UL 1998

STANDARD FOR SAFETY

Software in Programmable Components

UL Standard for Safety for Software in Programmable Components, UL 1998

Third Edition, Dated December 18, 2013

Summary of Topics

This revision of ANSI/UL 1998 dated November 16, 2022 includes the following:

- ***Removal of limitation to non-networked software; 1.5 and 1.7***
- ***Update of definitions; 2.27 and Appendix B***
- ***Clarification of risk analysis scope and requirements; 3.1, 3.5 and 3.6***
- ***Clarification of software development process requirements; 2.52A, 4.1, 4.6 and 4.7***
- ***Clarification of tool requirements; 5.1***
- ***Clarification of software design requirements; 6.8 – 6.10 and 12.2.3***
- ***Clarification of measures to address systematic microelectronic hardware failures; 8.4***
- ***Clarification of change management and document control requirements; 14.4 and 14.5***

Text that has been changed in any manner or impacted by UL's electronic publishing system is marked with a vertical line in the margin.

The new and revised requirements are substantially in accordance with Proposal(s) on this subject dated August 8, 2022 and October 7, 2022.

No Text on This Page

ANSI/UL 1998-2022

UL 1998

Standard for Software in Programmable Components

First Edition – January, 1994
Second Edition – May, 1998

Third Edition

December 18, 2013

This ANSI/UL Standard for Safety consists of the Third Edition including revisions through November 16, 2022.

The most recent designation of ANSI/UL 1998 as an American National Standard (ANSI) occurred on November 16, 2022. ANSI approval for a standard does not include the Cover Page, Transmittal Pages, and Title Page.

Comments or proposals for revisions on any part of the Standard may be submitted to UL at any time. Proposals should be submitted via a Proposal Request in UL's On-Line Collaborative Standards Development System (CSDS) at https://csds.ul.com.

No Text on This Page

CONTENTS

No Text on This Page

1 Scope

1.1 These requirements apply to non-networked embedded software residing in programmable components performing safety-related functions whose failure is capable of resulting in a risk of fire, electric shock, or injury to persons.

1.2 This is a reference standard in which the requirements are to be applied when specifically referenced by other standards or product safety requirements.

1.3 These requirements address the risks unique to product hardware controlled by software in programmable components.

1.4 These requirements are intended to supplement applicable product or component standards and requirements, and are not intended to serve as the sole basis for investigating the risk of fire, electric shock, or injury to persons.

1.5 These requirements are intended to address risks that occur systematically or randomly in the software or in the process used to develop and maintain the software, such as the following:

a) Requirements conversion faults that cause differences between the specification for the programmable component and the software design;

b) Design faults such as incorrect software algorithms or interfaces;

c) Coding faults, including syntax, incorrect signs, endless loops, and other coding faults;

d) Timing faults that cause program execution to occur prematurely or late;

e) Microelectronic memory faults, such as memory failure, not enough memory, or memory overlap;

f) Induced faults caused by microelectronic hardware failure;

g) Latent, user, input/output, range, and other faults that are only detectable when a given state occurs;

h) Failure of the programmable component to perform any function at all; and

i) Failures in data communication such as transmission errors, repetitions, deletion, insertion, resequencing, corruption, delay and masquerade.

1.6 Product standard requirements may amend or supersede the requirements in this standard, as appropriate.

1.7 These requirements are not intended to address risk of remote software update and cybersecurity. Risks associated with unauthorized access or attack through a network shall be addressed in the product standard or other referenced standards such as the Standard for Remote Software Updates, UL 5500 and the series of Standards for Software Cybersecurity for Network-Connectable Products, UL 2900.

2 Definitions of Terms Used

2.1 For the purpose of this standard, the following definitions apply.

2.2 APPLICATION-SPECIFIC INTEGRATED CIRCUIT (ASIC) – An electronic device comprised of many transistors and other semiconductor components which integrate standard cells and arrays from a library into one piece of silicon intended for a particular use.

2.3 BUILT-IN TEST – A design method that allows a product to test itself by adding logic for test signal generation and analysis of test results.

2.4 CENTRAL PROCESSING UNIT (CPU) – The unit of a computing and controlling system that includes the circuits controlling the interpretation of instructions and their execution.

2.5 CRITICAL SECTION – A segment of the software that is intended to perform the functions that address or control risks.

2.6 DATA – A representation of facts, concepts, or instructions in a manner suitable for storage, communication, interpretation, or processing.

2.7 DESIGN – The process of defining the software architecture, components, modules, interfaces, test approach, and data for a software system to satisfy specified requirements.

2.8 ELECTRONICALLY ERASABLE PROGRAMMABLE READ ONLY MEMORY (EEPROM) – A reprogrammable read-only memory in which cells may be erased electrically and in which each cell is capable of being reprogrammed electrically.

2.9 EMBEDDED SOFTWARE – Software that is physically part of a product and whose primary purpose is to maintain some property or relationship between other components of the product in order to achieve the overall system objective.

2.10 ERASABLE PROGRAMMABLE READ ONLY MEMORY (EPROM) – A type of programmable memory device which can only be read and not altered under normal use. The memory is capable of being erased by ultraviolet light and reprogrammed.

2.11 ERROR – A discrepancy between a computed, observed, or measured value or condition and the true, specified, or theoretically correct value or condition.

2.12 FAIL-OPERATIONAL PROCEDURE – A procedure executed in the event that a failure has occurred which continues product operation but provides degraded performance or reduced functional capabilities.

