DOT/FAA/TC-17/50

Federal Aviation Administration
William J. Hughes Technical Center
Aviation Research Division
Atlantic City International Airport
New Jersey 08405

Commercial Off-The-Shelf Airborne Electronic Hardware Assurance Methods—Phase 3—Embedded Controllers

November 2017

Final Report

U.S. Department of Transportation
Federal Aviation Administration

NOTICE

Technical Report Documentation Page

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
DOT/FAA/TC-17/50		

4. Title and Subtitle	5. Report Date
COMMERCIAL OFF-THE-SHELF AIRBORNE ELECTRONIC HARDWARE ASSURANCE METHODS – PHASE3 – EMBEDDED CONTROLLERS	November 2017
	6. Performing Organization Code

7. Author(s)	8. Performing Organization Report No.
Laurence H. Mutuel	D8

9. Performing Organization Name and Address	10. Work Unit No. (TRAIS)
Thales Air Traffic Management U.S. 10950 El Monte Street Overland Park, Kansas 66211	11. Contract or Grant No.

12. Sponsoring Agency Name and Address	13. Type of Report and Period Covered
U.S. Department of Transportation Federal Aviation Administration 950 L'Enfant Plaza SW Washington, DC 20024	Final Report
	14. Sponsoring Agency Code AIR-134

15. Supplementary Notes

The FAA William J. Hughes Technical Center Aviation Research Division Technical Monitor was Manny Rios.

16. Abstract

This report provides information on assurance methods for commercial off-the-shelf (COTS) airborne electronic hardware (AEH) embedding microcontrollers. These methods address the safe operations aspects of certification and contribute to the development of a comprehensive framework for COTS assurance. This research highlights the current use of microcontrollers in AEH and the trends for the near future, and describes the challenges related to safety.

Microcontrollers cover a large spectrum of devices, which complicates the definition of an overarching method to determine the applicability of assurance methods. The current classification is primarily driven by the microcontrollers' manufacturers and therefore focuses on hardware characteristics that may or may not impact the ability to perform a safety assessment in a manner consistent with the system-level assessment.

The proposed safety assurance process follows the steps of system safety assurance, which is the most applicable, given that COTS microcontrollers embed both software and hardware functions. These steps include the identification of failure modes, the determination of their effects given an intended use at system level and usage domain at component level, the classification of these effects, and the recommendation of mitigation techniques. As these components are COTS, their design may or may not include built-in mitigation techniques. The last step is to assess the effectiveness of the built-in mitigations and recommend additional ones as needed to meet the safety objectives. Existing guidance documents for assurance at system-level, software, and hardware-levels, and regulatory material are reviewed for applicability. Tailoring is recommended to better cover COTS microcontrollers.

The approach is to target the main objectives of a structured development process (i.e., ensuring the correct functional performance under all foreseeable conditions with no anomalous behavior). The aim is to return to certification basics of 14 CFR (i.e., 14 CFR 23/25/27/29.1301 and 23/25/27/29.1309): intended functions that are fit-for-purpose, proper, and safe functioning within aircraft operating conditions, and technical suitability supporting continued airworthiness.

17. Key Words	18. Distribution Statement
COTS, Embedded controller, Microcontroller, assurance, Failure mode, ECC, AEH, Redundancy, Fault injection, Safety process	This document is available to the U.S. public through the National Technical Information Service (NTIS), Springfield, Virginia 22161. This document is also available from the Federal Aviation Administration William J. Hughes Technical Center at actlibrary.tc.faa.gov.

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price
Unclassified	Unclassified	49	

ACKNOWLEDGEMENTS

This research has been coordinated with technical experts and reviewers at Thales Avionics SAS by Cyril Marchand. Didier Regis and Guy Berthon acted as internal reviewers for this report.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVATIONS AND ACRONYMS

A/D	Analog to digital
AEH	Airborne electronic hardware
AFE	Authority for Expenditure
ALU	Arithmetic logic unit
BIT	Built-in test
CFC	Control flow checking
CISC	Complex instruction set computer
COTS	Commercial off-the-shelf
CPU	Central processing unit
D/A	Digital to analog
DAL	Development assurance level
DMA	Direct memory access
EASA	European Aviation Safety Agency
ECC	Error correcting code
FHA	Functional Hazard Assessment
FMEA	Failure mode and effects analysis
FTA	Fault tree analysis
I2BCFC	Intra-inter block control flow checking
IDAL	Item development assurance level
IP	Intellectual property
MMU	Memory management unit
NAND	Not-AND
OS	Operating system
RAM	Random access memory
RISC	Reduced instruction set computer
ROM	Read-only memory
RTCA	Radio Technical Commission for Aeronautics
RTL	Register transfer level
SDS	Software and Digital Systems
SPI	Serial peripheral interface
SRAM	Static random access memory
SWTES	Software-based error detection technique using encoded signatures
UART	Universal Asynchronous Receiver/Transmitter
VeTeSS	Verification and Testing to Support Functional Safety Standards

EXECUTIVE SUMMARY

This report is part of a research thrust on commercial off-the-shelf (COTS) airborne electronic hardware (AEH) assurance methods that addresses the "safe operation" aspect of certification and promotes the development of a comprehensive framework for COTS assurance. This framework covers:

- The understanding of current and foreseen future uses of COTS in AEH.
- The description of related safety issues and concerns.
- The documentation of failure modes and their relevance to an AEH context.
- The investigation of existing mitigation techniques and their effectiveness.
- The development of objective criteria for determining the effectiveness of safety.
- The determination of airworthiness assurance methods for AEH integrating the COTS under consideration.

This report applies the above focus points to embedded controllers (also called COTS microcontrollers). It also initiates the process of investigating the shortcomings of current assurance methods and the potential for remedies. Applicable assurance methods include system-level guidelines within SAE/ARP4754A and SAE/ARP4761, software-level guidance in RTCA/DO-178C, and hardware-level guidance in RTCA/DO-254. Other assurance standards exist in other domains, particularly for road vehicle safety in ISO 26262, that recommend assurance processes applicable to microcontrollers.

Microcontrollers cover a large spectrum of devices; this complicates the definition of an overarching method to determine how to apply assurance methods and which ones are applicable. Currently, the classification schemes are mostly based on hardware-related characteristics that may or may not impact the ability to perform a safety assessment in a manner consistent with the system-level assessment.

This report focuses on microcontrollers embedded within COTS products, not only executing software but also providing so-called peripheral hardware functions. This distinction eliminates COTS controllers (i.e., not able to execute application software) and core processors alone (i.e., not providing peripheral hardware functions other than those necessary for interfacing outside the device, such as memory access controllers). Because of both technology capability and performance needs, the current trend is moving toward highly integrated core processors that embed on the same die the potentially multiple core processing, memories, and peripherals.

Microcontroller manufacturers categorize their devices primarily by their technical performance in terms of number of hardware functions, core processing power capabilities, and other physical characteristics, such as electrical power consumption. However, the certification community using DO-254-based classification criteria addresses hardware characteristics as they relate to the notion of complexity; unfortunately, the notion of complexity inevitably suggests multiple interpretations. EASA certification memorandum SWCEH-001 further introduced the category of the highly complex microcontroller based on criteria related to the internal architecture of a device. This introduction further increased the range of interpretative material that would be required to determine an acceptable path to certification compliance. Beyond the mere assessment of COTS

microcontrollers' characteristics, the relationship with the objectives of the development-assurance process would require clarification.

Therefore, the approach is to target the main objectives of a structured development process (i.e., ensuring the correct functional performance under all foreseeable conditions with no anomalous behavior). The aim is to return to certification basics of 14 CFR (i.e., 14 CFR 23/25/27/29.1301 and 23/25/27/29.1309): intended functions that are fit-for-purpose, proper and safe functioning within aircraft operating conditions, and technical suitability supporting continued airworthiness. System-level safety assessments are then applicable to identify microcontrollers' failure modes, analyze and classify their effects according to the applicable classification scheme (e.g., per AC 25.1309), and evaluate existing failure mitigations.

The automobile industry provides most of the verification results from fault injection in microcontrollers. The fault testing is performed at physical and logical levels, but rarely at the functional level at which the intended use is attached. The results indicate that most faults in microcontrollers manifest as bit-flips, justifying the most common mitigation technique to be error detection and correcting codes. Memory management units provide wider fault detection and recovery and are more and more embedded in microcontrollers. However, with respect to AEH for COTS in general and for complex or highly complex COTS microcontrollers in particular, design data are not available, at least to the level necessary to provide sufficient development assurance commensurate with the expected usage.

This report provides recommendations for assurance process for COTS microcontrollers. An a-priori classification could be determined by (1) taking into account the criticality (from the allocated development assurance level), and (2) on the basis of an assessment of both the device characteristics and its target usage domain. The notion of independence between the processing core (performing purely software functions) and the other peripheral hardware functions can inform on the potential applicability and scope of system-level safety assurance combined with software-level assurance (when independence claim cannot be formally justified), or hardware-level assurance only (with justified independence claims). In other words, the challenge is to discern among miscellaneous considerations, whether or not a COTS microcontroller can be assured at system-level, possibly with the help of software, and, if not, how hardware-level assurance only can be achieved. In the latter case, collecting all available artifacts in a structured manner could provide acceptable assurance. For example, the definition and analysis of a device's usage domain, together with its validation and verification, could allow a COTS microcontroller classified as complex or highly complex under current criteria to still be shown as mastered in terms of both adequate functioning and safe operation. For highly complex microcontrollers with allocated development assurance levels of A or B, additional assessments of potential dysfunctional behavior should be performed. Finally, the issue of documentation to support the required artifacts is no different from other COTS devices. The recommendation is to integrate the information from the available COTS artifacts into the development process to perform to the largest extent possible the traditional activities of a requirements-based approach (e.g., requirements capture, requirements validation, design data production, consistency of the overall process via traceability, implementation within the surrounding AEH, and performing requirements-based verification).

