Programmable Logic Device (PLD) Safety Design Approach

by Martin S. Chizek
Orlando, Florida

Programmable Logic Devices (PLDs) in ordnance fuze and ignition systems have well-defined design and verification requirements based on U.S. Department of Defense (DoD) Safety Review Board guidelines and military standards. However, there are few established safety design and verification requirements for PLDs used in non-fuze safety-significant applications. The primary objective of this paper is to (1) establish a process that assures that PLDs in products and systems are developed and tested to a level of rigor commensurate with the safety risk of the specified application, including fuze and non-fuze safety systems, and (2) to comply with recent guidance from DoD Software System Safety Technical Review Panels on firmware and programmable logic safety assurance. The paper's secondary objective is to make the PLD safety process applicable to non-DoD and commercial programs such as autonomous vehicles, aerospace and energy systems. To meet this objective, this document incorporates best practices of NASA, commercial aviation, the Nuclear Regulatory Commission (NRC), and from international programmable electronic functional safety standards.

Introduction

The DoD software safety process includes firmware and programmable logic. PLDs in ordnance fuze and ignition systems have well-defined design and verification requirements based on DoD Safety Review Board guidelines and military standards [Ref. 1]. However, there are non-fuze applications where a PLD alone may control safety-significant functions (as defined in MIL-STD-882E) [Ref. 2] and only software/firmware provides redundancy or a safety interlock. Additionally, functions such as initiating electro-explosive devices (EEDs) and squibs, directed energy arming and firing, weapon line of sight control, weapon termination, and autonomous vehicle control may be considered safety-critical under many applications. There are few established safety design and verification requirements for PLDs used in these safety-significant applications.

Programmable Logic Device Safety Background

Software, Firmware and Hardware Device Safety Standards

For DoD programs compliant with MIL-STD-882E, the assessment of software's contributions to system risk considers the potential risk severity, and the degree of control that software exercises over the hardware. This establishes the Software Criticality Indices (SwCIs) used to define the required Level of Rigor (LOR) tasks.

Safety standards in domains such as medical, transportation, aviation, and nuclear and industrial control systems designate a Design Assurance Level (DAL) or Safety Integrity Level (SIL) to indicate the tolerable failure rate of an electrical/electronic safety function, and to assign the level of risk reduction necessary to maintain an acceptable level of assurance/integrity. The higher the assurance/integrity level, the more stringent are the verification and validation activities, and the evidence of compliance [Ref. 3].

The Federal Aviation Administration (FAA) defines most PLDs as Complex Electronics Hardware (CEH), whereby a comprehensive combination of deterministic tests and analyses *cannot* ensure correct functional performance under foreseeable operating conditions with no anomalous behavior. In other words, the item is so complex that it is impossible or impractical to completely test and analyze it; instead, one must rely on a rigorous, structured design assurance process for the item, in addition to comprehensive tests and analyses [Ref. 4].

Nuclear power safety standards consider firmware to be software in a programmable device. NASA maintains that complex electronics such as Field Programmable Gate Arrays (FPGAs) are not firmware because what resides in them is not a software program; rather, software is used to define the logic structure for a hardware device, which is what these devices become once they are configured [Ref. 5]. The NASA software safety standard specifies that software to be executed on a processor embedded within a PLD be evaluated from a software safety perspective, while the design and resulting hardware is evaluated from a system

safety perspective [Ref. 6]. This would seem particularly appropriate for a System on a Chip (SoC), which may have embedded processors in addition to programmable logic cells.

Programmable Logic Device Technologies

Programmable logic devices such as FPGAs use a collection of configurable logic blocks arranged in an array, with interspersed switches that can rearrange the interconnections between the logic blocks. Each logic block is individually programmed to perform a logic function (such as AND, OR, XOR, etc.) and the switches are then programmed to connect the blocks and realize the complete logic function. The device may use Antifuse, Flash, Electrically Erasable Programmable Read-Only Memory (EEPROM) or Static Read-Only Memory (SRAM) technology to store the programming. In many devices, the configuration is volatile and must be reloaded into the device whenever power is applied, or different functionality is required.

Antifuse devices are one-time programmable and non-volatile, which means the device retains its memory contents even when powered off. An antifuse device contact is in a high-impedance "open" state until a relatively high voltage is applied to transition it to a low-impedance "closed" state; it also retains programmed information indefinitely. These devices do not require external storage, such as an EEPROM, for program storage and are not susceptible to Single Event Effects (SEE) from ionizing radiation. This is the technology generally preferred by the DoD Fuze and Ignition System Safety Boards for Safety-Critical applications, and by NASA and the FAA for space and high-altitude environments.

Flash-based devices use flash EEPROM, allowing them to be erased and reprogrammed, and are essentially non-volatile. These devices have limited operational reconfigurability and write cycle life and retain programmed information for years. Configuration is considered hardened (but not immune) to SEEs and they are suitable technology for controlling most safety-critical functions, and all safety-related functions.