2.13 FAIL-SAFE PROCEDURE – A procedure executed to maintain the Risks Addressed (RA) state of a product while transitioning into a non-operational mode.

2.14 FAILURE – The inability of a product or component to perform its specified function.

2.15 FAILURE MODE – The physical or functional manifestation of a failure.

2.16 FAILURE MODE TEST – A suite of tests that have been specifically developed based upon the failure modes that exist in a programmable component or product.

2.17 FAULT – A deficiency in a product or component which is capable of, under some operational conditions, contributing to a failure.

2.18 FAULT-TOLERANT – The capability of software to provide continued correct execution in the presence of a defined set of microelectronic hardware and software faults.

2.19 FLASH MEMORY – A type of non-volatile memory which is capable of being erased electrically and reprogrammed, but only in blocks, as opposed to one byte increments.

2.20 HAZARD – A potential source of physical injury to persons.

2.21 INSTRUCTION – A statement that specifies an operation to be performed and that is capable of identifying data involved in the operation.

2.22 INTEGRITY – The degree to which a system or component prevents unauthorized access to, or modification of, computer programs or data.

2.23 MICROCONTROLLER – A microcomputer chip capable of executing instructions.

2.24 MICROELECTRONICS – Monolithic, hybrid, or module circuits, where the internal connections are not accessible, which satisfy one or more of the following criteria:

a) More than 1000 gates are used in digital mode;

b) More than 24 functionally different external electrical connections are available for use; or

c) The functions can be reprogrammed.

The circuits are capable of functioning in the analogue mode, digital mode, or a combination of the two modes. Examples of microelectronics include: ASICs, ROMs, RAMs, PROMs, EPROMs, PALs, and PLDs.

2.25 NEGATIVE CONDITION BRANCH – A code construct implementing an alternate path of program control flow intended to provide a deterministic path from the normal operating path to the Risks Addressed (RA) state whenever the normal operating path cannot be taken.

2.26 NON-DETERMINISM – The state in which output is not predictable in advance because data dependencies effect task execution order.

2.27 NON-NETWORKED EMBEDDED SOFTWARE – Embedded software that executes safety related functionality on a single microprocessor/microcontroller or on microprocessors/microcontrollers residing in the same physical enclosure, which does not depend on data received or transmitted outside of the enclosure.

2.28 NON-TERMINATING STATE – Any state of the software characterized by the execution of instructions that do not permit exiting from that state.

2.29 NON-VOLATILE MEMORY – A storage device not alterable by the interruption of the power to the memory, for example; ROM, FLASH, PROM, EPROM, and EEPROM.

2.30 OFF-THE-SHELF (OTS) SOFTWARE – Software which is made available on the market and intended for broad distribution, generally without the need for additional tailoring, including, but not limited to, operating system software, runtime libraries, real-time executives, kernel primitives, shareable re-entrant library-type routines, and the like. This includes software that has been developed by another developer.

2.31 OPERATIONAL TESTING – Evaluations on a product or component using an input profile representative of its operational environment.

2.32 PARAMETER SETTINGS – A finite collection of values assigned to variables to select, enable or disable known pre-existing function(s) or features of the software.

2.33 PARTITIONING – The act of segregating the functions of a system into verifiably distinct and protected collections of functions.

2.34 POST-RELEASE TESTING – All testing for changes implemented after the final production software is released.

2.35 PROCEDURE – A course of action taken to perform a task.

2.36 PROCESS – A sequence of steps performed to produce a given result.

2.37 PRODUCT – An instrument, apparatus, implement, or machine intended for personal, household, industrial, laboratory, office, or transportation use.

2.38 PRODUCT HARDWARE – Any hardware that is part of a product and that provides electrical, mechanical, and (or) electromechanical functions.

2.39 PROGRAMMABLE COMPONENT – Any microelectronic hardware that can be programmed in the design center, the factory, or in the field. Here the term "programmable" is taken to be "any manner in which one can alter the software wherein the behavior of the component can be altered."

2.40 PROGRAMMABLE COMPONENT CONFIGURATION – The configuration of the microelectronic hardware and software that must be present and functioning for the software to operate and control the equipment as intended. This configuration includes, but is not limited to, the operating system or executive software, communication software, microcontroller, network, input/output hardware, any generic software libraries, database management and user interface software.

2.41 PROGRAMMABLE READ-ONLY MEMORY (PROM) – A memory chip whose contents are capable of being programmed by a user or manufacturer for a specific purpose.

2.42 RANDOM ACCESS MEMORY (RAM) – Data storage where the time required for data access is independent of the location of the data most recently obtained or placed in storage.

2.43 READ ONLY MEMORY (ROM) – Data storage that stores data not alterable by computer instructions.

2.44 RISK – The potential for fire, electric shock, or injury to persons associated with the intended use of the product as specified by the product safety requirements.

2.45 RISKS ADDRESSED (RA) STATE – A state that is characterized by all reasonably foreseeable risks associated with the intended use of the product being addressed such that there is no longer a likelihood of the risk.

2.46 SAFETY-RELATED FUNCTIONS – Control, protection, and monitoring functions which are intended to reduce the risk of fire, electric shock, or injury to persons.