1. INTRODUCTION

1.1 BACKGROUND

Commercial off-the-shelf (COTS) items are increasingly penetrating into both the commercial and the military segments of the aerospace market. Radio Technical Commission for Aeronautics (RTCA) standards, namely DO-254 "Design Assurance Guidance for Airborne Electronic Hardware" [1] and DO-178C "Software Considerations in Airborne Systems and Equipment Certification" [2], were developed with the issue of COTS assurance in mind. However, they did not recommend specific methods or objective criteria for safety assurance and airworthiness.

In a general context of airworthiness, the focus is threefold:

- Intended function (14 CFR 23.1301, 25.1301, 27.1301, and 29.1301)—Component selection from a functional standpoint and design to meet the function contribute to this aspect.
- Operating conditions (14 CFR 23.1309, 25.1309, 27.1309, and 29.1309)—Component selection from the point of view of characteristics and performance and environmental qualification of these components contribute to this aspect.
- Safe operation (14 CFR 23.1309, 25.1309, 27.1309, and 29.1309)—Failure modes and failure mitigation of the selected components contribute to this aspect [3–6]. Reliability considerations are integrated in the demonstration of achievement of safe operation.

The research thrust on COTS airborne electronic hardware (AEH) assurance methods addresses safe operation and supports the development of a comprehensive framework for COTS assurance. The framework includes:

- The understanding of current and foreseen future use of COTS in AEH.
- The description of safety issues and concerns.
- The documentation of failure modes for these COTS and the relevance to AEH context.
- The investigation of existing mitigation techniques and their effectiveness.
- The development of objective criteria for determining the effectiveness of safety.
- The determination of airworthiness assurance methods for AEH integrating the COTS under consideration.

Previous research under Authority for Expenditure (AFE) Project 75 (COTS AEH Assurance Methods) documented 22 COTS issues and proposed a structure to address future COTS AEH assurance standards [7–10]. Continuation of the research on the identified issues with COTS is currently allocated in part to a supplement to AFE Project 75 and to a phase 3 research task order under the Software and Digital Systems (SDS) Program, which focuses on commodity memories and embedded controllers [11]. This report is produced under the SDS Program and pertains to embedded controllers. Note that in the process of describing the COTS embedded controllers in AEH, considerations of intended function and operating conditions are integrated in the discussion.

1.2 PURPOSE

This report is the second on the SDS research and addresses embedded controllers (also called microcontrollers). This report informs on applicability of safety assurance processes based on the characteristics of microcontrollers, their fault models, and mitigation means.[1] More specifically, the following topics are covered:

- The identification of the various types of embedded controllers that are being used and might be used in the near future by aerospace equipment manufacturers
- The definition of categories for embedded controllers based on their characteristics
- The identification of methods to categorize a given device into one of the defined categories
- The description of the types of embedded controllers failure modes
- The identification of issues and adequacy associated with the embedded controllers' built-in fault mitigation techniques for the above failure modes (if any)
- The identification of additional potential internal and external fault mitigation techniques for the defined categories of embedded controllers
- The investigation of how embedded controllers could be integrated within system safety analyses (e.g., fault trees)
- The recommendation of software and AEH development assurance processes, including means of compliance

In the context of investigating assurance methods for embedded controllers, the focus is on COTS microcontrollers, which embed a processing core with software, as opposed to COTS controllers, which do not have a software function. Therefore, this report uses the term "microcontroller." Furthermore, the focus is on microcontrollers embedded within COTS products, executing not only protected code (from an intellectual property [IP] standpoint) but also providing hardware functions (e.g., memory controllers or input/output functions).

Section 2 of this report discusses the identification and classification schemes for microcontrollers. Section 3 describes the fault model of microcontrollers at different abstraction levels and lists identified issues with these COTS devices. Section 4 investigates the effects of faults in microcontrollers and the mitigation techniques. Section 5 focuses on the applicability of assurance processes, to include system, software, and hardware; this section also consolidates the findings and recommendations resulting from this research effort.

[1] Disclaimer: In this report, COTS components for which the primary target is the consumer electronics market may be qualified as exhibiting a level of quality that is insufficient for AEH quality requirements, especially for safety-critical applications. This statement does not mean in any way that manufacturers may be lax or incompetent. What is meant is that manufacturing and supply processes that are perfectly rigorous and appropriate for the consumer electronics market may not meet requirements for AEH application, which require a mandatory level of assurance.

2. IDENTIFICATION OF EMBEDDED CONTROLLERS FOR AEH PURPOSES

This section provides the context information for executing the overarching tasks of a safety process: the identification and description of failure modes associated with embedded controllers, the assessment of their effects, and the determination of mitigation techniques.

The main challenge is related to the wide variety of embedded controllers, not only in terms of features but also in terms of intended functions. The objective of the following sections is to recommend a categorization scheme using characteristics that will be relevant in the context of safety assurance.

2.1 CHARACTERISTICS OF EMBEDDED CONTROLLERS

2.1.1 Definition and Architectures

An embedded controller is defined as a microprocessor-based system that is built to control a function or a range of functions that resides in a larger mechanical or electrical system. An embedded controller is also referred to as a microcontroller. This definition is coherent with the scope of this report to consider controllers embedded in a larger COTS product. However, the proposed scope is wider.

European Aviation Safety Agency (EASA's) definition of a COTS microcontroller is consistent with the generic definition above; the definition is more specific because it includes controllers that embed several hardware functions interacting with the processing core executable code. CM SWCEH-001 [12] defines a COTS microcontroller as any integrated circuit that executes software in a central processing unit (CPU) (processing core) and that implements peripheral hardware elements.

A generic microcontroller is described as a highly integrated component containing a single processing core, an internal memory, and programmable input/output peripheral interfaces and controllers (see figure 1).

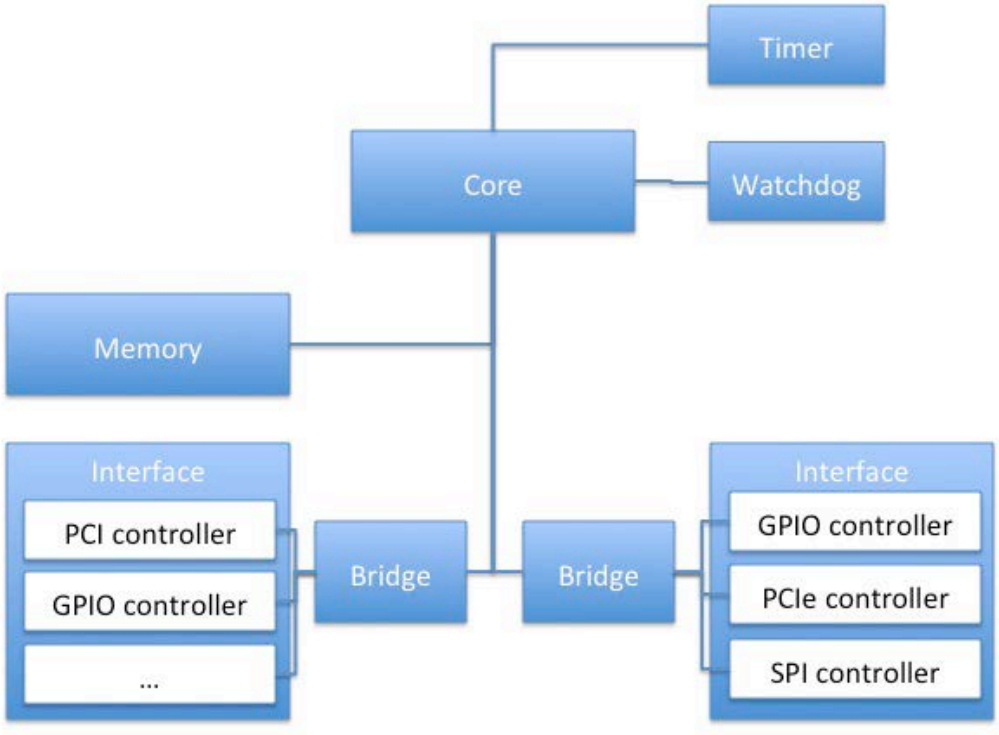

Figure 1. Block diagram of generic microcontroller

This figure illustrates some of the most common peripheral interfaces: general-purpose input/output, serial peripheral interface (SPI), peripheral component interconnect, and express.

For a COTS product in general, the only available level of description is the one provided by its datasheet (e.g., block diagram, functional, and interface description), possibly complemented by additional data from the COTS supplier (e.g., users manuals and applications notes). This level is mainly a very first level of description (i.e., down to the strict necessary information for the user, at the system or software interface) to determine normal and proper functioning of the COTS to the desired behavior. For COTS microcontrollers in particular, the software interface is described through both the instruction set of the core processing portion and a set of control and status registers for the associated peripherals.

The assessment of failure modes and mitigation mechanisms requires a more dysfunctional than functional analysis, which can then only be performed at the COTS device level but is based on the available description of the architecture in terms of input, functional blocks diagram, and output. With those restrictions, the failure modes and mitigation mechanisms can only be assessed at this first level of description. In addition, failure rates and fault coverage figures can only be estimated, possibly with the help of the COTS device supplier but with inevitable assumptions, which might be difficult to validate.

Consider specific instances to highlight the spectrum of embedded controllers. Figure 2 shows the block diagram of a MPC7447A, which is the fifth implementation of the fourth generation microprocessors from Freescale [13]. It is a standalone processing core. In comparison, figure 3 shows the block diagram of a MPC8610, which is an integrated processor: This single chip replaces four chips. All core-to-peripheral connections are integrated on the die [14].