SRAM devices utilize use Complementary Metal-Oxide-Semiconductor (CMOS) technology and volatile memory; contents are lost whenever the power is removed. After every power up, the device reads the configuration bitstream from a non-volatile memory such as an EPROM, EEPROM or Flash memory, and transfers the memory contents to volatile RAM cells. SRAM-based FPGAs are the most common type for commercial applications and have the highest capacity and performance of the FPGA technologies. SRAM devices are considered the most susceptible to SEEs from ionizing radiation and should not be used in critical high altitude and satellite applications, or for autonomous control of safety-critical functions [Ref. 7].

PLD Faults and Failure Modes

The logic device hardware failure rates, failure modes and technology selection must be considered as part of the system safety program. The primary concern is that a device controlling safety-significant functions, signals and/or data will fail to operate, or will operate in an unpredictable manner, resulting in a safety mishap. In general, a PLD should be designed to perform its safety function when subjected to hazards — external or internal — that have significant potential for defeating the safety function. Such hazards include input and output processing failures, precision or roundoff errors, improper recovery actions, electrical supply, input voltage and frequency fluctuations, maximum credible number of coincident signal changes, and environmental stressors [Ref. 8].

Some specific errors, faults and failure modes that have historically resulted in unsafe PLD behavior include:

- safety requirements not included in device and Common Component Architecture (CCA) specifications
- device not rated for its eventual operating environment or service life
- improper grounding or termination of pins
- exposure to power transfers, interrupts, transients and/or brownout resulting in unpredictable behavior
- inadequate power supply voltage or rise/fall times, power interrupts, high current draw during initialization, noise on clock/interrupt lines and data busses, and electromagnetic effects and electrostatic discharge resulting in Common Cause Failure
- reset circuitry failure causing indeterminate state of device outputs
- initialization in Safe State failure, or failure to transition to Safe State
- memory or messaging corruption of stored bit map or logic equations
- transients and noise on EEPROM causing false write cycles to be generated, resulting in inadvertent altering of the device's contents
- optimization use during synthesis introducing changes in configuration data or logic not intended by the designer
- use of unproven compilers, synthesizers, and other development tools introducing errors in configuration

- failure to maintain configuration control, or to conduct regression testing after configuration changes
- single Event Effects and bit flips caused by radiation, electromagnetic interference or power supply transients resulting in unpredictable behavior

Common Cause Failure

Common Cause Failure (CCF) is a term used to describe random and systematic events that cause multiple devices, systems or layers to fail simultaneously. The use of PLDs in safety systems has led to concerns that software/firmware design errors could lead to a single-point failure or a CCF, which might then disable one or more redundant safety functions. Common Mode Failure (CMF) is a subset of CCF and describes the simultaneous failure of two devices in the same mode, such as two identical devices in a redundant subsystem. Potential sources of CMFs include common firmware used in redundant channels; sensitivity to electromagnetic interference and radiation; improper modification of software and hardware configurations; improper system integration; and inappropriate commercial-grade digital electronics [Ref. 8].

Latent software faults are design errors that may remain undetected until challenged by a triggering mechanism. A latent fault only becomes an issue if the fault is systematic (i.e., existing in multiple divisions) and results in an unsafe condition. CCFs that result in a designated safe state are still inherently safe. The failures are considered simultaneous (and therefore CCF) when the time interval between the failure detections is too short for repair or recovery measures to be taken.

Single Event Effects

SSEs on electronics resulting from elevated levels of ionizing radiation have long been an issue for spacecraft and high-altitude aircraft. SEEs occur when atmospheric radiation, comprising high-energy neutrons and alpha particles, collides with specific locations on semiconductor devices. The major circuit effects from high-energy particles include transient current pulses, changes in memory values (bit flips), and latch-up. Trends toward higher-density semiconductors, lower operating voltages and greater usage of memory bits and registers increase the sensitivity to atmospheric radiation and, correspondingly, higher likelihood of SEEs even in ground-based or low-altitude systems.

SEEs are required to be included in the safety assessment of PLDs by the Nuclear Regulatory Commission in nuclear power safety controls, by DoD for logic devices in fuzes and airworthiness certification, and by the FAA for commercial aircraft certification. Mitigation techniques for SEE protection of PLDs include [Ref. 9]:

- using SEE-resistant technologies, such as one-time programmable/antifuse devices; selecting flash-based over SRAM-based reconfigurable devices
- run-time checking of the configuration memory and watchdog timers; error-initiated reloading of the configuration memory (i.e., scrubbing) and periodic resetting of the FPGA logic
- error Correcting Code (ECC) and/or Error Detection and Correction (EDAC) on state machines, control logic and memory data
- redundancy of user memory; e.g., Triple Modular Redundancy (TMR)
- voting schemes (on logical feedback paths on output)
- interleaving; e.g., using logical check-words from physically dispersed locations in the memory array, protecting SRAM cells against soft upsets

Fault tolerance techniques can be implemented on the device (e.g., ECC), at circuit level (e.g., TMR), at assembly level (e.g., watchdog timer) and at system level (e.g., duplex architecture).