2.47 SOFTWARE – Computer programs, procedures, and data pertaining to the operation of a programmable component that provides safety-related functions as follows:

a) Exercises direct control over the state of microelectronic or product hardware. When not performed, performed out of sequence, or performed incorrectly, such programs, procedures, and data are capable of resulting in a risk.

b) Monitors the state of microelectronic or product hardware. When not performed, performed out of sequence, or performed incorrectly, such programs, procedures, and data provide data that is capable of resulting in a risk.

c) Exercises direct control over the state of the microelectronic or product hardware. When not performed, performed out of sequence, or performed incorrectly, such programs, procedures, and data are capable of, in conjunction with other human actions, product hardware or environmental failure, resulting in a risk.

2.48 SOFTWARE CODE – Computer instructions and data definitions expressed in a programming language or in a form output by an assembler, compiler, or other translator.

2.49 SOFTWARE DESIGN – The process of defining the software architecture, components, modules, interfaces, test approach, and data for a software implementation to satisfy specified requirements.

2.50 STATE – The values assumed, at a given instant in time, by a collection of variables that define certain specified characteristics of the product or its components.

2.51 STRESS TESTING – Testing conducted to evaluate a product or component at or beyond the limits of its specified requirements.

2.52 SUPERVISORY SECTION – The main section of software that controls the execution of the critical and non-critical sections of the software (e.g., the operating system of a microprocessor).

2.52A SYSTEMATIC FAULT – Type of faults which occurs when a certain set of particular input conditions causes a process or product to react in a certain manner. Typical examples are specification errors, design errors and software coding errors.

2.53 TEST CONFIGURATION – The detailed identification and layout of the test and measurement equipment used for testing. The test configuration is often documented directly in the test plan or the test procedures and specifies all special test fixtures, supplementary software, software tools, and data files along with a description of the test set-up and any preparation activities.

2.54 TEST CRITERIA – Objectives that a product or component must meet in order to comply with a given test.

2.55 TEST PARAMETERS – Variables used to alter and define a test.

2.56 TEST PLAN – A document describing the scope, approach, resources, and schedule of intended test activities. It defines test items, the features to be tested, the testing tasks, who will do each task, and any risks requiring contingency planning.

2.57 TEST PROCEDURES – Detailed instructions for the set-up, execution, and evaluation of results for a given test case.

2.58 TOOL – Any equipment (e.g., logic analyzers, oscilloscopes, multimeters, digital and analog computers), devices, or software programs [e.g., simulators, computer-aided software/systems engineering (CASE) tools, compilers, type checkers, static analyzers, automated testing scripts, debuggers, linkers, loaders, assemblers, code generators, code librarians, editors, and software analyzers] used to automate or partially automate software development activities, including design, implementation, and testing.

2.59 TOOL QUALIFICATION – The acceptance of analysis, testing, and resulting evidence for which confidence is obtained regarding the correctness of a tool's outputs.

2.60 TOOL VENDOR – The organization responsible for providing the tool.

2.61 UNIQUE IDENTIFIER – A value that distinctively characterizes each individual version or revision of a manufacturer's critical and supervisory sections of software.

2.62 USER – A qualified service person or an operator of the programmable component.

2.63 VALIDATION – The test and evaluation of the integrated computer system (hardware and software) and its specification to determine whether it carries out its intended purpose.

2.64 VERIFICATION – The process of determining whether the products of a given development phase are correct and consistent with the products and standards provided as input to that phase.

3 Risk Analysis

3.1 A risk analysis shall be conducted to determine:

a) The set of risks (for scope of risk analysis see Section 1, Scope); and

b) That the software addresses the identified risks.

3.2 The risk analysis shall be based on the safety requirements for the programmable component.

3.3 An analysis shall be conducted to identify the critical, non-critical, and supervisory sections of the software.

3.4 An analysis shall be conducted to identify states or transitions that are capable of resulting in a risk.

3.5 The risk analysis shall follow an easily readable approach with decomposition into small comprehensible parts which facilitates verification and traceability (see 4.3).

3.6 Each identified risk shall:

a) Have documented evidence to demonstrate the risk has an effective design measure; and

b) Have documentation showing assurance evidence (e.g. executed test cases, simulation results, or qualification data) to support coverage through risk mitigation and/or other relevant control measures.

4 Process Definition

4.1 All software process activities through the product lifecycle shall be described (see Section 12, Documentation).

4.2 The software process activities shall be identified with distinct entry points, exit points, and criteria for transitioning among activities.

4.3 The software process shall include a risk analysis in accordance with the requirements of Section 3, Risk Analysis. The risk analysis shall be traceable to the Programmable Component and Software Requirements Specification (see 12.6).

4.4 Criteria for transitioning among activities shall include consideration of the safety-related requirements for the programmable component.

4.5 Work products (e.g., meeting minutes, analysis and test results, formal documentation, etc.) shall be identified and associated with software process activities.

4.6 All software process activities shall support the communication of issues (for example any types of errors, ambiguous or contradicting requirements, non-testable requirements, any types of gaps and omissions, neglected or overseen hazards and risks, any kind of systematic faults not considered) that could impact the safety-related functioning of the programmable component.

4.7 Safety-related requirements for the programmable component shall be traceable forward and backward throughout the software process activities and shall be documented as evidence supporting compliance with this standard.

4.8 The verification, validation, and testing activities in the software process shall address errors at their source.

5 Qualification of Design, Implementation, and Verification Tools

5.1 Evidence of tool qualification shall be provided for all tools used in the design, implementation, and verification of software in the programmable component using at least one of the following forms:

a) Documentation attesting to supporting measurement tool calibration and verification and validation activities of software design, implementation, analysis, verification and process support tools in accordance with 6.7, 11.1.1, 11.2.1, 11.2.4, 12.1, 12.2, 12.4, 12.8 and Sections 14, and 15 of this standard.

b) Evidence that the tool(s) has/have met formally defined requirements by a third-party tool certification program covering as minimum the requirements of 5.1(a).