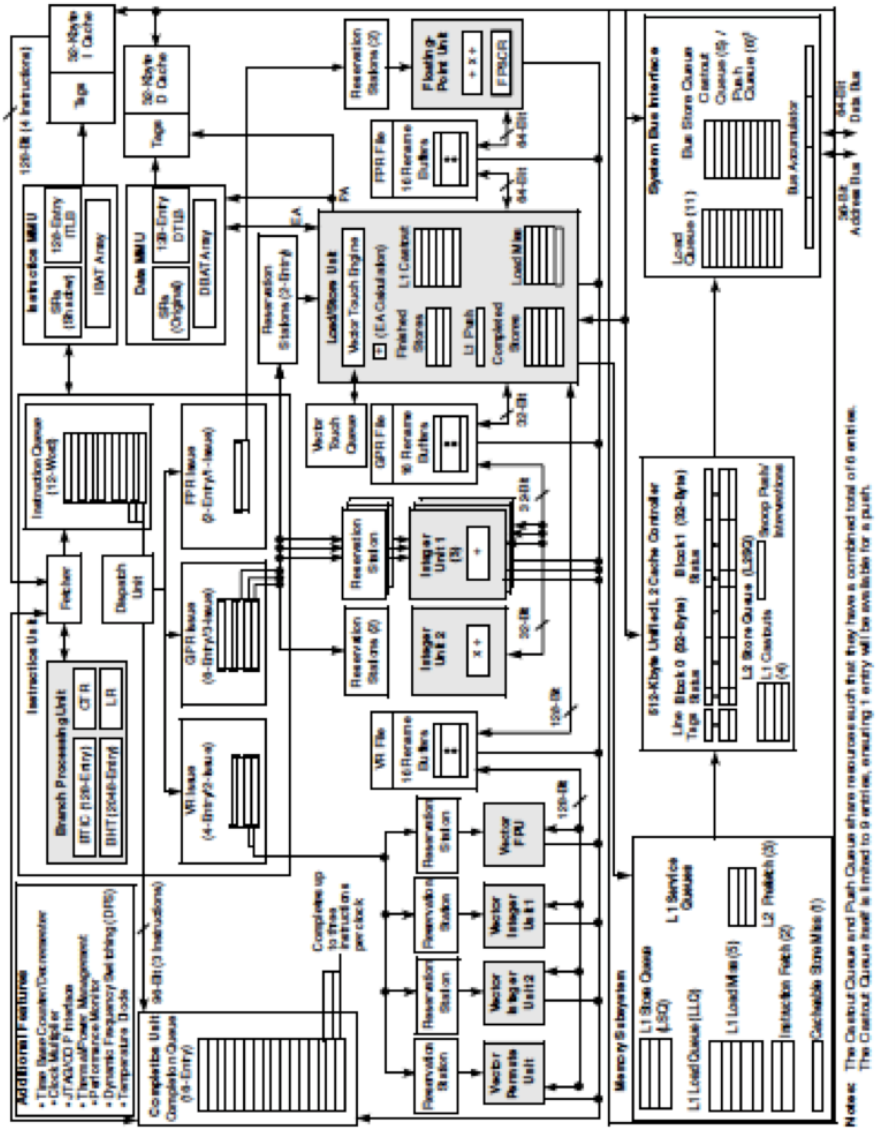

Figure 2. Block diagram of Freescale MPC7447A standalone core processing

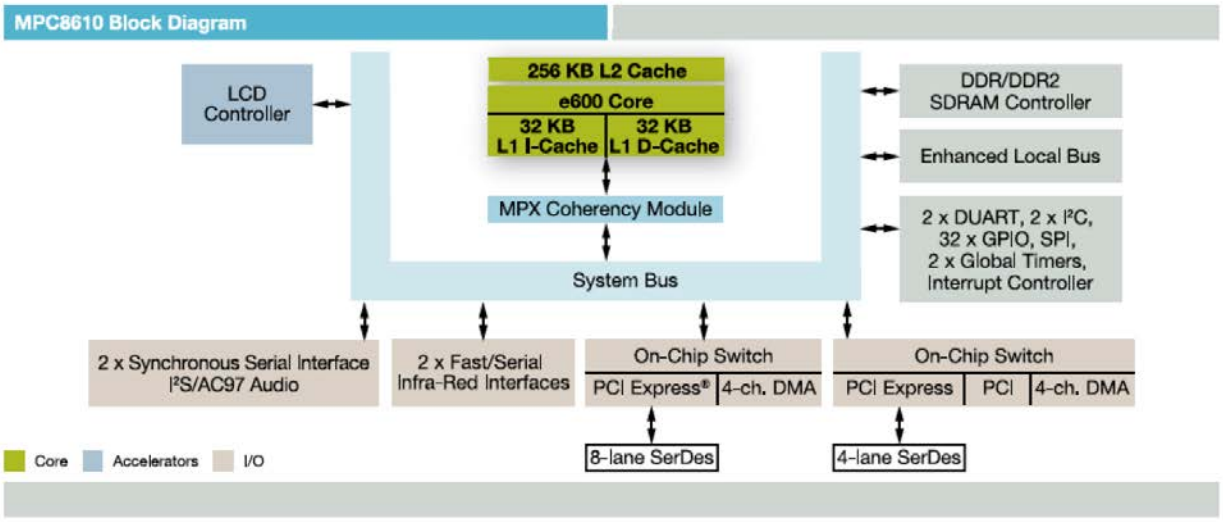

Figure 3. Block diagram of Freescale MPC8610 integrated host processor

The spectrum of available devices is significantly varied, and often the AEH designer and manufacturer need to work closely with the embedded controller supplier to ensure the availability of information that is required to support the development assurance processes.

2.1.2 Microcontrollers Families

The classification by family is provided as an example. It is specific to the microcontroller manufacturer and bears no impact on the assurance issue that is not covered by one or several of the criteria based on technical characteristics.

- Atmel® AVR® family – tinyAVR®, megaAVR®, AVR XMEGA®
- PIC family – PIC16F, PIC18F
- ARM family – ARM7, ARM9, ARM11
- Intel 8051 family – AT89s52, p89v51rd2
- Motorola – 68HC11

2.2 CRITERIA USED FOR CATEGORIZING MICROCONTROLLERS

Categorization of COTS microcontrollers may be performed based on one or a combination of several of the criteria in this section. These criteria are currently being used, but they may not be relevant to the objective of integrating embedded controllers in the safety-assurance process. Section 5.4.1 discusses recommendations to remedy the classification issues related to complexity, criticality, and hardware-based criteria.

2.2.1 Functionality or Usage

This criterion pertains to whether the controller is dedicated to a specific function of a particular hardware or whether the controller is generic. Generic embedded controllers may perform the following tasks:

- Receiving and processing signals from controls (e.g., switches and buttons)
- Powering on and off
- Managing access control
- Managing thermal controls (e.g., fan control, CPU throttling, emergency shutdown in response to over-temperature condition)
- Controlling the indicator-light-emitting diodes
- Managing battery power and battery charging
- Allowing diagnostics and remote fault management
- Performing software-commanded reset
- Controlling the watchdog timer

This criterion is key to the safety-assurance process, regardless of whether the functionalities are performed by software, hardware, or both. However, it is rare for this criterion to drive the classification.

2.2.2 Complexity

This criterion allows for differentiating controllers that are simple sequencers, perform arithmetic logic, or are reconfigurable.

As a start, the following definitions are based on DO-254 and are used in the classification of microcontrollers:

- Simple electronic hardware—A hardware device is considered simple only if a comprehensive combination of deterministic tests and analyses appropriate to the development assurance level (DAL) /item development assurance level (IDAL) can ensure correct functional performance under all foreseeable operating conditions with no anomalous behavior.
- Complex electronic hardware – All devices that are not simple are considered to be complex.

Note: An interpretation of the above definition for simple electronic hardware COTS is based on EASA CM SWCEH-001 and can be summarized as “the ability to verify by test on the physical device all requirements in all configurations, is a pre-requisite for a classification of a device as simple.” [12]

A new category was introduced in EASA CM SWCEH-001 in addition to the DO-254 definitions for highly complex COTS microcontrollers. The definition invokes architectural specificities, such as number of cores, peripherals, or buses within the device. The practicality of such a definition may be questionable as the complexity gap is then highly dependent on the type of functionality provided by the COTS product and not on universally acceptable independent criteria. This might be the reason why only COTS microcontrollers will fall under this new category. As evolving technology will typically lead to an increase in complexity, other COTS products could eventually fall in that category. This classification has not been adopted by the FAA to date, the guidance for which is in reference [15], and the author of this report, as elaborated in section 5.4.1, instead recommends a classification that is less dependent on the microcontroller’s functionality.

The EASA COTS-AEH report [16] and system-on-chip research report [17] considered microprocessors to be standalone complex COTS. Simple microcontrollers can be defined as microcontrollers implementing simple peripherals around their core processing.

Examples of simple interface peripherals include universal asynchronous receivers/transmitters, inter-integrated circuits, and SPIs. Examples of simple microcontrollers include Freescale MPC5567, the Texas Instrument C2000™ microcontroller series, or the older NXP LPC2119. Examples of complex microcontrollers include the Texas Instruments digital signal processor or Freescale MPC8610 (see figure 3).

Categorization by complexity is generally performed via an assessment of the COTS characteristics, which requires an adequate description of the COTS architecture down to an acceptable level of functional block identification, and with respect to one or several generally accepted criteria. Such criteria include the observable behavior of the various internal functions.

Categorization of COTS by complexity has always generated misunderstandings.