Functional Safety Approach

Functional Safety is a subset of the system safety process that focuses on hazards caused by the malfunctioning behavior of electrical, electronic or programmable electronic (E/E/PE) control systems/equipment and related software. Functional Safety depends on the correct functioning of the E/E/PE safety-related systems, other safety-related systems and external risk reduction means to achieve or maintain a safe state for the system/equipment being controlled. The necessary risk reduction allocated to an E/E/PES is expressed as a failure probability limit, which in turn is used to select the Safety Integrity Level (SIL) for the required safety functions. SIL1 represents the lowest level of safety integrity, while SIL4 represents the highest level of safety integrity.

The Functional Safety approach is most common in the energy, automotive, aviation and process control industries, and is described in the international standard IEC 61508 and its derivatives. The Functional Safety approach is recommended for Complex Electronics Hardware (CEH), such as FPGAs, PLDs and SoCs since

it addresses systematic failures of PLD programming, as well as random failures of the hardware device.

Functional Failures

The Functional Safety approach treats hardware and software hazard causal factor categories as "failure events" which result in the inability of a system or a component to perform its required functions when a fault is encountered. A failure event results from the execution of either random faults (in hardware) or systematic faults (in hardware or software). A safety system is functionally safe if random, systematic and common-cause failures of hardware or software do not lead to a malfunction of the safety system resulting in personnel injury or death, loss of equipment, or environment damage [Ref. 3].

A Random Fault occurs during the lifetime of a hardware element and follows a probability distribution. A random hardware failure is caused by a permanent fault (for example, physical damage), an intermittent fault or a transient fault. Random hardware failure types include single-point failure, CCF, and CMF. Thus, measures reducing the likelihood of random hardware faults are either detection/control of the faults during the lifetime, or a reduction of failure rates.

A Systematic Fault is manifested in a deterministic (non-random) predictable way from a certain cause (fault) that can be eliminated only by a change in the design process, manufacturing process, operational procedures, documentation or other relevant factors. A systematic fault in software is a defect (incorrect step, process or data definition) in the code that causes the program to perform in an incorrect, unintended or unanticipated manner. Thus, measures against systematic faults can reduce systematic failures (for example, implementing and following adequate processes).

Common Cause Failure Mitigation

Developers should mitigate CMFs in the FPGA Input/Output (I/O) banks, clock, configuration memory, reset and power sequencing, and power supply. Mitigations for common cause/mode failures in PLDs include:

- prevention of firmware defects and common triggers (e.g., coding guidelines and partitioning)
- fault detection (e.g., watchdog timers for interfaces and functions)
- system design that forces error conditions to analyzed safe states
- design redundancy, diversity and defense in depth. In the case of logic devices controlling safety-critical electro-explosive devices, common mode failure prevention may require implementation using dissimilar logic devices

Functional Safety Architecture

The objective of the functional safety concept is to derive the functional safety requirements from the safety goals, and to allocate them to the preliminary architectural elements of the item — or to external measures.

A Safety Goal is a top-level safety requirement, such as a Safety Integrity Level or a tolerable system failure rate, that is determined from the results of the hazard analysis and risk assessment. The functional safety requirements are then derived from the safety goals and are allocated to the elements of the preliminary architecture. The functional safety concept should address:

- System Interlocks upstream and downstream from the failure,
- Fault Detection mechanisms (e.g., BIT, Error Detection and Correction (EDAC), readback),
- Fault Tolerance mechanisms (e.g., redundancy, diversity, memory and logic scrubbing, communications integrity), and
- Failure Mitigation mechanisms (e.g., partitioning, transitioning to Safe State).

Safety Interlocks

A safety interlock is a mechanical, electrical or other type of device, the purpose of which is to prevent the operation of hazardous system functions under specified conditions. In the case of a PLD, an upstream safety interlock would prevent the PLD from functioning or producing an output, while a downstream interlock would prevent the device outputs from causing a safety mishap due to the failure. When assessing the degree to which the software/firmware has influence on the safety related aspects of a system, it is imperative to consider whether any other interlocks (both hardware and separate independent software) exist in the system, since system-level interlocks will lower the necessary safety Level of Rigor (LOR).

Fault Detection and Diagnostics

PLDs should incorporate functions to detect and report system faults and failures in a timely manner. Self-diagnostic functions should not adversely affect the ability of the PLD to perform its safety function or cause spurious actuations of the safety function. A typical set of self-diagnostic functions includes [Ref. 8]:

- memory functionality and integrity tests (e.g., ROM checksums and RAM memory tests)
- computer instruction set tests (e.g., calculation tests)
- PLD peripheral hardware tests (e.g., watchdog timer and power tests)
- PLD architecture support hardware tests (e.g., address lines and shared memory interfaces)
- communication link diagnostics (e.g., Cyclic Redundancy Checks (CRC))
- internal signal monitoring (e.g., JTAG)

Memory Integrity

The simplest form of protection used on PLD configuration memory is open-loop scrubbing. This consists of using a separate radiation hardened or tolerant device to continuously overwrite the configuration memory of the PLD with configuration data from a known and reliable source. Another technique is to read back and verify the configuration memory contents by either comparing it to a reliable source or checking the associated CRC signature against the expected value. Once the configuration memory is deemed corrupted, it can be overwritten with data from the reliable source. Error Correcting Code (ECC) and Error Detection and Correction (EDAC) for flash memory and SRAM can reduce the effect of transient faults and permanent faults in integrated volatile and nonvolatile memory.