5.2 When available from the tool vendor or other sources (e.g., the user community), the manufacturer shall provide a list of known bugs for the precise revision/version of the tool that the manufacturer intends to use to develop software. For each identified error in the known bug report for the tool, the following evidence shall be provided when implementing 5.1(a).

a) The feature that leads to an error has been fixed, tested, and approved for distribution by the tool vendor in a new release that has been incorporated into the manufacturer's version of the tool, or

b) The feature that leads to an error has not been used by the manufacturer in the development of safety-related software and does not lead to a risk.

6 Software Design

6.1 A fault in the software shall not initiate an event that results in a risk.

6.2 The software shall maintain an RA state upon detection of a condition that is capable of resulting in a risk as identified in Section 3, Risk Analysis.

6.3 Detection of a failure in the software during the intended operation of the product shall be handled in a manner that is in accordance with the product safety requirements.

6.4 The software shall employ means to identify and respond to states that are capable of resulting in a risk. Examples of such means include initialization, fail-safe and fault-tolerant concepts, run-time checks, and built-in tests.

6.5 In allocating resources to tasks, consideration shall be given to the scheduling frequency of the task, the criticality of the task, and the resources utilized by the task, as well as the impact that each of these factors has on the ability to address the identified risks.

6.6 Means shall be employed for the prevention, detection, and resolution of non-terminating and non-deterministic states and error states such as division by zero, under/overvoltage condition of internal/external power supplies to the programmable components, and under/overflow that are capable of affecting the intended operation of the software.

6.7 All variables shall be set to initial values before being used by any instruction. If a design, implementation, or verification tool is responsible for ensuring that variables are set to initial values, the tool's qualification shall include verification of this (see Section 5).

6.8 The software shall fulfill the specified requirements for the safety related software.

6.9 The software design shall avoid unnecessary complexity and shall allow verification by static analysis.

6.10 Manufacturers shall develop their own company-specific Coding Standards/Guidelines that specify:

a) Which software constructs and practices should be avoided in general (and why),

b) Conditions under which certain constructs and tactics not normally recommended may be employed, and

c) Coding practices that are recommended because they are known to reduce the likelihood of risk occurring and/or they increase the ability to systematically detect conditions that can lead to risk events such as serious defects.

7 Critical and Supervisory Sections of Software

7.1 The software shall be initialized to a documented RA state.

7.2 All critical and supervisory sections of the software shall be partitioned from other critical and supervisory sections of the software, as well as from non-critical sections.

7.3 When it is not possible to partition the critical and supervisory sections from the non-critical sections of the software as in 7.2, all of the software shall be considered critical or supervisory.

7.4 Means shall be employed to avoid, or detect and recover from, memory usage and addressing conflicts.

7.5 The supervisory section shall maintain control of the execution of the software during the operation of the programmable component.

7.6 Software shall initiate a fail-safe or a fail-operational procedure in the event that a failure in a critical or supervisory section is detected.

7.7 When the initiation of a safety-related function is capable of resulting in a risk, a minimum of two instruction sequences shall be employed to verify the initiation of the safety-related function, unless otherwise specified in the product safety requirements.

7.8 There shall be provisions to control the accessibility of instructions and data dedicated to critical and supervisory section functions.

7.9 There shall be provisions to protect instructions and data for critical and supervisory sections of software from being affected by any function except critical and supervisory section functions.

7.10 Supervisory and critical sections of software shall be resident in non-volatile memory. If supervisory and critical sections of software are transferred to volatile memory during operation, risks associated with the transfer and execution of software from volatile memory shall be considered in Section 3, Risk Analysis.

7.11 Means shall be employed to preserve the integrity of data used by critical and supervisory sections of software.

7.12 Fixed or one-time changing data used for critical and supervisory sections of software shall reside in non-volatile memory. If fixed or one-time changing data used for critical and supervisory sections of software are transferred to volatile memory during operation, risks associated with the transfer and access of this data from volatile memory shall be considered in Section 3, Risk Analysis.

8 Measures To Address Microelectronic Hardware Failure Modes

8.1 Means shall be employed to address all microelectronic hardware failure modes identified by Section 3, Risk Analysis. Appendix A contains examples of acceptable measures for microelectronic hardware.

8.2 Physical failures of the following microelectronic hardware shall be considered:

a) CPU registers, instruction decoding and execution, program counter, addressing and data paths;

b) Interrupt handling and execution;

c) Clock;

d) Non-volatile and volatile memory and memory addressing;

e) Internal data path and data addressing;

f) External communication and data, addressing, and timing;

g) Input/output devices such as analog I/O, D/A and A/D converters, and analog multiplexers;

h) Monitoring devices and comparitors; and

i) Application-Specific Integrated Circuits (ASICs), Gate Array Logics (GALs), Programmable Logic Arrays (PLAs), and Programmable Gate Arrays (PGAs) hardware.

8.3 Analysis of possible combinations of microelectronic hardware failures, software faults, and other events that are capable of resulting in a risk shall be conducted. This includes, for example, microelectronic hardware failures that cause software faults that are capable of resulting in a risk.