2.2.3 Criticality

Criticality, as reflected by the allocated DAL, is an input requirement flowed down from the system safety assessment. Although it cannot be modified, it is unlikely that compliance to the allocated DAL can be shown because the COTS design/development data may not be available.

2.2.4 End-User Accessibility

This criterion specifically addresses whether the end user can modify the controller's instruction set or whether the user has access to the source code.

2.2.5 Instruction Type or Set

This criterion presupposes end-user accessibility and focuses on whether the instruction set is complex instruction set computer (CISC) or reduced instruction set computer (RISC).

With CISC, the programmer can use one instruction in place of many simpler instructions. The RISC instruction set allows each instruction to operate on any register or to use any addressing mode and simultaneous access of program and data. RISC systems shorten execution time by reducing the clock cycles per instruction. CISC systems shorten execution time by reducing the number of instructions per program and having the hardware complete complex instructions.

The selection of instruction type or instruction set has no impact on the assurance issues (e.g., safety impact, intended function) that the classification would help address.

2.2.6 Instruction Storage

Embedded controllers typically have their own random access memory (RAM), independent from the memory for the main processing unit. The controller software may be stored in its own read-only memory (ROM).

This criterion allows for differentiating between controllers fetching instructions in an internal memory (i.e., embedded memory microcontroller) and controllers fetching instructions in an external memory (i.e., external memory microcontroller). Sub-criterion could consider the type: flash memory, electrically erasable programmable ROM, and static random access memory (SRAM).

The integration of memory units within the microcontroller implies that a large amount of computation by the core processing unit is hidden to the systems with which it interfaces.

An embedded microcontroller has all functional blocks on a chip: program, data memory, input/output ports, serial communication, counters, timers, and interrupts. An example of such structure is Intel® 8051.

An external memory microcontroller has no program memory on a chip. An example of such structure is Intel 8031.

This criterion plays a role in the assurance process, as instruction storage may be a source of issue. The following is an example: an embedded hardware controller that checks, or even corrects, a value during instruction fetching or RAM access.

2.2.7 Memory Architecture

This criterion allows for distinguishing between Harvard memory architecture microcontrollers and Princeton memory architecture microcontrollers.

Harvard memory architecture microcontrollers have a dissimilar memory address space for the program and data memory in their processor. Princeton memory architecture microcontrollers have a common memory address for the program and the data memory in their processor.

2.2.8 Internal Bus Width

This criterion classifies microcontrollers based on internal bus widths: 4 bits, 8 bits, 16 bits, or 32 bits.

In microcontrollers with 8-bit internal bus widths, the arithmetic logic unit (ALU) performs the arithmetic and logic operations at specific precision and performance. Examples of 8-bit microcontrollers are Intel 8031/8051, PIC1x, and Motorola MC68HC11 families.

Precision and performance are increased in microcontrollers with 16-bit internal bus widths compared with 8-bit microcontrollers. The timer can also be longer. Examples of 16-bit microcontrollers are Intel 8051XA and 8096, PIC2x, and Motorola MC68HC12 families.

Microcontrollers with 32-bit internal bus widths are typically used in automatically controlled devices, including implantable medical devices, engine/power control systems, office machines, and various appliances. Examples include Intel/Atmel 251 and PIC3x families.

This criterion primarily covers the performance aspects (precision and speed); although increased performance may allow for more critical applications, this particular criterion is qualified by the author and technical reviewers as having no impact on safety assurance issues. The usage criterion is retained instead for safety impacts.

2.3 USE OF MICROCONTROLLERS IN AEH

2.3.1 Current Usage

Microcontrollers are currently used in aircraft to perform the following functions:

- Receiving and processing signals from controls in the cockpit control panels
- Powering on and off in power distribution units (standalone or embedded in the chassis of avionics systems)
- Managing thermal controls (e.g., fan control, CPU throttling, emergency shutdown in response to over-temperature conditions) typically embedded in the avionics chassis
- Controlling the indicator-light-emitting diodes on faceplates and cockpit panels
- Managing battery power and battery charging
- Controlling the watchdog timers on built-in test (BIT) programs (e.g., power-on BIT)

Microcontrollers are associated with NAND flash memory for all its applications in AEH.

2.3.2 Foreseen Future Usage

The foreseen usage of microcontrollers in AEH revolves around an increasing complexity in the control and monitoring functions. The microcontrollers are already available and embedded in the chassis of avionics systems. They control temperature, power sequencing, and power distribution. For the monitoring function, foreseen use includes the capability to log data in the internal memory and communicate failure conditions with the external environment using hardware discrete signals.

The control function also comes into play when allowing external systems to connect and transfer of information between two remote systems. This growing functionality supports integrated diagnostics and remote fault management. The most common fault management strategy is to perform software-commanded resets. Microcontrollers perform this function.

In association with non-volatile memory storage, microcontrollers will take over the boot-loading capability and access control to data stored in memory for more complex applications than are currently used.

3. FEARED EVENTS AND FAILURE MODES OF EMBEDDED CONTROLLERS

As previously discussed in this task order, the following are the failure modes identified in the context of abstraction level, from the most abstract description to the least abstract:

- Functional abstraction level
- Logical abstraction level
- Physical abstraction level

The functional abstraction level contains both a system-level and a hardware-level transfer function representation. At this level, the microcontroller cannot be investigated independently from its intended function. The notion of usage domain is attached to an investigation at this abstraction level using the system-level transfer function. When considering the hardware-level transfer function, the focus of the failure mode identification is on the delivery of a coherent signal to the different software levels. Functional fault detection and mitigation means for the hardware-level transfer function viewpoint to cover the service provided by hardware to connected hardware or to software.

The logical abstraction level addresses the output of the microcontroller from its logical content viewpoint. This level is the most adapted to investigate failure modes of microcontrollers and to identify the possible fault detection and mitigation means.

At the physical level, the microcontroller's outputs are separated, and the signals are detailed down to the physical characteristics, including voltage, amperage, and timing. This abstraction level is too low to effectively describe the failure modes of a complex COTS microcontroller (e.g., because of the number of input/output pins). However, this level of abstraction is useful in particular cases, such as to characterize a failure mode identified at a higher level (e.g., bit stuck-at value, voltage oscillations, timing drift) or to estimate the effectiveness of detection/mitigation means (e.g., monitoring of a signal). This abstraction level is also typically used for faults caused by environmental factors, such as single-event effects, electromagnetic pulse, current, or voltage fluctuations.

Most of the literature addressing microcontrollers and compliance with safety requirements relates to the automobile industry and demonstration of compliance with ISO 26262 standard requirements [18]. The study of microcontrollers—even complex—is often limited to hardware-related issues, but there is a trend in the latest publications to consider software-related safety issues.

3.1 GLOBAL FAULT CLASSIFICATION

Faults are typically classified into two broad categories:

- Permanent faults
- Transient faults

The effects of permanent faults are non-reversible. Based on this consideration, aging effects can be included in the permanent fault category, whereas intermittent faults can be included in the transient fault category. Reference [19] considers specifically faults caused by electromagnetic interferences and adds the class of static faults: The pulse width is sufficient to deviate the microcontroller's state parameters from the rated values, but no abnormal behavior is observed at the system level.

Errors are classified in two categories:

- Soft errors can be recovered from, and typically with, a system restart.
- Hard errors require replacement of affected components.

Faults on a target system (e.g., microcontroller) can be investigated from three different levels:

- Hardware faults: covers the physical device, the logic, and the register-transfer level (RTL)
- Software faults: applies to operating system (OS) (kernel and middleware) and application
- Operation faults: covers faults committed by the user when interacting with the system

Furthermore, hardware faults can be tagged according to two categories of causal factors:

- Random hardware failures, which include faults caused by the environment
- Systematic failures or faults related to defects introduced during the design, the manufacturing process, and (visible or not) after production testing. Systematic failures can also be introduced by the operational procedures or documentation deficiencies.

For COTS products and microcontrollers, design errors can be introduced because of the complexity of the design process; they may include: non-instantiated features, non-documented features interfering with documented features, or erroneous behavior under conditions that were not tested by the COTS manufacturer (related to usage domain). Manufacturing errors may include contamination of the silicon, micro-cracks in wafer, etc.

3.2 ISSUES IDENTIFIED WITH COTS MICROCONTROLLERS

FAA COTS AFE75 report [7] and EASA COTS-AEH report [16] have identified several issues with microcontrollers. This section summarizes the issues and how they could influence the selection of characteristics in the proposed classification scheme (see section 5.4.1).

3.2.1 Issues with Derating

Microcontrollers are subject to derating practices from the device manufacturer. These practices impact voltage, frequency, and input/output current. Impacts on the device architecture include down binning, power-aware design, and process scaling.

3.2.2 Sparing Reliability

With process scaling and reduction in feature size in general, the behavior of the material changes so that the percentage of on-chip defects increases. One of the architectural solutions to this issue is the implementation of on-chip redundancy. For microcontrollers, this physical redundancy may lead to the following situations:

- A microcontroller may be implemented using two microprocessor cores with one of the cores being defective.
- A microcontroller embedding two cores, one of which is only slightly defective, may exhibit margins to failure much shorter than anticipated in the design.

3.2.3 Embedded Controllers for Flash NAND Memories

As previously discussed, flash (NAND) memories must be accompanied by an embedded controller. The trend in these devices is with integrated solutions, in which the NAND memory, the input/output, and the memory controller are tied together.

In these architectures, the built-in mitigation technique is generally an error-correcting code (ECC) that is resident in the microcontroller. The usage domain and the reduced accessibility to the ECC may cause concern to the AEH manufacturer when considering:

- Whether the reliability of the device properly addresses the AEH avionics application lifetime.
- Whether a specific wear-leveling algorithm is implemented, and it is unknown to the AEH manufacturer.