Redundancy and Diversity

For safety-critical systems, redundancy is essential to operate properly in the event of a failure. Two well-known techniques are dual modular redundancy (DMR) and triple modular redundancy (TMR). In the case of DMR, duplicate designs work in parallel, each processing element receives the same input and a fail-safe engine checks for consistency. If a fault is identified, preventive action must be taken to avoid a failure. TMR creates three duplicate designs and the results of each output are presented to a voting circuit such that the output state that receives the most votes is set. Such a system can withstand the complete failure of one subsystem and allows a supervisor circuit to attempt to fix the fault or alert an operator.

In conjunction with redundancy techniques, design diversity may be employed to further improve reliability and attain a higher SIL. Using this methodology, the parallel designs are not just duplicated (perhaps by using a copy of the same processor or FPGA design) but will perform the same function using a different implementation. For example, an FPGA might be used for one of the designs, while the parallel design might use a processor.

> “Communications that are needed to support a safety function should include provisions for ensuring that received messages are correct and are correctly interpreted. Such communications should employ error-detecting coding along with means for dealing with corrupt, invalid, untimely or otherwise questionable data.”

Communications Integrity

Communications that are needed to support a safety function should include provisions for ensuring that received messages are correct and are correctly interpreted. Such communications should employ error-detecting coding along with means for dealing with corrupt, invalid, untimely or otherwise questionable data. Implement fault-tolerant communication protocols such as:

- end-to-end CRC to detect data corruption
- sequence numbering to detect message repetitions, deletions, insertions and resequencing
- use of an acknowledgement mechanism or time domain multiplexing to detect message delay
- use of sender identification to detect masquerade

Partitioning and Isolation

CMFs are reduced by system-level approaches such as physical and functional partitioning. The preferred partitioning method is to use distinct components with separate electrical paths [Ref. 1]. Safety functions should be separated from non-safety functions so that the non-safety functions cannot prevent the safety system from performing its intended functions. Common clock and power need to be verified to work as expected since these are common resources for the entire device.

Safe State

A “Safe State” is a state in which the system poses an acceptable level of risk for the operational mode. Safe States should be determined by the system-level hazard analysis, and may exist at the system, subsystem and component level. At the system level, a Safe State may be the intended operating mode or a mode