8.4 When available from the programmable component vendor or other sources (e.g. the user community), the manufacturer shall provide erratum for the precise revision/version of the programmable component that the manufacturer intends to use. For each identified error in the erratum, the following evidence shall be provided:

a) The error has been fixed, tested, and approved for distribution by the programmable component vendor in a new release that has been incorporated into the manufacturer’s version of the programmable component, or

b) Feature(s) affected by the error have not been used by the manufacturer in the development of safety-related software.

c) Appropriate software related documentation shall be updated to avoid risks related to changes of either the software or the programmable component.

9 Product Interface

9.1 For power interruptions of any duration, the software shall maintain a documented RA state.

9.2 When initialization is allocated as a software function, the software shall initialize the product to a documented RA state.

9.3 Upon any situation in which the software terminates, the product shall maintain a documented RA state.

9.4 A procedure or instruction intended to halt the programmable component shall maintain an RA state of the product.

10 User Interfaces

10.1 The requirements in this section only apply to software that accepts user input, unless otherwise specified in the product safety requirements.

10.2 The time limits and other parameters of the software shall not be changeable by a user such that the intended execution of critical and supervisory sections of software is adversely affected.

10.3 The time limits and other parameters of the software that are intended to be configured by qualified service personnel shall be prevented from being changed to the extent that the intended operation of the critical or supervisory sections of software is adversely affected.

10.4 The software shall require two or more user responses to initiate an operation that is capable of resulting in a risk.

10.5 Input commands which are capable of resulting in a risk when executed, shall not be initiated without operator intervention when those commands are received from an external source.

10.6 For a system as described in 10.5, no operation that is capable of resulting in a risk shall be initiated by a single user input.

10.7 Incorrect input shall not adversely affect execution of critical sections of software.

10.8 If required to do so by the product safety requirements and when determined by the Risk Analysis, the software shall provide for user cancellation of the current operation and return the programmable component to an RA state.

10.9 User cancellation of the current operation shall require a single input by the user.

10.10 Cancellation of processing shall leave the software in an RA state.

11 Software Analysis and Testing

11.1 Software analysis

11.1.1 Software design and code analysis shall be conducted to evaluate that the critical and supervisory sections of software only perform those functions which they are intended to perform and do not result in a risk.

11.1.2 The software design and code analysis shall be conducted to demonstrate:

a) Correctness and completeness with respect to the safety requirements for the programmable component;

b) Coverage of each branching condition and function evaluation that addresses and remediates risks associated with abnormal operations, or involves a risk associated with its normal operation;

c) That fail-safe and fail-operational procedures bring the product to an RA state. See 6.3, 6.4, and 7.6;

d) That the scheduling requirements are met and safety-related functions meet the timing constraints specified by the safety requirements for the programmable component. See 6.5;

e) The integrity of the partitions between supervisory, critical, and non-critical sections of software. See 7.2;

f) That partition violations caused by such occurrences as data handling errors, control errors, timing errors, and misuse of resources do not occur; and

g) Consistency in the data and control flows across interfaces.

11.2 Software testing

11.2.1 Software testing shall include development and post-release testing.

11.2.2 Tests of the software shall be conducted and test results documented to evaluate that the software only performs those functions for which it is intended and does not result in a risk.

11.2.3 Test cases shall be developed based on the risk analysis, the documented descriptions of the software operation and safety features (see 12.7.2), and the software analysis. See 11.1.

11.2.4 Tests shall be conducted to demonstrate:

a) Correctness and completeness with respect to the safety requirements for the programmable component;

b) Coverage of each decision and function that is capable of involving a risk;

c) That fail-safe and fail-operational procedures bring the product to an RA state. See 6.3, 6.4, and 7.6;

d) That the scheduling requirements are met and safety-related functions meet the timing constraints specified by the safety requirements for the programmable component. See 6.5;

e) The integrity of the partitions between supervisory, critical, and non-critical sections of software. See 7.2;

f) That partition violations caused by such occurrences as data handling errors, control errors, timing errors, and misuse of resources do not occur; and

g) Consistency in the data and control flows across interfaces.

11.2.5 The outputs that the software generates to control product hardware shall be tested to determine their effects on the product hardware, based on the expected output.

11.3 Failure mode and stress testing

11.3.1 In addition to testing under normal usage, failure mode tests and stress tests shall be conducted.

11.3.2 Failure mode and stress testing shall include consideration of the following:

a) Operator errors that are capable of adversely affecting the intended operation or the control of the programmable component;

b) Microelectronic hardware component faults;

c) Errors in data received from external sensors or other software processes;

d) Failures associated with the entry into, and execution of, critical and supervisory sections of software;

e) Failures, errors, and other abnormal conditions associated with decisions and functions that are capable of providing risk reduction, including negative condition branches; and

f) Other processes and procedures that are capable of adversely affecting the intended operation of the software.

11.3.3 Test cases shall include the following, as determined in accordance with 11.1.2(b):

a) Out-of-range;

b) Boundary condition; and

c) Type mismatched values for parameters at which decisions are made.

11.3.4 Failure mode tests shall address all foreseeable faults identified in Risk Analysis, Section 3.

12 Documentation

12.1 User documentation

12.1.1 Except for embedded software that has no direct user interaction, user documentation (e.g, manual, guide, or other documents) shall be prepared.

12.1.2 The user documentation shall describe the required data and control inputs, input sequences, options, program limitations and other activities or items necessary for intended execution of the software.

12.1.3 All error messages shall be identified and corrective actions described in the user documentation.

12.2 Software plan

12.2.1 A software plan shall be documented, which describes the software development activities.

12.2.2 The software plan shall include a description of the software design methodology, development rationale, any metrics to be collected, applicable standards and the engineering methods/techniques employed, and an itemized list of all documents produced throughout the software process.

