss of function may be acceptable, but unpredictable or catastrophic effects are not tolerated.

Safety-related hardware, which is hardware that is implemented solely to detect or mitigate failure modes, requires careful design consideration. This type of hardware may reside inside the controller (e.g. brownout detection circuitry that shuts down the controller when power drops) or external to the controller (e.g. mechanical stops used to prevent an actuated swing arm from exceeding a safe operating range). A system risk is created when hardware is implemented solely for monitor purposes. These monitor interfaces may fail, causing false failures to be detected by the system, and thereby taking the actuator out of operation. Furthermore, operational testing that uses these interfaces typically requires the unsafe condition to be applied, and therefore can only be completed in a controlled environment such as IBIT. Use of safety-related hardware must be carefully analyzed for its effects on system performance.

c. Software Design

Many of the safety-critical system architecture development strategies are also applicable to safety-critical software designs. Partitioning, redundancy and real-time performance practices are employed in software architecture as well as system architectures. System independence and diversity can also be applied in software by using multiple, independent technologies to perform the same task. Software designs should also utilize independence of function and monitor so that the monitor software is not rendered inoperative by the failure that is being monitored.

Independence is also applied to development processes. Depending on the safety integrity level, DO-178B [8] demands independence between requirements, design and verification. The engineer that designs a software function cannot be the same engineer that defined the requirements; and the engineer that verifies a software performance cannot be the same engineer that designed it.

Safety-critical real-time software must be correct and deterministic. It must perform to specification (correct) and it must always react the same way to the same inputs (deterministic). Correctness is tested through verification and validation. Determinism is guaranteed only through design processes, practices and analyses. When software is deterministic, the response to a failure condition is guaranteed to behave in a known manner.

DO-178B also demands the use of requirements-based testing as an effective means to reveal design errors. Test cases should include normal range and robust operation. Methods may include low-level testing (i.e. unit testing with structural coverage), software integration testing and hardware/software integration testing.

Recall the F-16 simulation case discussed earlier. These scenarios include normal range tests (e.g. recovering from a spin) and robustness tests (e.g. jettisoning cargo while flying upside down). It is certainly preferable to test responses to these operating modes during simulation than during field test.

Much more is known about how to develop safe mechanical and electrical systems than safe software programs. To mitigate this issue, the aerospace industry has taken a strong position on software safety and the automotive industry is also building standards. Other safety-critical industries are also following suit, as can be seen in Table 1.

d. Sample Safety-Critical Development Process

Many of these strategies are described throughout the safety-critical development processes of the appropriate industry. Product requirements generally prescribe the specific guidelines and standards that must be followed for application development.

Figure 6 shows a sample safety-critical development process used for aircraft.

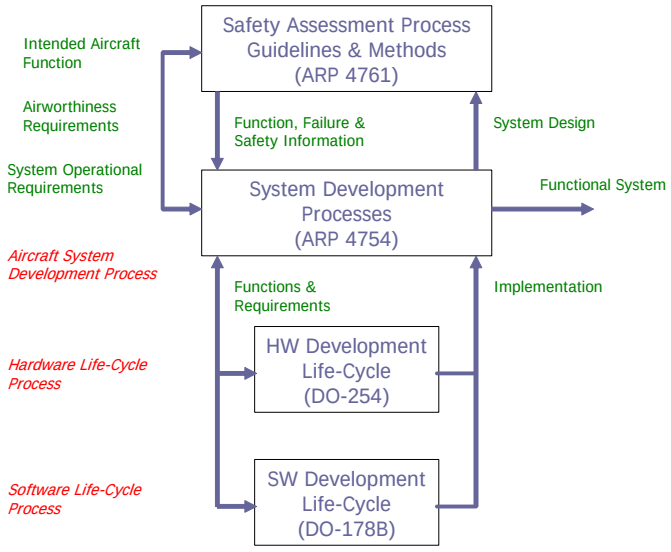

Figure 6: Safety-Critical Development Process

Design and development using these standards is an iterative process. There are interdependent relationships among the processes. Safety requirements are allocated by system development process to hardware and software. The hardware and software design performance analyses are used to verify requirements from the safety assessment process. The analyses continue until a system design is implemented that meets the safety and functional requirements of the application.

e. Tools

Tools used to assist in the development of safety-critical systems must not compromise the safety of the system. Any design or test processes that are automated must be proven to be failure free – either by the automated process itself or by subsequent testing or analysis.

Let's consider two examples:

1) Automated software code generation – DO-178B states that tools used for software development must be qualified to the same level as the software [8]. Unqualified automation tools may be used if appropriate qualification processes are exercised on the output. For instance, auto-generated code can be manually tested in accordance with DO-178B guidelines.

2) Programmable logic devices – DO-254 states similar requirements for programmable hardware [9]. Outputs of an unqualified FPGA compiler must be tested in accordance with DO-254 guidelines.

Lastly, do not underestimate the power of modeling and analysis tools to perform safety analyses and characterize the performance of safety-critical systems. These tools can be used to generate and model "what-if" scenarios quickly and effectively. As was seen in the F-16 simulation cases, it is much more effective to discover failures during simulation and test than in a fielded application. Tools like MatLab and MATRIXx can simulate system failures and validate that the design responds safely. Engineering analysis tools, such as those provided with software compilers and PLD design tools, can aid engineers in evaluating fault modes.