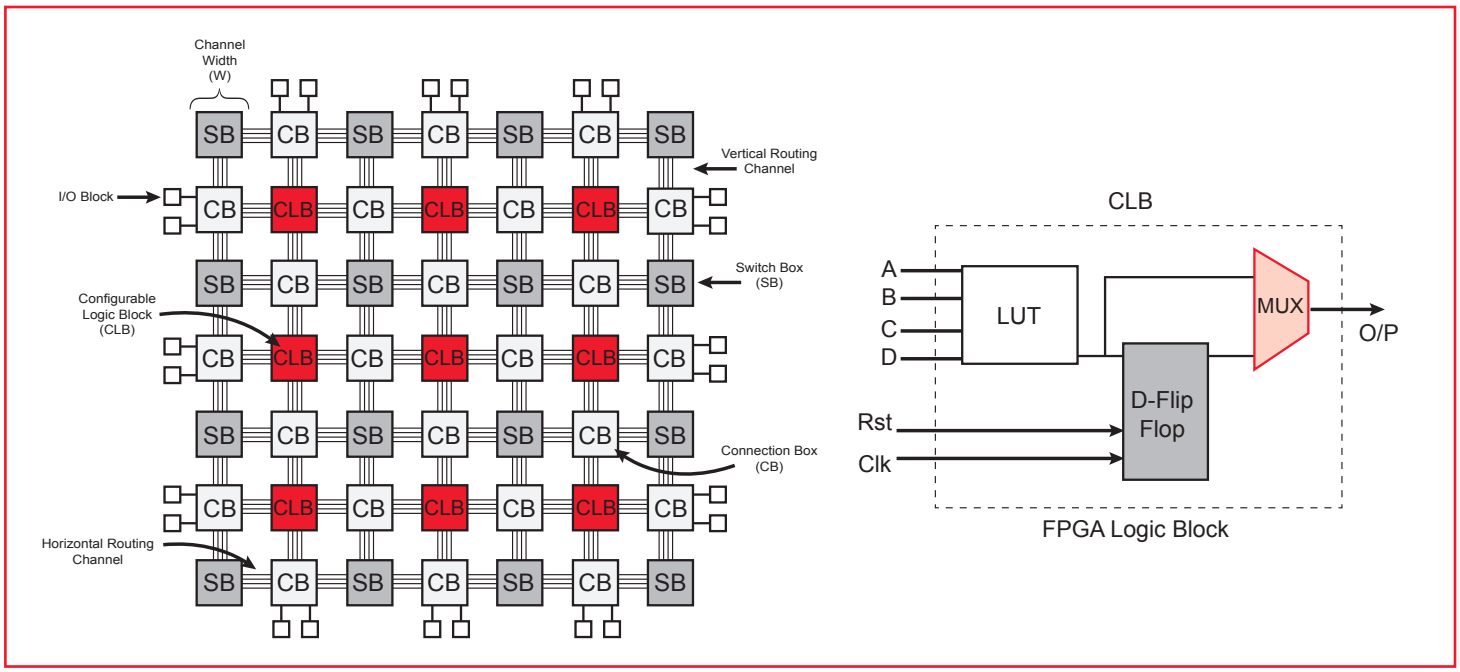

Figure 1 — Typical FPGA Architecture.

in which the system has been disabled. At the subsystem or PLD level, the Safe State may be where the subsystem/device is operating correctly, is explicitly indicating an internal error, is in reset, or is completely unpowered. Logic device state machines should start in a known state and transition through all states as intended in the presence of off-nominal events that could cause faults that interrupt nominal state transitions. Recovery options can include an automated system recovery or an external event (watchdog reset, commanded reset or power cycling), if acceptable to the system design [Ref. 6].

Field Programmable Logic Design

In an FPGA, the computing functionality is provided by its programmable logic blocks, connected to each other through a programmable routing network. This programmable routing network provides routing connections among logic blocks and I/O blocks to implement any user-defined circuit. The routing interconnect of a FPGA consists of wires and programmable switches that form the required connection. These programmable switches are configured using the programmable technology described earlier.

FPGA Development Overview

A configurable logic block (CLB) is a basic component of an FPGA that provides the logic and storage functionality, and will generally consist of look-up tables (LUTs) together with flip-flops and multiplexers for routing purposes. The routing matrix is made up of switchboxes, multiplexers, buffers and programmable interconnect points configured by the corresponding bits in the configuration memory.

A generalized example of an FPGA is shown in Figure 1, where CLBs are arranged in a grid and are interconnected by programmable routing resources. I/O blocks are arranged at the periphery of the grid, and are also connected to the programmable routing interconnect.

In a generalized FPGA design flow, digital designs are described in a hardware description language (HDL) and synthesized into a netlist, which defines the cell instances (Boolean logic gates, flip-flops) and connections needed to implement a given user design. Placement algorithms determine which logic block within an FPGA should implement the corresponding logic block (instance) required by the circuit. The optimization process places connected logic blocks close together to minimize the required wiring and/or to maximize circuit speed. The connections are then routed using programmable interconnects.

Once a netlist is placed and routed on an FPGA, bitstream information is generated for the netlist. This bitstream is programmed on the FPGA using a bitstream loader, and contains the contents of the configurable RAM (CRAM) bits and the initial value of the block RAM (BRAM) bits (i.e., which bit is to be programmed to 0 or to 1). When the bitstream is configured onto the FPGA, data associated with programmable interconnects, look-up tables, control signals and the contents of smaller user-memories (e.g., flip-flops,