12.2.3 The software plan shall ensure that design methodologies support readability, understandability, testability and maintainability of the software.

12.3 Risk analysis approach and results

12.3.1 The risk analysis approach and results (see Section 3) shall be documented.

12.3.2 The risk analysis shall illustrate how events, or logical combinations of events, are capable of leading to an identified hazard.

12.3.3 The risk analysis shall list all identified risks associated with the product.

12.4 Configuration management plan

12.4.1 A configuration management plan, which applies to off-the-shelf software, software tools, and the manufacturer-provided software, shall be documented.

12.4.2 The configuration management plan shall describe:

a) How changes to the software and hardware are managed;

b) The initiation, transmittal, review, disposition, implementation and tracking of discrepancy reports and change requests; and

c) The methods and activities employed to formally control receipt, storage and backup, handling, and release of software and all documentation identified in this section.

12.5 Programmable system architecture

12.5.1 The programmable system architecture shall be documented.

12.5.2 The programmable system architecture shall describe the programmable component, including interfaces to users, sensors, actuators, displays, microelectronic hardware architecture, top-level software architecture, the mapping of the software to the hardware, and block diagrams of the programmable system showing a high-level view of the product architecture.

12.5.3 The programmable system architecture shall describe the software-to-software interfaces and the hardware-to-software interfaces.

12.5.4 The configuration(s) of the programmable component(s) that the software is intended to be operated with shall be identified. See 2.39.

12.6 Programmable component and software requirements specification

12.6.1 A programmable component and software requirements specification shall be documented.

12.6.2 The programmable component and software requirements specification shall describe functional, performance, and interface requirements of the programmable system and the software.

12.6.3 The specification shall include a description of all modes of operation, identification of failure behavior, and required responses.

12.6.4 The programmable system and software requirements specification shall be traceable to the risk analysis results documented in 12.3.

12.7 Software design documentation

12.7.1 Software design documentation shall be prepared.

12.7.2 The software design documentation shall include a description of the operation and safety features of the software, with respect to the intended function.

12.7.3 The software design documentation shall include the inputs and outputs, functions, data descriptions and relationships, control and data flow, fault handling, and algorithms.

12.7.4 The software design documentation shall describe details of how the design of the software meets the system and software requirements specification.

12.8 Analysis and test documentation

12.8.1 All analysis and test methods and results shall be documented.

12.8.2 A test plan shall be documented which covers all software that is used in the programmable component, including off-the-shelf and third-party supplied software (See Off-The-Shelf Software, Section 13).

12.8.3 The test plan shall include or reference the documented test procedures which are used to verify the correct implementation of the software in the programmable component.

12.8.4 Test procedures shall include a description of the test parameters, test criteria, test configuration, and any special provisions or assumptions regarding the set-up, execution, and interpretation of the test cases (See 11.2.3, 11.2.4, and 11.3.2).

12.8.5 Test cases shall be documented and traceable to the software implementation.

12.8.6 Test results shall be documented and traceable to the test case(s) that produced them.

13 Off-the-Shelf (OTS) Software

13.1 For all OTS software that interfaces with the manufacturer-supplied software, the following information shall be provided in the software plan, see 12.2:

a) The name and version/revision identifier of the OTS software;

b) The name of the OTS software provider;

c) A description of the purpose for which the software is being used;

d) A clear description of the function provided by the software;

e) An interface specification showing all control and data flows in and out of the OTS software; and

f) References to the OTS software documentation for each callable routine that interfaces with the manufacturer's software.

13.2 At least one of the following forms of evidence shall be provided for OTS software:

a) Documentation attesting to verification and testing activities of the OTS software to the extent that risks involving the OTS software are addressed.

b) Evidence that the OTS software has met formally defined requirements by an independent OTS software certification program.

13.3 When available from the OTS software developer or other sources (e.g., the user community), the manufacturer shall provide a list of known bugs for the precise revision/version of the OTS software that the manufacturer intends to use in the embedded software. For each identified error in the known bug report for the OTS software the following evidence shall be provided when implementing 13.2(a):

a) The feature that leads to an error has been fixed, tested, and approved for distribution by the OTS software developer in a new release that has been incorporated into the manufacturer's version of the software; or

b) The feature that leads to an error has not been used by the manufacturer in the development of safety-related software and does not lead to a risk.

13.4 For OTS software that performs supervisory section functions or is used by the supervisory section of software, the requirements contained in 6.5, 11.1.2(d), 11.1.2(e), 11.1.2(f), and Section 7, of this standard apply.

14 Software Changes and Document Control

14.1 Changes to parameter settings and data shall not create a risk or impact a risk that has previously been identified other than to reduce or eliminate it.

14.2 Changes or patches to the software shall not create a risk or impact a risk that has previously been identified other than to reduce or eliminate it.

14.3 To determine compliance, all changes are to be evaluated in accordance with this standard.

14.4 Documentation shall be reviewed to determine that it correctly reflects safety-related design and changes that have been made in the software.

14.5 There shall be procedures to maintain and control software changes to the configuration of the programmable components and critical and supervisory sections of software including related documentation to facilitate traceability.

15 Identification

15.1 Software shall be traceable to a unique identifier stored in non-volatile memory.

15.2 The unique identifier shall reflect changes that have been made to critical and supervisory sections of the software.