Wear-leveling methods are used to extend the flash-device life expectancy. These devices are limited to a finite number of program erase cycles. The usage model of the flash memory built by the device manufacturer determines whether wear leveling is needed to extend the device's life to be compatible with the expected embedding system's usage life. Wear-leveling algorithms arrange or store data in a manner that sector erasures are distributed more evenly across the memory array. The system embedding the flash memory uses a flash file system to perform its read and write operations to logical-sector addresses. The wear-leveling algorithm is often part of the flash file system and remaps logical-sector addresses to different physical-sector addresses in the flash array. Wear-leveling remapping can be dynamic or static. The algorithm tracks the sector usage to identify the best area to write data. For the AEH manufacturer, it is important to understand the flash usage model and the impact of the wear-leveling algorithm on the memory array arrangement.

3.2.4 Handling of Errata

As a consequence of the device complexity, exhaustive testing is no longer achievable prior to production. Therefore, the focus is heightened on the importance of continued validation while the device is in use.

The microcontroller manufacturer, the AEH manufacturer, and the end user must work together to support and track this validation. A priori knowledge by the AEH manufacturer of the policy on errata (if any) from the microcontroller manufacturer is recommended. For issues regarding in-service data collection, the reader is directed to the report on service history produced under the SDS program as part of Task Order 3 [20]; previous research on the service history is also available from FAA reports [21–24].

3.2.5 Intellectual Property

The presence of IP has an impact on the performance of development assurance process steps. In particular, the presence of IP may impact the availability of verification artifacts and the availability of the source code for the software assurance reviews.

3.2.6 Spectrum of Microcontroller Devices

Whereas the previous subsections reported on COTS AEH issues, this section focuses on controllers embedded into COTS products (i.e., the focus is shifted from COTS controllers to COTS products embedding controllers).

For microprocessors hosting avionics applications, the applicability of software assurance processes using DO-178C standard and hardware assurance processes using DO-254 was well known.

However, when faced with the large spectrum of microcontroller devices, the applicability of guidance material—whether it pertains to the use of DO-178C, DO-254, or other assurance guidelines (e.g., ISO 26262)—is less clear [16].

This issue is further complicated by the fact that, in some instances, the device manufacturer does not reveal to the AEH manufacturer the existence of a microcontroller in the integrated component. When the determination of the microcontroller's presence is made, it is often late in the product lifecycle so that a redesign or modification to the architecture has a cost and a schedule impact.

If the issue is investigated from the angle of assurance, it presents the problem of how to address the impact of a faulty controller embedded in a COTS product. If the controller is faulty, then its intended function is not correctly performed: This is a system-level safety assessment issue from the viewpoint of the product (whether the fault is identified to the hardware die, the controller hardware functions, or the software executing on the controller). If the assessment of such controller would address the embedded hardware functions at the same time as the software executing on its core, the issues reported in the AFE75 report [7] section 2.17 "Embedded Controllers" may be handled differently.

3.3 GENERIC FAILURE MODES AT LOGICAL LEVEL FOR COTS PRODUCTS

Reference [16] was used in a previous report on COTS commodity memory to present a generic failure model at a logical level. That model is applicable to microcontrollers because they provide information transfer (payload or control). This failure model can be related to commonly used failure modes (see table 1).

Table 1. Relation between microcontroller failure modes and commonly used failure modes

Microcontroller Failure Model Derived From [16]	Commonly Used Failure Mode
Loss of message	Loss
Untimely transfer of message	Erroneous transmission in time
Abnormal sequence of messages	Out of sequence
Untimely or forbidden transition of information	Erroneous data
Impossible transition of information	

This model was recommended because it is easily identifiable with the abstraction level to which it corresponds and because of its ability to demonstrate that the model is complete at this abstraction level.

Failure modes that are associated with the software-hardware interface are defined based on the services provided by hardware for software:

- Inability to get program instruction or data
- Erroneous instruction or data retrieved
- Latency in data delivery (drift in maximum execution time)

3.4 FAULT MODEL AT PHYSICAL AND LOGICAL ABSTRACTION LEVELS

Under the patronage of ARTEMIS joint undertaking, the Verification and Testing to Support Functional Safety Standards (VeTeSS) program investigated fault types and common cause faults found in literature to compile a fault catalogue and develop a reliability knowledge matrix [25]. As the objective of the fault catalogue was to provide the necessary information to model and inject faults in simulations to select the most adapted validation and testing methods, the catalog primarily addresses hardware and the physical abstraction level.

3.4.1 Identification of Failure

In addition to the fault classification in section 3.1, a panel of experts selected from the participants to the VeTeSS program provided a recommendation to add the following concepts to support identifying common-cause failure initiators in microcontrollers' digital components [25]:

- Locality—allows distinguishing between local effects (e.g., gate oxide breakdown) and wider effects (e.g., energy pulses)
- On-chip propagation—indicates how a fault may propagate in the hardware (e.g., via substrate, on the power lines)

- Locality after propagation—allows for assessment of the locality factor based on the ability of the fault to propagate (e.g., a locality assessed as “local” may be reassessed as “global” if the fault propagated through the power lines).

As a result, the fault catalogue is structured according to the following parameters: causes, physical effect and physical effect type, consequence, locality, on-chip propagation, and locality after propagation. Because microcontrollers contain different types of basic components and embed different functions, the fault catalog addresses the effects at:

- Basic component level—SRAM, flip-flops, logic, and NAND flash memory
- Functional part level—core voltage, clock system, analog-to-digital converter, input/output, and reset system

Based on the causal factors, knowledge of the fault and its effect is more relevant at the basic component level (e.g., single-event effects) or at the functional part level (e.g., electromagnetic interference).

3.4.2 Major Findings for Fault Models at Logic and RTL Levels

The following sections summarize the findings and conclusions of various research reports, which analyzed existing literature [25, 26]. The reader is directed to these reports for more details.

3.4.2.1 Permanent Faults

The most common manifestation of hardware permanent fault at the transistor level is stuck-on/stuck-off or short/open. It is less common and more difficult to test for bridging, which refers to a combination of short and open lines. A catch-all fault type is denoted indetermination and is due to either a short in the logic circuit outputs or an opening in the inputs.

The microcontroller logic circuit can suffer from delays associated with a permanent modification of the parasitic capacitances at the transistor level.

3.4.2.2 Intermittent Faults

Faults may first manifest as intermittent before becoming permanent, especially if caused by wear-out. Intermittent faults may occur when certain conditions are encountered (e.g., environmental, functional, or at the interface) that, combined with operation of the device close to its limit characteristics and performance, will trigger faults, revealing a temporary failure or malfunction at the level of the embedding system.

Intermittent faults may become permanent when limit characteristics and performance are exceeded, preventing any proper functioning even within normal operating conditions They may also become permanent when abnormal operating conditions combined with limit characteristics and performance lock up proper functioning beyond robustness features.

Fault models are therefore the same as those for permanent faults.

3.4.2.3 Transient Faults

These faults include soft errors and single-event upsets. As they do not introduce a physical defect in the circuit, they cannot be located spatially. Their duration is limited. The combination of these two characteristics contributes to the difficulty in modeling them for verification and robustness testing purposes.

The most common fault models for transient faults include bit-flip (for storage components), pulse (in the combinational logic), indeterminism, and delay.

High-frequency pulses were found to have a high potential of common cause fault between the functional parts of a microcontroller and their safety mechanisms, particularly lockstep. Reference [19] recommends that vulnerability assessments against electromagnetic pulse be performed at system level using system safety assessment methods.

Alpha particles and single-event effects were found to largely contribute to soft errors, which are the dominant type of faults in modern technologies for microcontrollers.

The fault models recommended for use in verification and robustness testing activities (and listed in ISO 26262 [18]) include stuck-at, bridging, opens, and bit-flips.

4. MITIGATION TECHNIQUES FOR EMBEDDED CONTROLLERS

The mitigation strategy is based on the determination of effects from the failure modes identified in the previous section. Detection mechanisms can be defined based on what they apply to:

- Detection of failures at the output:
    - Direct observations of faults at accessible points of reference using absolute or relative value (fault naturally occurring or injected for test)
    - Remote observations—a fault occurs at some point of control in the device and its effect is checked at a distinct point of observation (fault naturally occurring or injected for test).
- Monitoring of abnormal behavior using microcontroller's internal resources.

4.1 SUMMARY OF EFFECTS OF HARDWARE FAULTS

Fault manifestations at the various abstraction levels for a microcontroller are well detailed in technical publications addressing the representativeness of fault-injection techniques used to verify the design and test for robustness. The issue is that the injection is primarily performed at RTL and logic levels—the literature is much sparser for the functional abstraction level—at which top-down system safety analyses typically apply (e.g., ARP4761 [27]).

Reference [26] investigated the propagation of faults within a PIC16X microcontroller in the presence of various transient and permanent faults. The faults targeted any combinational signal of the ALU and the general clock line of the microcontroller at the electron level. The propagation

was observed in the microcontroller's registers as a corruption for both user registers (accessible to software) and hidden registers (not accessible by software).

The notable results are summarized as:

- Most of the faults manifested as bit-flips. This observation substantiates that the most common mitigation technique in microcontrollers is an ECC. The second-most-common manifestation is indetermination with the preponderance in permanent faults.
- The percentage of propagated faults and their multiplicity[2] increases with the fault duration for transient faults. Therefore, if the clock frequency increases, the percentage of transient faults that propagate it is higher, with an almost-linear dependency between them, whereas there is no dependency on the clock frequency for permanent faults.
- The microcontroller workload impacts the percentage of propagated faults. Complex workloads produce a higher sensitization of faults, for both permanent and transient faults. The workload does not modify the fact that the most common manifestation of fault is bit-flip.
- Many hidden registers are affected by faults at the logic level. Consideration of user registers only for mitigation is not sufficient.