V. EXAMPLE CASES OF SAFETY-CRITICAL ACTUATOR CONTROLS

Throughout my career, I have worked on safety-critical applications on a variety of platforms, including airplanes, locomotives, heavy-duty trucks and busses, passenger vehicles, power systems, and medical diagnostics. Interestingly, the concepts needed to make these systems fault-tolerant, fail-operative or fail-safe are quite common.

The following describes some examples of safety-critical systems.

a. Fly-by-wire Aircraft Control Systems

In the past, many aircraft had mechanical backups for their electronic flight control systems. However, fly-by-wire systems with no mechanical backups are becoming more prevalent in both military and commercial environments. Actuation systems, such as those that move flight surfaces, have become safety-critical.

b. Automotive Drive-by-wire System

Automotive manufacturers are now developing full authority drive-by-wire systems, including brake-by-wire, steer-by-wire, and throttle-by-wire. The general practice has been to achieve fault tolerance through hardware redundancy. Redundancy can be cost and space prohibitive in automotive applications. Software standards such as MISRA are being developed to facilitate fault tolerance with less hardware redundancy.

c. Railway Signaling System

Railway signaling systems enable operators to direct trains while preventing trains from colliding. A malfunction in these systems could cause death. In many subway signaling architectures, safety-related hardware is used to provide redundancy and control independence is provided through coordinated actions by the train operator and the dispatcher.

d. Submarine Depth Control System

Automatic depth control of a submarine presents challenges in safety-critical control design that are similar to those in aircraft applications. The preferred fail-safe response depends on the application and environment. In battle environments, underwater concealment is important, so surfacing the submarine may not be the best mitigation. However, the submarine pilot must be able to eventually surface a failed vehicle to recover personnel.

e. Nuclear Power Station Shutdown

All nuclear power stations require a protection system that closes the station down in the event of a malfunction. This protection system must mitigate the consequences of the situation. Often, both hardware and software protection systems are used, since the multiple redundancy increases confidence in safety.

f. Medical Systems

Medical systems can be directly responsible for human life. Some examples include hardware that controls laparoscopic surgery control devices, software that monitors safe amounts of x-rays, and information systems that doctors use to coordinate medication. Safety development standards have recently become a strong focus in the medical industry.

The Therac-25 radiation therapy machine has resulted in lost lives due to "the most serious computer-related accidents to date" [10]. At least six accidents occurred during 1985-1987. The machine did not respond appropriately to failures, and typically overdosed the patients.

Designers were overconfident in the ability of the software to adequately respond safely to faults. They relied heavily on software to protect against overdose, but the software was not designed to safety standards. Furthermore, they could not replicate the failures of early accidents. The software design did not provide the required testing and failure analysis.

VI. CONCLUSION

Safety is a system issue that requires robust design and controlled operation during failure modes. Actuator control designers for safety-critical applications utilize industry standards and development processes, as well as engineering best practices, to develop safe, validated and deterministic designs.

Developers of safety-critical systems must learn to expect the unexpected. The response to an unexpected event must be known and designed into the system before that unexpected event occurs. Predictable and deterministic responses to all potential failure modes are paramount in safety-critical design.

If we all keep our wits about us as we develop these safety-critical systems, we can sit back and relax in our confined space

onboard an aircraft without worrying that a design failure will send our steel horse for a loop – even when we fly over the Equator.

Many of the issues, practices and strategies described in this paper are a result of previous or ongoing projects at On Target Technology Development.

VII. REFERENCES

[1] Janssen, B., "F-16 Problems - Contribution to the RISKS Digest", Volume 3, Issue 44, 1986.

[2] McGettrick, A. "Software for Safety-Critical Systems – Introduction Lecture", University of Strathclyde, 2003.

[3] Touloupis, E., et al, "Safety-Critical Architectures for Automotive Applications," Electronic Systems and Control Division Research, Loughborough University, UK, 2003.

[4] Socci, Vincent P., "Control Systems in a Nutshell," On Target Technology Development. Endicott, NY, 2004.

[5] SAE, Inc. "Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment," Report ARP 4761, Warrendale, PA, 1996.

[6] Pimentel, J. R., "Designing Safety-Critical Systems: A Convergence of Technologies," Kettering University. Flint, Michigan.

[7] Aeronautical Radio, Inc., "Avionics Application Software Standard Interface, ARINC Specification 653," January 1997.

[8] RTCA, Inc., "Software Considerations in Airborne Systems and Equipment Certification," Report RTCA/DO-178B, Washington DC, USA, 1992.

[9] RTCA, Inc., "Design Assurance Guidance for Airborne Electronic Hardware," Report RTCA/DO-254, Washington DC, 2000.

[10] Nancy Leveson and Clark Turner, "The Investigation of the Therac-25 Accidents", Computer, 26, 7 (July 1993) pp 18-41.

ABOUT THE AUTHOR:

Vincent Socci is a product manager and cross-disciplined engineer (systems, HW, SW). His technology expertise includes embedded systems, sensors and signal processing, power control systems, and diagnostics. Socci has over 15 years of experience in aerospace, automotive and defense systems. He facilitates business and technology courses for the State University of New York and the University of Phoenix. Mr. Socci holds an MBA in technology management, and MS and BS degrees in electrical engineering. As Principal of On Target Technology Development, Socci supports clients with technology planning, program management, systems engineering and new product development. He has applied the safety-critical design concepts presented in this paper in aerospace, automotive, locomotive, marine, utility and medical applications. He can be contacted at vsocci@ontargettechnology.com.