15.3 Each time a change or patch is incorporated in the critical or supervisory sections of software, a new unique identifier shall be assigned.

15.4 Documentation shall include sufficient information to identify each item that is investigated with the software. For example, identification of software elements shall include the version number, release number, and date. Microelectronic hardware elements shall include the component vendor, part number and revision level that uniquely identifies the programmable component die.

APPENDIX A (INFORMATIVE) – EXAMPLES OF MEASURES TO ADDRESS MICROELECTRONIC HARDWARE FAILURE MODES

A1 Scope

A1.1 Appendix A is to be considered normative when so stated by a product standard, directive, or regulation. Appendix A is otherwise to be considered informative.

A1.2 Table A2.1 of this appendix provides examples of acceptable measures for microelectronic hardware failure modes consistent with the requirements in Automatic Electrical Controls, Part 1, IEC 60730-1.

A1.3 Examples of software class definitions and requirements are provided in this appendix. These examples do not account for all software class definitions that may be developed in the context of applying the requirements in UL 1998. When referencing UL 1998, software classes will be further determined based on risk identification methods associated with the application of software to perform safety-related functions in products.

A1.4 Also included in this appendix are descriptions of fault models, system structures, and an example application of Table A2.1.

A2 Examples of Acceptable Measures for Microelectronic Hardware Failure Modes

A2.1 The following table provides examples of acceptable measures for covering various failure modes for select components.

A2.2 Other measures not specified in Table A2.1 are permitted if they can be shown to detect faults/error conditions specified in Table A2.1.

Table A2.1
Coverage for microelectronic hardware failure modes

Component[a]	**Fault/error**	**SW class**[b]		**Examples of acceptable measures**[c, d, e]	**Description**
		1	**2**		
1. **CPU**					
1.1 Registers	stuck-at	rq		functional test; or	A5.5
				periodic self test using either:	A5.6
				– static memory test	A7.2.9
				– word protection with single-bit redundancy	A7.3.2
	DC fault		rq	comparison of redundant CPU's by either:	
				– reciprocal comparison	A7.1.19
				– independent HW comparator; or	A7.1.6
				internal error detection; or	A7.1.10
				redundant memory with comparison; or	A7.2.8
				periodic self tests using either:	
				– walkpat memory test	A7.2.10
				– abraham test	A7.2.1
				-transparent galpat test; or	A7.2.3

Table A2.1 Continued on Next Page

Table A2.1 Continued

Component[a]	Fault/error	SW class[b]		Examples of acceptable measures[c, d, e]	Description
		1	2		
				word protection with multi-bit redundancy; or	A7.3.1
				static memory test and word protection with single-bit redundancy	A7.2.9 A7.3.2
1.2 Instruction decoding and execution	wrong decoding and execution		rq	comparison of redundant CPU's by either: – reciprocal comparison – independent HW comparator; or	 A7.1.19 A7.1.6
				internal error detection; or	A7.1.10
				periodic self test using equivalence class test	A7.1.7
1.3 Program counter	stuck-at	rq		functional test; or	A5.5
				periodic self test; or	A5.6
				independent time-slot monitoring; or	A7.1.13
				logical monitoring of the program sequence	A7.1.12
	DC fault		rq	periodic self test and monitoring using either: – independent time-slot and logical monitoring – internal error detection; or	A5.7 A7.1.14 A7.1.10
				comparison of redundant functional channels by either: – reciprocal comparison – independent HW comparator	 A7.1.19 A7.1.6
1.4 Addressing	DC fault		rq	comparison of redundant CPU's by either: – reciprocal comparison – independent HW comparator; or	 A7.1.19 A7.1.6
				Internal error detection; or	A7.1.10
				periodic self test using a testing pattern of the address lines; or	A5.7
				full bus redundance; or	A7.1.1
				multi-bit bus parity including the address	A7.1.2
1.5 Data paths	DC fault		rq	comparison of redundant CPU's by either: – reciprocal comparison or – independent HW comparator; or	 A7.1.19 A7.1.6
				Internal error detection; or	A7.1.10
				periodic self test using a testing pattern; or	A5.7
				data redundancy; or	A7.1.4

Table A2.1 Continued on Next Page

Table A2.1 Continued

Component[a]	Fault/error	SW class[b]		Examples of acceptable measures[c, d, e]	Description
		1	2		
				multi-bit bus parity	A7.1.2
2. **Interrupt handling and execution**	no interrupt or too frequent interrupt	rq		functional test; or	A5.5
				time-slot monitoring	A7.1.13
	no interrupt or too frequent interrupt and interrupt related to different sources		rq	comparison of redundant functional channels by either:	
				– reciprocal comparison	A7.1.19
				– independent HW comparator; or	A7.1.6
				independent time-slot and logical monitoring	A7.1.14
3. **Clock**	wrong frequency (for quartz synchronized clock: harmonics/ subharmonics only)	rq		frequency monitoring; or	A7.1.11
				time slot monitoring	A7.1.13
			rq	frequency monitoring; or	A7.1.11
				time slot monitoring; or	A7.1.13
				comparison of redundant functional channels by either:	
				– reciprocal comparison	A7.1.19
				– independent HW comparator	A7.1.6
4. **Memory**					
4.1 Non-volatile memory	all single-bit faults	rq		periodic modified checksum; or	A7.2.4
				multiple checksum; or	A7.2.5
				word protection with single-bit redundancy	A7.3.2
	all single and double bit errors		rq	comparison of redundant CPU's by either;	
				– reciprocal comparison or	A7.1.11
				– independent HW comparator; or	A7.1.6
				redundant memory with comparison; or	A7.2.8
				period cyclic redundancy check, either:	
				– single word	A7.2.6
				– double word; or	A7.2.7
				word protection with multi-bit redundancy	A7.3.1
4.2					