From the registers, the next step is to investigate the propagation from the corrupted registers to the system behavior (e.g., disruption of the execution of the system workload), using the fact that most faults manifested as bit-flips in the registers. The conclusions of the study [26] are:

- Lower failure percentages were observed when the ratio of the fault duration over the time to execute the workload is lower.
- The use of the register plays a significant role in the level of sensitization by the workload. Examples of critical registers include the status register (containing the ALU flags), the instruction register, the constant generator registers, and storage registers for immediate data of ALU calculations. If a register is critical (regardless of whether it is a user register or a hidden register) and sensitized by the workload, the impact of a fault will be higher. In microcontrollers, most of the critical registers are hidden registers; therefore, these registers are likely to play key roles in the fault propagation.

4.2 SUMMARY OF EFFECTS OF SOFTWARE FAULTS

The literature on faults applied to microcontrollers primarily covers physical faults (hardware) and concludes that most of these faults result in bit-flip. Besides a few studies, it is more difficult to find investigations of software faults. However, the first causes of actual loss of function in COTS microcontrollers can be tied to software faults [29].

The following applies to COTS microcontrollers that run complex applications and not to microcontrollers running small and simple programs. Fault models (e.g., orthogonal defect

[2] Multiplicity reflects the average number of registers that a single fault is able to corrupt. The longer a fault lasts in time and the larger the number of registers being corrupted by that one fault; the bigger the multiplicity.

classification) support the classification of software faults that occur during the development phase of the OS. This particular model was developed by IBM.

Examination of real cases indicates that, although software faults are permanent in nature, the errors they generate are similar to those caused by transient faults. For example, the activation of the fault is dependent on the state of the system.

The most critical element is the kernel because it gathers the functional elements providing the various services necessary to run the user applications (e.g., an application programming interface is the most important interface type to provide user services) and to manage the hardware. The drivers used to communicate with the physical environment are also critical, as they run in the same context as the kernel and are implemented based on the services provided by the kernel. One of the issues is that error-detection mechanisms are frequently omitted for performance reasons. The consequence is that a driver may be able to corrupt the kernel internal state. Device drivers are the primary source of fault, and these faults can propagate to the OS [29–30].

Using investigation of software faults in the program execution via fault injection as an example, the failure conditions can be categorized as:

- Time-out—The fault causes the program execution time to change in such a way that it cannot be completed in the specified time.
- OS exception—The fault causes an OS exception.

4.3 OVERVIEW OF BUILT-IN MITIGATION IN MICROCONTROLLERS

In this section, the built-in mitigation techniques in microcontrollers to protect against microcontroller failure are discussed. Mitigations (e.g., wear-leveling or ECC) implemented by the microcontroller to protect an external memory (e.g., NAND flash memory) are not considered.

Microcontroller caches are generally protected by ECC or parity. Error reporting and handling are configurable in the CPU error interrupt enable register. The AEH manufacturer must pay attention to that configuration. Single errors are corrected by the ECC on the transmitted data. Course of action has to be taken by the CPU under dedicated privilege mode (either supervisor or hypervisor, depending on the processor type). The OS is in charge of correcting the error in the cache and restoring the error-reporting register.

Freescale's microcontrollers allows for checking ECC or the parity bit's correct behavior by injecting faults in memories. This capability is key to testing the validity and the correct setting of the integrity-protection mechanism before each operation. This capability is typically implemented in power-on built-in self-tests. In parallel, periodic flushing of the cache provides mitigation to single-event, upset-induced failures.

4.3.1 Memory Management Unit

A memory management unit (MMU) performs a fault-detection activity and typically restarts the component as a one-size-fits-all fault-response strategy. An MMU is a safety feature that is designed to contain the fault caused by a crash of the microcontroller software. However, the majority of microcontrollers are not fitted with MMUs. This is because small programs were run

in the early days of microcontrollers. Currently microcontrollers often include large address spaces and run complex applications.

An MMU would provide graceful degradation (e.g., by detecting access violations) if a task attempted to access a protected chunk of memory outside of the one to which it is allocated. The ARM Cortex-M7 includes a memory-protection unit, which is a crude version of an MMU. Its design allows eight memory regions that are hardware protected from interfering with each other. The access to each region is controlled. Slowly, MMUs are offered on microcontrollers, and the real-time OS adds code to manage it, but it is not yet widely implemented.

In an AEH context, the MMU can be used to implement partitioning of software functions with different DALs (as applicable). However, because the initial use of MMU was not to contain software errors, but rather to organize the virtual memory, it is insufficient to implement robust partitioning, as expected for avionics functions.

4.3.2 Control Flow Checking

Control flow checking (CFC) is one of the most implemented techniques to detect the occurrence of control flow errors. These errors constitute the majority of transient faults in electronic devices (microprocessors and microcontrollers) in remote terminal units (part of most industrial control systems) when operating in harsh environments and submitted to electromagnetic interferences, power supply disturbances, radiations, or high operating temperatures. The effect of such faults is a single bit flip.

Reference [31] summarizes the CFC techniques that have been used on microcontroller-based industrial control systems. These techniques are classified in three groups: hardware-based, software-based, and hybrid. Hardware-based techniques employ a redundant hardware, such as a watchdog (processor or timer) or lock-stepping, to monitor the behavior of the main processor. Software-based techniques employ software redundancy to monitor specific signatures indicative of errors in the program execution. The criteria for selection of the hardware-based technique or the software-based technique include flexibility, cost, overhead, and maintainability. Hybrid techniques represent a compromise between hardware and software CFC techniques to balance cost and overhead.

Signature monitoring mechanisms require extracting an abstract of the program before the system's runtime as a model of correct execution. Signatures from that abstract are assigned to the program and stored. During runtime, signatures are generated in real time and compared with the stored signatures. A control flow error is detected when a disagreement between the signatures occurs (requiring additional instructions in the code) and is reported by the error handler. For more information on signatures, the reader is directed to [31].

Examples of software-based CFC techniques include CFC using software signatures, ECC with assertions, relationship signatures for CFC, intra-inter block control flow checking, and software-based CFC. Examples of hybrid techniques include CFC using branch tree exceptions or a software-based error-detection technique using encoded signatures (SWTES). In SWTES, the software-based component monitors the behavior of a program using an encoded signature,

whereas an on-chip microcontroller timer is used as a watchdog timer to protect against program crashes; it is experimentally evaluated on an Atmel MCS51 microcontroller.

4.4 DESCRIPTION OF ISSUES WITH BUILT-IN MITIGATION TECHNIQUES

Whereas usage errors are addressed by specification robustness, random failures and systematic failures in design and manufacturing process are covered by fault detection and redundancy-based mitigation means.

4.4.1 Accessibility

The executable code in microcontrollers is generally developed by the device manufacturer and is likely not user-modifiable. Consequently, the information and access to the built-in mitigation technique implemented on the device may not be documented at all or may not provide sufficient details for the AEH manufacturer to produce the required artifacts for assurance.

The issue of accessibility is extended to the products of verification and validation, including parameter data items and executable code.

4.4.2 Management of Errata

Design errors are documented in errata, whether accessible to the community or via a non-disclosure agreement between the microcontroller manufacturer and the AEH manufacturer. Design errors do not “age” during the product lifetime. They should be tested, discovered, and mitigated prior to the product entering into service. It is relevant for the AEH manufacturer to control that the most up-to-date errata information is included throughout the design and not only the errata that exist at the start of the development activity.

4.5 IDENTIFICATION OF POTENTIAL ADDITIONAL MITIGATION

In the past, for safety-related applications, external mitigations have focused on the robust deactivation of COTS features. For example, microcontroller input/output complex features have been limited to ground maintenance. Standalone microprocessors with simple input/output have been used for in-flight safety-related applications but with strong internal limiting mechanisms preventing the triggering of the microcontrollers’ full features. Such widely used architectural mitigation consists of implementing a control path that is independent from the functional path in which the microcontroller is involved.

Other architectural mitigations include the implementation of an independent means that would detect the microcontroller failure by diagnosing its output. This method is called fault symptom detection.

5. RECOMMENDATIONS ON ASSURANCE PROCESSES

The notion of embedded controllers ties with a system-level approach, rather than a distinct software-level or hardware-level approach. Software and hardware components within a microcontroller are highly integrated and interdependent. This section focuses on the application of assurance standards, guidance, and system safety methods. Based on the applicability,

recommendations are made for integration of microcontrollers in safety processes and identification of means of compliance.

5.1 APPLICABILITY OF EXISTING STANDARDS AND GUIDANCE

5.1.1 Applicability of Hardware and Software Assurance Standards

The following higher-level regulatory documents are used to introduce the applicability of software and hardware assurance standards:

- EASA CM-SWCEH-001 explicitly addresses expected activities for COTS products in DAL A, B, and C applications in which microcontrollers can be embedded.

- FAA AC 20-152 invokes DO-254 [1] to explicitly apply to custom micro-coded or programmable complex devices but currently does not address COTS microcontrollers.

5.1.1.1 Arguments for Applicability of DO-254

This assurance standard addresses design of hardware components. It is applicable to embedded controllers because it explicitly covers design assurance of COTS products.

5.1.1.2 Arguments for Applicability of DO-178C

EASA CM SWCEH-001 recommended that the development assurance of microprocessors, core processing part of the microcontrollers, and highly complex COTS microcontrollers (e.g., core processing unit) will be based on the application of DO-178B to the software they host, including testing of the software on the target microprocessor/microcontroller/highly complex COTS microcontroller.

No such equivalent exists within FAA's current regulatory and guidance material documents.