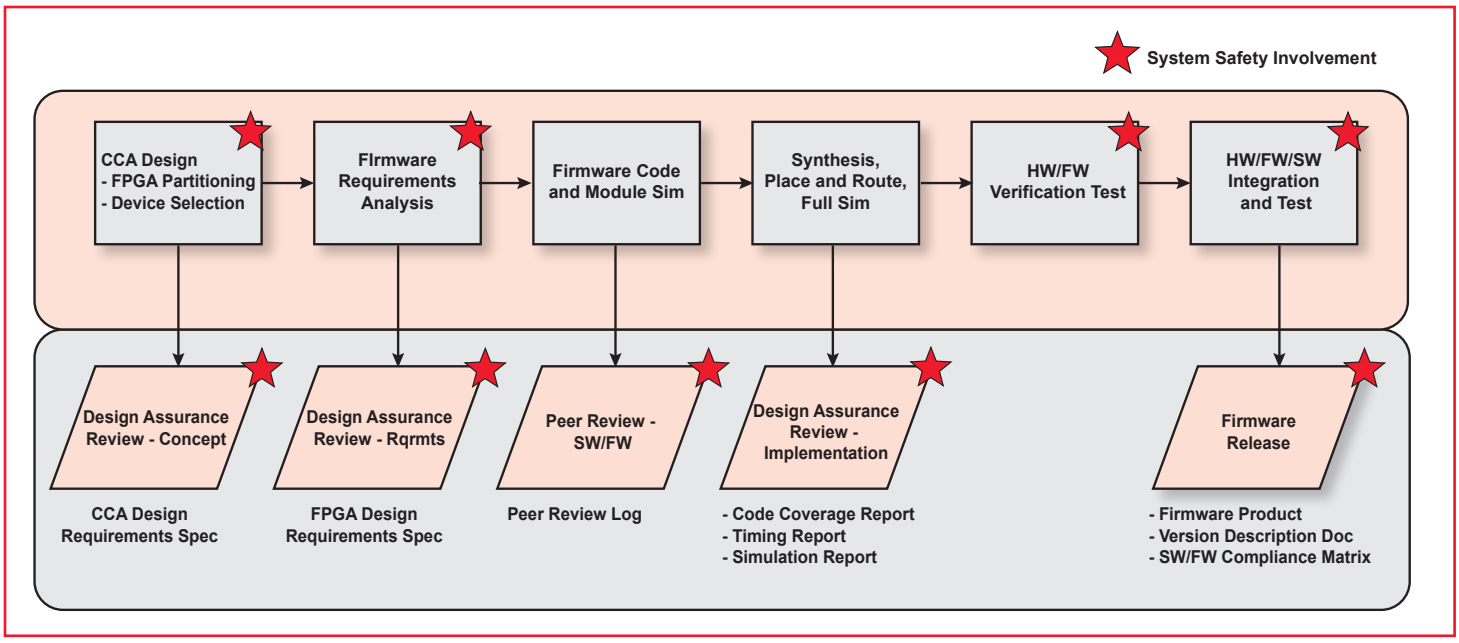

Figure 2 — Example of System Safety Tasks and Artifacts Integrated with Development Process.

LUTRAMs, shift-registers) are stored in CRAM bits. Data associated with the contents of larger blocks of user memory are stored in BRAM bits. The routing information of a netlist is used to correctly program the bits of connection boxes and switch boxes. Once the FPGA is loaded, the design begins to operate on the FPGA fabric.

PLD/Firmware Safety Design and Assessment Process

The following sections identify the safety tasks that should be integrated with the firmware and PLD development process as part of the overall product system safety program.

System Definition and Planning Phase

The System Safety Program Plan (SSPP) should detail how the software safety requirements will be addressed for all safety-significant software (as defined in MIL-STD-882E) within the system under development. Safety-Significant Functions are assessed for their criticality and are architected, designed, coded and verified in accordance with the approved Level of Rigor (LOR). LOR tasks must include the firmware development process tasks, firmware test and verification tasks, and design assurance reviews for each affected SSF. Documents typically include the hardware and firmware design requirements at the CCA and PLD level, as well as safety input to the Software Quality Plan (SQP) and Software Development Plan (SDP), if applicable. The flowchart in Figure 2 shows the suggested integration of system safety activities into a generic digital design process.

Perform System-Level Functional Hazard Analysis

Per the SSPP, a system-level Functional Hazard Analysis (FHA) should be conducted to identify and classify the system functions and the safety consequences of functional failure or malfunction (i.e., hazards). These consequences will be classified in terms of severity to identify the safety-significant functions of the system. The SSFs will be allocated or mapped to the system design architecture in terms of hardware, software/firmware and human interfaces with the system.

The FAA recommends a functional Failure Modes and Effects Analysis (FMEA) to qualitatively assess software and complex devices, and to identify single-point failures, hazardous failure rates, failure effects that prevent achieving safety design requirements, and failure detection provisions. The FMEA should include an interface analysis of interconnections between safety-significant subsystems [Ref. 10].

Identify Safety-Significant PLDs and Firmware

If the SSF is fully or partially implemented by programmable logic devices such as FPGAs, SoCs, etc., each such device should be identified as a safety-significant hardware component that warrants further safety assessment. If the SSF is fully or partially implemented by software/firmware, identify this as safety-significant software/firmware that warrants further safety assessment. Programmable devices and firmware that are not associated with a safety-significant function should require no further testing/analysis beyond the normal firmware development process and qualification testing.

Assess Criticality of PLD Hardware

Each programmable logic device that has control over safety-significant signals or functions should be assigned a severity category and mishap potential based on the hardware FHA. This is used to determine a MIL-STD-882E risk assessment level for the PLD (High, Serious, Medium or Low). Since a single PLD can output multiple safety-significant signals or data, the worst-case risk assigned to the individual outputs should be assigned as the overall risk for that device.

The recommended technology types, based on each PLD risk assessment level, are:

- High Risk: Antifuse technology; EEPROM/Flash technology with System Safety approval
- Serious Risk: Antifuse technology or EEPROM/Flash technology; SRAM technology with System Safety approval
- Medium Risk: EEPROM/Flash; SRAM technology with System Safety approval
- Low Risk: Any technology type is acceptable

NASA and the FAA specifically require that SRAM technology PLDs not be used for safety-critical functions in high-flying aircraft or satellites due to SEE concerns.