Table A2.1 Continued on Next Page

Table A2.1 Continued

Component[a]	Fault/error	SW class[b]		Examples of acceptable measures[c, d, e]	Description
		1	2		
Volatile memory	DC fault	rq		periodic static memory test; or	A7.2.9
				word protection with single-bit redundancy	A7.3.2
	DC fault and dynamic cross links		rq	comparison of redundant CPU's by either:	
				– reciprocal comparison	A7.1.19
				– independent HW comparator; or	A7.1.6
				redundant memory with comparison; or	A7.2.8
				periodic self tests using either:	
				– walkpat memory test	A7.2.10
				– abraham test	A7.2.1
				– transparent galpat test; or	A7.2.3
				word protection with multi-bit redundancy	A7.3.1
4.3 Addressing (relevant to variable and invariable memory)	stuck-at	rq		word protection with bit parity including the address	A7.3.2
	DC fault		rq	comparison of redundant CPU's by either:	
				– reciprocal comparison or	A7.1.19
				– independent HW comparator; or	A7.1.6
				full bus redundancy	A7.1.1
				testing pattern; or	A7.1.24
				period cyclic redundancy check, either:	
				– single word	A7.2.6
				– double word; or	A7.2.7
				word protection with multi-bit redundancy including the address	A7.3.1
5. **Internal data path**					
5.1 Data	stuck-at	rq		word protection with single-bit redundancy	A7.3.2
	DC fault		rq	comparison of redundant CPU's by either:	
				– reciprocal comparison	A7.1.19
				– independent HW comparator; or	A7.1.6
				word protection with multi-bit redundancy including the address; or	A7.3.1
				data redundancy; or	A7.1.4
				testing pattern: or	A7.1.24
				protocol test	A7.1.18
5.2	wrong address	rq		word protection with single-bit redundancy including the address	A7.3.2

Table A2.1 Continued on Next Page

Table A2.1 Continued

Component[a]	Fault/error	SW class[b]		Examples of acceptable measures[c, d, e]	Description
		1	2		
Addressing					
	wrong and multiple addressing		rq	comparison of redundant CPU's by:	
				– reciprocal comparison	A7.1.19
				– independent HW comparator; or	A7.1.6
				word protection with multi-bit redundancy, including the address; or	A7.3.1
				full bus redundancy; or	A7.1.1
				testing pattern including the address	A7.1.24
6. **External communication**					
6.1 Data	all single-bit and double bit errors	rq		word protection with multi-bit redundancy; or	A7.3.1
				CRC – single word; or	A7.2.6
				transfer redundancy; or	A7.1.5
				protocol test	A7.1.18
	all single-bit, double bit and triple bit errors		rq	CRC – double word; or	A7.2.7
				data redundancy; or	A7.1.4
				comparison of redundant functional channels by either:	
				– reciprocal comparison	A7.1.19
				– independent HW comparator	A7.1.6
6.2 Addressing	wrong address	rq		word protection with multi-bit redundancy, including the address; or	A7.3.1
				CRC – single word, including the addresses; or	A7.2.6
				transfer redundancy; or	A7.1.5
				protocol test	A7.1.18
	wrong and multiple addressing		rq	CRC – double word, including the address; or	A7.2.7
				full bus redundancy of data and address; or	A7.1.1
				comparison of redundancy communication channels by either:	
				– reciprocal comparison	A7.1.19
				– independent HW comparator	A7.1.6
6.3 Timing	wrong point in time	rq		time slot monitoring; or	A7.1.13
				scheduled transmission	A7.1.21

Table A2.1 Continued on Next Page

Table A2.1 Continued

Component[a]	Fault/error	SW class[b]		Examples of acceptable measures[c, d, e]	Description
		1	2		
			rq	time slot and logical monitoring; or	A7.1.15
				comparison of redundant communication channels by either:	
				– reciprocal comparison	A7.1.19
				– independent HW comparator	A7.1.6
	wrong sequence	rq		logical monitoring; or	A7.1.12
				time slot monitoring; or	A7.1.13
				scheduled transmission	A7.1.21
			rq	time slot and logical monitoring; or	A7.1.14
				comparison of redundant communication channels by either:	
				– reciprocal comparison	A7.1.19
				– independent HW comparator	A7.1.6
7. **Input/output periphery**					
7.1 Digital I/O	open and short circuit or as specified in the product standard	rq		plausibility check	A7.1.17
			rq	comparison of redundant CPU's by either:	
				– reciprocal comparison	A7.1.19
				– independent HW comparator; or	A7.1.6
				Input comparison; or	A7.1.9
				multiple parallel outputs; or	A7.1.15
				output verification; or	A7.1.16
				testing pattern; or	A7.1.24
				code safety	A7.1.3
7.2 Analog I/O					
7.2.1 A/D-and D/A-converter	open and short circuit or as specified in the product standard	rq		plausibility check	A7.1.17
			rq	comparison of redundant CPU's by either:	
				– reciprocal comparison	A7.1.19
				– independent HW comparator; or	A7.1.6
				Input comparison; or	A7.1.9
				multiple parallel outputs; or	A7.1.15
				output verification; or	A7.1.16

Table A2.1 Continued on Next Page