5.1.1.3 Other Industry-Based Standards

ISO-26262 [18] part 5 "Road Vehicles – Functional Safety – Product Development: Hardware Level" is the automotive standard for functional safety. This document can be of interest because it introduces development-assurance concepts relevant to complex COTS architectures and addresses mitigation by design of hazards. Microcontrollers are specifically discussed in part 10 of Annex A.

5.1.2 Applicability of System Safety Standards

5.1.2.1 Arguments for Applicability of ARP4761

This guideline document provides a toolset of assessments to determine the failure modes, set safety objectives for the maximum allowable probability of occurrence of the failure conditions associated with the failure modes, and assess the architecture for mitigation. The assessments are scalable to microcontrollers with little modification. Nevertheless, this standard is more a catalog of techniques for safety analyses than guidance for development assurance; the encompassing and

governing guideline for system development assurance is ARP4754A [32], discussed below, and should bear a revision effort.

5.1.2.2 Arguments for Applicability of ARP4754A

This industry standard guideline addresses development issues with complex embedded systems. This document is not limited to digital avionics systems. Because the definition of systems in ARP4754A is fairly wide, it can be considered potentially applicable to microcontrollers. It would benefit from a review for clarification on applicability and a revision to address microcontrollers explicitly.

In particular, if the microcontroller is embedded within another COTS product (e.g., the embedded controller of a non-volatile memory storage device or hard-drive), the system's approach needs to be applied to characterize the microcontroller's intended function, functional failure, and severity classification.

This standard is currently under revision by SAE committee S-18.

5.1.2.3 Other Industry-Based Standards

ISO-26262 [15] part 4 "Road Vehicles – Functional Safety – Product Development: System Level" is the system-level assurance guideline.

5.1.3 Details Within EASA CM SWCEH-001

Section 9.3 of EASA CM SWCEH-001 categorization scheme for COTS products, including microcontrollers, is based on the following criteria:

- Criticality
- Complexity
- Relevance of service history

The microcontrollers' classification scheme is discussed in section 9.3.1 of EASA CM SWCEH-001 as an activity to be conducted for DAL A through C. The result of the activity is a classification into simple microcontroller, complex microcontroller, or highly complex microcontroller.

The assessments to be performed as part of this activity represent the current practice for microcontroller classification within an EASA context. Note that the certification memorandum lists 16 activities relevant to the assurance of microcontrollers. The scope in this white paper is limited to the activities and criteria relevant to the classification of microcontrollers.

5.1.3.1 Criticality: Allocated DAL Criteria

Allocated DAL activity is derived in a top-down approach using SAE ARP4754A [27] guidance and DO-254 appendix B.2 [1]. The assurance activities in EASA CM SWCEH-001 section 9.3.11 through 9.3.13 [12] are tailored for microcontrollers for which the IDAL is allocated as A through C or which failure mode contributes to failure conditions of CATASTROPHIC to MAJOR, regardless of the quantity of relevant service history.

5.1.3.2 Complexity: Functional Architecture Assessment

Using the definitions in section 2.1.1, the objective of the assessment is to classify the microcontroller as simple, complex, or highly complex. This assessment is applicable for microcontrollers for which IDAL is allocated as A through C or which failure mode contributes to failure conditions of MAJOR to CATASTROPHIC, regardless of the quantity of relevant service history.

The assessment investigates the functional architecture of the microcontroller, particularly:

- The description of the functions performed by the device.
- The description of the functional blocks.
- The types of interfaces between functional blocks.
- The description of data processing performed within these blocks.
- The number of functional modes or states for a state machine.
- The type of functional modes.

The assessment is performed to determine whether verification by test can be executed on the physical device for all requirements in all configurations. The artifact for this assessment can be a documented engineering analysis of the device's logic and design.

A microcontroller can be classified as simple if:

- The findings of the previous assessment demonstrate that exhaustive verification by test is achievable.
- The device's logic can be comprehended without the aid of analytical tools (this criterion relates to a "measurement" of simplicity).

Note that other criteria for the assessment of complexity have been proposed, such as documented engineering analysis of the device's logic.

If the microcontroller cannot be classified as simple, it is complex. The microcontroller can be classified as highly complex if any of the following are implemented in the device:

- More than one CPU is embedded, and they use the same bus (which is not strictly separated or which uses the same single port memory).
- Several controllers of complex peripherals are dependent on each other and exchange data.
- Several internal buses are integrated and are used in a dynamic way (e.g., a dynamic bus switch matrix).

5.1.3.3 Relevance of Service History

In an EASA context, service history for microcontrollers is related to documenting:

- Target market (e.g., microcontrollers are very common in automotive systems).
- Operating environment.
- Criticality of usage.

- Total order of magnitude of operating time (classification into low or sufficient product service experience for DAL A through C).
- Evidence of stability and maturity of product (e.g., using rate and type of modifications, rate of occurrence of errata).

These criteria are consistent with the criteria for considering hardware product experience when using alternative methods per DO-254.

5.1.4 Current Practices

The criteria for classifying a microcontroller as simple, complex, or highly complex are solely hardware based and related to the device itself. To paraphrase the criteria in the EASA CM SWCEH-001, a microcontroller will be classified as:

- Simple when demonstration can be made of an exhaustive verification by test of the device.
- Complex when the "simple" cannot be demonstrated, but at least some interconnections between the microcontroller's block components are simple.
- Highly complex when the exchanges between blocks internal to the microcontroller are complex.

5.2 APPLICABILITY OF SAFETY ANALYSES

The safety process is applicable to COTS products, as discussed in reference [16]. The complexity of systems and functions performed by microcontrollers challenges existing safety standards in such a way that a unified analysis framework is needed to regain manageability of the assessment (e.g., IEC 61000-5-9) [19]. However, the issue of accessibility of data and assurance artifacts impacts the reasonably achievable depth of the system-/subsystem-level safety analyses. In particular, ARP4761 appendix A for the Functional Hazard Assessment (FHA) and appendix B for the Preliminary System Safety Assessment require input information on the fault model. Although different implementations exist in the papers surveyed for this research, a common concept emerges: the combination of a top-down analysis to bind the scope of interest for a complementary bottom-up analysis providing the quantitative information on the failures. This approach has been recommended in the context of different COTS: multicore processors, commodity memories, and, in this report, microcontrollers.

5.2.1 Fault Tree Analysis

The qualitative analysis of the vulnerability of a complex microcontroller to electromagnetic pulse is performed using a fault tree structure in [19]. The structure function derived from the fault tree (i.e., probability of failure) is directly compatible with the electromagnetic topology used in all vulnerability assessments. Additionally, mitigation measures at system level, such as shielding, can be readily inserted as AND-gates in the fault tree analysis (FTA). In this specific example, a Bayesian network is used to quantitatively assess the fault propagation model from the combination of the fault tree and the electromagnetic topology using the causal relationship. The vulnerability is characterized by the joint probability of system failure and electromagnetic interference.

Reference [19] provides a quantitative analysis of the probability of vulnerability for three fault propagation scenarios over the three vulnerable nodes of the Bayesian network and computing for each joint probability. In the first scenario, an electromagnetic pulse propagates to the cabling interface and breaks down the first vulnerable node (485 integrated circuit) leading to the failure of the transmission subsystem. In the second scenario, the electromagnetic pulse interrupts the power supply of the second vulnerable node (DC/DC converter), which causes a hard failure in the microcontroller. In the third scenario, the electromagnetic pulse triggers the third vulnerable node (solid state relay), which causes a soft failure in the microcontroller. The joint probability expresses the probability of the electromagnetic interference and the probability of failure of the component. This is illustrated in the fault tree structure as an AND-gate or as a converging structure in the Bayesian network.

A thorough system-level assessment is time-consuming and costly. The application of a system-level method, such as FTA, allows identification of the most significant and measurable effects and informing on failure paths for which further, more detailed, investigation is needed. For failures with environmental sources, the vulnerability function can be determined from the probability of system failure derived from a system safety method, such as FTA, and the stochastic topology of the environmental factor.

5.2.2 Failure Mode and Effect Analysis

The determination of reliability information, such as failure rate, and the refinement of the failure model call for a failure mode and effect analysis (FMEA). The performance of an FMEA or failure mode, effects, and criticality analysis are predicated on the availability of reliability information and effectiveness of mitigation to verify that the design meets the safety objectives. Only a functional FMEA or failure modes and effects summary can be performed at the available level of abstraction of a COTS microcontroller.

Considering the guidance of EASA CM-SWCEH-001 (see the table in section 9.3.13), this activity is justified only for highly complex COTS microcontrollers with low product service experience and DALs A or B, as it often implies a high cost. An additional issue is that FMEAs, based on the device knowledge that they require, are at best difficult to obtain and typically remain the property of the device manufacturer. As a fallback in case the required level of details is not available, the functional FMEA can be performed using hypotheses on the generic internal components, including MMU, ALU, branch processing unit, etc.

5.2.3 Link With Usage Domain Analysis

Usage domain analysis is an integral part of safety assessment applied to COTS products. In the context of microcontroller, the usage domain should be analyzed for functions that may be inactive, for the configuration piece of the microcontroller, and as part of the errata assessment.

5.3 ISSUES TO TRACK FOR ASSURANCE PROCESS

Based on the existing research reports on COTS for AEH [7, 16, 17, 33] and specific issues listed in section 3.2, assurance issues can be grouped into the categories detailed in the sections 5.3.1 through 5.3.4.