Assess Criticality of PLD Firmware

PLD firmware that has control over safety-significant signals or functions should be assigned a Firmware Criticality Index (FwCI) that corresponds to the MIL-STD-882E Software Criticality Index (SwCI) identified in the software FHA. Similar to the software safety process, this will indicate the LOR to be applied to the PLD firmware safety design and assessment.

PLD/Firmware Safety Level of Rigor

This section describes the Programmable Logic Device safety integrity process. For each PLD hardware and firmware configuration item that controls or monitors a safety-significant function, a safety analysis and verification should be performed, based on the LOR that corresponds to the following Firmware Criticality Index (FwCI):

- FwCI 1 — Perform analysis of *requirements*, *architecture*, *design* and *implementation/code*, and conduct in-depth *safety-specific testing*.
- FwCI 2 — Perform analysis of *requirements*, *architecture* and *design*, and conduct in-depth *safety-specific testing*.
- FwCI 3 — Perform analysis of requirements and architecture, and *safety-specific testing*.
- FwCI 4 — Perform analysis of *requirements* and conduct performance *testing*.

Requirements Analysis

Requirements Analysis is the suggested LOR for FwCI 1, 2, 3 and 4. During this phase, the design team conducts requirements definition, logic device technology selection, circuit card considerations, and design input mode selection. System safety identifies and derives firmware and programmable logic safety requirements to safely implement functionality and mitigate identified hazards for both developed and commercial off-the-shelf (COTS) devices in support of the preliminary design. General design requirements to consider include fault tolerance, fault detection, fault isolation, fault annunciation, fault recovery, redundancy, independence, functional partitioning (modules) and physical partitioning (processors). These requirements should be tagged as safety-significant in the program requirements documents, including the CCA and PLD design requirement and Firmware Requirements Specification (FRS). Safety-significant CCA interfaces should be identified in the Interface Requirements Specification (IRS) and/or Interface Control Document (ICD), and marked accordingly.

Specific safety requirements should be levied on each safety-significant PLD, based on the Software Criticality Index (SwCI) of the corresponding Safety-Significant Function (SSF). The following recommended top-level PLD safety requirements are based on best practices and safety standards [Ref. 11]:

- A PLD shall not have autonomous control over Safety-Critical functions.
- Safety-significant PLDs shall have a pre-defined Safe State.
- Safety-significant PLDs shall be configured to power-up and power-down in a known Safe State.
- PLDs that control Safety-Critical functions shall be designed to provide a safe shutdown under power failure/interruption conditions.
- PLD design shall ensure that corrupted Safety-Critical inputs do not result in a Safety-Critical output.
- Safety-significant PLD pins shall be terminated in accordance with manufacturer specifications and data sheets.
- PLDs that control Safety-Critical functions shall employ measures to mitigate corruption or failure of programmable memory.

- PLDs that control Safety-Critical functions shall not be susceptible to Common Mode Failures or Common Cause Failures.
- PLDs that control Safety-Critical functions shall employ measures to mitigate Single Event Effects and similar soft failures.

Architectural Analysis

Architectural Analysis is the suggested LOR for FwCI 1, 2 and 3. During this phase, the design team conducts partitioning trades, preliminary timing analysis, CCA design, and tool suite selection. A top-level design is developed based on the requirements, including functional block diagrams and data flow diagrams that serve as the design architecture. System Safety reviews the architecture for top-level design partitioning to ensure safety, reliability, SEE mitigation, traceability and verifiability. Functional Failure Path (FFP) analysis may be performed to identify safety-significant failure paths of the design, and to analyze how the design architecture can address or mitigate the failure path to meet the target FwCI or Design Assurance Level (DAL). This is preferably accomplished by a combination of FHA and Functional FMEA to the CCA/LRU level. Architectural analysis also seeks instances where the architecture may introduce additional hazards or causal factors.

Detailed Design Analysis

Detailed Design Analysis is the suggested LOR for FwCI 1 and 2. During this phase, initial design capture, simulations and synthesis are performed in preparation for HDL coding that will capture the Register Transfer Level (RTL) design. The RTL will be used by a logic synthesis tool to create the gate-level abstraction of the design that is used for all downstream implementation operations. The design team improves on the results of the architectural analysis by defining specific design details, including state machines, memories, internal and external interfaces, data-path sizes, and clock speeds. Additionally, circuitry for health monitoring, BIT and redundancy are developed. Detailed design includes post-synthesis analysis, place and route, and post-place and route analysis. System safety reviews the CCA and PLD design to confirm that safety requirements are being addressed, and to understand how safety-significant data and functions are handled, including functionality, clock domains, SEE mitigation, and safety versus non-safety separation. CCA interfaces should be evaluated to verify that communication protocols in the design are compatible with interfacing requirements. Potential interface and timing problems include interfaces addressed incorrectly, incorrect input and output timing, incompatibility of data structure, and synchronization across interfaces. Each software/firmware hazard is reviewed to determine if design choices have affected currently documented mitigations.

Implementation/Code Analysis

Implementation/Code Analysis is the suggested LOR for FwCI 1 (and possibly FwCI 2, depending upon the customer). During the design implementation phase, design capture — such as coding in a hardware description language (HDL) — is completed. The design team performs final design simulation, synthesis, place and route, and timing analyses. The design may be tested on a development board and/or hardware resembling the final system. System safety participates in code reviews to verify the integrity of data variables while ensuring that the firmware exhibits strong data typing for all safety-critical variables. Safety must also ensure that intended safety or reliability features were not compromised by the synthesis tool optimization process. Finally, safety should analyze the error-handling implementation associated with internal processing and interface message traffic to identify potential failure modes that would result in safety-critical data erroneously used or transferred.

Verification and Validation

Verification and Validation (V&V) processes are used to confirm that the development products of an activity conform to the requirements of that activity, and that the system performs according to its intended use and user needs. V&V processes address programmable logic as it affects software and system integration of the digital system components, and the interaction of the resulting programmable logic in the system. The V&V activities and tasks include system testing of the final integrated hardware, software/firmware, and interfaces. Independent verification and validation may be required for safety-significant systems.

Performance Verification and Validation

This is the recommended LOR for FwCI 1 through 4. System safety reviews the performance verification test results for failures and anomalies that may have safety implications.

Safety-Specific Testing

Safety-Specific Testing is the recommended LOR for FwCI 1, 2 and 3. The PLD test cases and procedures

should be reviewed to ensure that testing verifies safety requirements in the context of their intended or expected functionality. Specific safety requirements included in the requirements documents should be verified. The PLD functions necessary to perform safety functions, as well as those functions whose operation or failure could impair safety functions, should be exercised during testing. This includes, as appropriate and practicable, exercising and monitoring the memory, the logic, inputs and outputs, display functions, diagnostics, associated components, communication paths, and interfaces. System safety should confirm that each software, firmware and/or hardware requirement that could impact safety has successfully passed V&V in accordance with the assigned LOR. The results of safety tests should be documented, along with the implementation of mitigations, and the identification of any safety anomalies encountered during test.

Common Mode Failure Testing is the recommended LOR for FwCI 1 and 2. This testing should be considered to demonstrate that a safety-significant PLD is not susceptible to common cause failures and common mode failures. The FAA best practice for safety-significant design is to target full structural coverage for Statements, Branches, Conditions, Expressions and State Machines [Ref. 4]. The Nuclear Regulatory Commission (NRC) considers a PLD integrated with representative hardware to be immune from CCFs if it is tested for correctness of outputs in the following cases [Ref. 8]:

- testing every combination of inputs
- testing the operational range of analog inputs for PLDs that include analog inputs
- testing every executable logic path (this includes non-sequential logic paths)
- testing every functional state transition

Fault Insertion Testing is the recommended LOR for FwCI 1 and 2. Fault insertion and failure mode testing should be performed on safety-significant PLD inputs and outputs to verify that the device responds correctly and safely during power-up, power-down, and brown-out transients [Ref. 7]. If applicable, fault insertion should include out-of-sequence testing and corrupted signal input testing.

About the Author

Martin S. Chizek, PE, CSP, is a product safety officer at Lockheed Martin in Orlando, Florida. He holds a Bachelor of Science in engineering from Wichita State University, a Master of Science in systems engineering and management from the University of Southern California, and a Juris Doctor from the University of Memphis.

References

1. U.S. Department of Defense Fuze Engineering Standardization Working Group (FESWG). JOTP-051, "Technical Manual for the Use of Logic Devices in Safety Features," February 2012.
2. U.S. Department of Defense. Military Standard MIL-STD-882E, "Standard Practice for System Safety," May 11, 2012.
3. International Electrotechnical Commission. IEC 61508 Edition 2, "Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems," April 2010.
4. Radio Technical Commission for Aeronautics. RTCA/DO-254, "Design Assurance Guidance for Airborne Electronic Hardware," April 2000.
5. National Aeronautics and Space Administration. NASA-HDBK-8739.23A, "Complex Electronics Handbook for Assurance Professionals," February 2016.
6. National Aeronautics and Space Administration. NASA-STD-8719.13C, "NASA Software Safety Standard," July 2013.
7. National Aeronautics and Space Administration. NASA-HDBK-4008, "Programmable Logic Devices (PLD) Handbook," December 2013.
8. Institute of Electrical and Electronics Engineers. IEEE Standard 7-4.3.2, "Criteria for Programmable Digital Devices in Safety Systems of Nuclear Power Generating Stations," January 2016.
9. U.S. Department of Transportation, Federal Aviation Administration. DOT/FAA/TC-15/62, "Single Event Effects Mitigation Techniques Report," February 2016.
10. SAE International. SAE ARP 4761, "Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment," December 1996.
11. Isaac, Todd. "Programmable Logic Device (PLD) / Firmware Safety," Seminar at the 35th International System Safety Conference, 2017.