5.3.1 Availability of Documentation

The availability of documentation, or lack thereof, on the embedded controller from the device manufacturer presents an issue for demonstration of compliance with safety-assurance processes. In particular:

- The details of the ECC implemented in the microcontroller (e.g., type, coverage, and limitations)
- The device's response to soft errors (e.g., error recovery, containment strategy, and reset mode)
- The number, user-accessibility, and coverage of the configuration modes

5.3.2 Robustness Verification (Fault Injection)

The content of this section is derived from concerns, methods, and tools investigated in the automotive industry to answer safety assurance requirements in the context of compliance with ISO 26262 safety standard [18]. Robustness verification has become a fundamental step in the certification process, and it is a top contributor to cost and schedule. The increased complexity in microcontrollers has aggravated the potential impact of the robustness-verification process on the production stages. The traditional use of fault-injection techniques applied at the gate has become impractical from a schedule standpoint. The industry is investigating increasing the level of abstraction by introducing simulation-based verification. such as RTL and instruction-set simulators [34].

The lack of information on the processor has a direct impact on the ability to perform meaningful fault injection. In particular, the majority of the nodes usable for injection are missing from the processor model. For example, typical instruction set simulator-based fault-injection experiments include injection into the file register because it is required by the safety standards for demonstration of compliance. In the context of microcontroller, these experiments cannot be used to estimate the failure rate because the node is not present in the model. It is therefore impossible to estimate the probability that a fault occurring in any microcontroller net or gate propagates to the register file.

As abnormal behavior can be caused by faults in both user registers and hidden registers, the capability to inject faults in hidden registers requires the application of varied and complementary techniques. For example, the popular software-implemented fault-injection techniques should be complemented with other injection techniques to access hidden registers.

5.3.3 In-Use Validation (Testing in General)

The in-use validation may be the primary source of errata, reliability, and deficiency reports. It is conditioned upon the implementation of adequate monitoring on the microcontroller. Typically, these devices host much fewer monitors than a microprocessor would, based on the safety assurance requirements. This issue rises during development when establishing the coverage of BIT and monitoring capabilities at last recently used level against safety-derived requirements, at validation prior to fielding (e.g., when verifying reliability estimates/predictions), and in service.

5.3.4 Assurance of Software Code and Qualification of Tools

This topic is fairly rich regarding microcontrollers in the context of AEH assurance but also reported for ground vehicle assurance. This concern grew with the allocation of more complex applications to microcontrollers and would benefit from further research. The challenges include issues with availability of code to perform assurance reviews and the lack of information on the development processes implemented by the device manufacturer. It also covers associated issues with the tools that were used to generate the code. These tools are likely not to be qualified.

5.4 SUMMARY OF FINDINGS AND RECOMMENDATIONS

5.4.1 Classification of COTS Microcontrollers

For complex commercial off-the-shelf (COTS) in general and complex COTS microcontrollers in particular, the top objective of a structured development process is the recommended path to certification compliance to be used to ensure the correct functional performance under all foreseeable operating conditions with no anomalous behavior. When these conditions of exhaustiveness are met, the device would be classified as simple, according to the DO-254 definition. This chicken-and-egg issue can be resolved by identifying the classification as simple or complex as resulting from an a priori assessment process used consequently to direct the development process with the adequate design/development assurance strategy. The a posteriori objective is then to target 100% correct performance with a known behavior in all conditions.

Given the unlikely availability of development artifacts on the COTS microcontroller to match the allocated DAL, assurance methods and related activities based on a birds-eye view of the device could be tailored to the DAL. Examples of such tailoring would impact the level of analysis or testing effort to detect the potential defects or errors in the COTS microcontroller.

5.4.2 Analyzing a Device for Usage Domain

The current classification of microcontroller devices is strictly based on the device itself and, therefore, does not call for an analysis of the device's usage domain. However, when considering the issues that were raised with respect to assurance of microcontrollers, an analysis of the device's usage domain seems appropriate.

The assurance activities include:

- The definition of the COTS microcontroller usage domain.
- The verification of the usage domain.
- The validation of the usage domain.

If the microcontroller is analyzed based on its usage, the following findings can be useful to the avionics manufacturer:

- The device's block components can be more or less complex themselves. If a block is not used or is deactivated, it can be ignored in the assurance assessment for the device based on the usage domain analysis that provides assurances that the block cannot be activated during operation.

- The interaction between the processing core and the other functions performed by the microcontroller can be specifically analyzed. The objective of this analysis is to justify that the microcontroller's behavior with respect to the application software is not affected by the other components (this would also justify the discarding of unused or deactivated blocks from the assurance investigation).
- If the above analysis between the processing core and the other functions shows no interdependence, the core can be verified using DO-178C and ARP4754A processes, whereas the rest (i.e., mostly hardware) can be verified using a DO-254 process.
- If there is interdependence, the recommendation may be to analyze the entire device against DO-178C and ARP4754A criteria.

After the analysis on usage is performed, a complex microcontroller (per EASA CM SWCEH-001 hardware criteria) may turn out to be completely manageable. The use and scope of applicability of DO-178C and DO-254 processes can be justified via the usage-based analysis. As a side note, a justification based on component's usage usually increases confidence in the processes and activities to be implemented.

For microcontrollers that have highly complex hardware-based classifications, the following additional activities are identified:

- Analysis of the device's configurability to exclude complex parts that are not used
- Determination of the domain usage

5.4.3 Application of Assurance Processes

The increasing level of complexity in microcontrollers raises the following issues that can be addressed by the application of assurance processes:

- The level of control on the device (overarching concern for COTS)
- The interdependencies between the different hardware and software components of the device, which challenge the notion of control over the useful part in the traditional way
- Because the primary function of a microcontroller is to execute code:
    - Microcontrollers that only contain a processing core are adequately covered by DO-178C assurance process and ARP4754A system integration process.
    - When a microcontroller's architecture exhibits interdependency between the processing core and other components, the adequate coverage described above can be questioned, either based on software or system integration.

If the coverage by DO-178C or ARP4754A can be questioned based on the properties of the architecture (interdependencies in particular), then what is actually questioned is the "way of working" for processing cores. Addressing this issue from a multispecialty (e.g., software, system, and hardware) viewpoint is complicated.

For this recommendation, consider a microcontroller implementing hardware functions in addition to the processing function (i.e., a complex or highly complex microcontroller using the EASA definition).

The controller that is deeply embedded within a COTS product would require the application of ARP4754A (or a system-level safety process) to characterize its intended function(s), its contribution to the COTS functions, the impact of the loss of the microcontroller's functions, and its impact at the COTS product level.

Once these characteristics are defined, the device can be classified from a hardware standpoint, targeting the reachable level of test. Finally, if the processing core is allocated the safety-critical function of the COTS product, an equivalent level of safety needs to be determined in the absence of artifacts that assurance processes have been implemented. However, finding the equivalent level of safety for the software using service history is not simple [20].

The assurance activities to be performed include:

- Hardware-hardware integration.
- Hardware-software integration.
- System-level integration and verification.

The application of system-level safety assessment techniques is straightforward. Based on the classification of the effects of microcontroller failures, mitigation in the architecture can be proposed. As discussed in section 4, various mitigation techniques exist that address specific failure modes of microcontroller components. Such mitigations include redundancy, which, depending on the severity classification of effects, may introduce requirements for dissimilar hardware or software diversity.

5.4.4 Obtaining COTS Microcontroller Data and Documentation

COTS microcontroller data (e.g., datasheets, user manuals, application notes, and associated errata sheets) are available from the COTS supplier to a certain extent. The same remark applies to additional literature and "how-to" information pertaining to the use of the COTS product. Regardless, such data may not be sufficient, in terms of level of details, to demonstrate correct behavior in all foreseeable conditions. In addition, the avionics manufacturer must rely on the COTS supplier to ensure that the data reliably reflect the actual characteristics and behavior of the device. One specific problem is with proprietary data retained by the COTS supplier for a variety of reasons.

To overcome the issue, the following approach is recommended:

1. Capture the COTS microcontroller artifacts as formalized requirements, interface descriptions, and design data (see note) from all available sources of information.
2. Incorporate the above COTS microcontroller artifacts within the overall design at the circuit board assembly level, either via traceability or matching analyses.
3. Perform the related requirements-based validation and verification activities, as if the COTS microcontroller were designed with a requirements-based process.

Note: Design data may include configuration settings, activation of used/unused functions, internal control, or monitoring mechanisms.

Other activities include the formalization of a process to address COTS errata:

- Capture of errata sheets from the COTS supplier
- Performance of an impact analysis of the errata sheets
- Definition of a process to follow-up on the errata

To address the configuration management process, the following activities should be considered:

- Monitoring the COTS microcontroller configuration
- Managing the change notices related to the device
- Performing the impact analysis on the change notices

5.4.5 Means of Compliance

When COTS microcontrollers provide safety-critical functions (i.e., when they contribute to catastrophic or hazardous events), fault detection and mitigation activities are recommended regardless of their classification (e.g., simple, complex, or highly complex) and irrespective of the product service history.

The proposed process relies on different patterns and methods including:

- The choice of a failure model at logical abstraction level.
- A preliminary black box description focusing on microcontroller output that includes:
    - The description of the outputs to be considered in the failure mode and effect assessment.
    - The analysis of the selected output failure modes.
    - A list of input/output level integrated detection/mitigation mechanisms.

This step provides the internal failure mode from the point of view of the output signals.

- A second step at grey box breakdown level that covers:

  - The selection of an architecture.
  - The description of blocks to be considered in the analysis.
  - The analysis of failure modes at block level.
  - A list of internal detection/mitigation mechanisms.

This step provides insight on the microcontroller's internal failures that can impact various outputs.

The grey box approach appears important to understand as well as possible the internal behavior of the microcontroller and its logical structure (block breakdown). However, its physical structure remains generally nonaccessible to the airborne electronic hardware manufacturer. The analysis of various microcontrollers suggests that few patterns generate most of the systematic failures.

The use of tests and the determination of mitigation means can be split into three categories:

1. Mechanisms relying on internal mitigation means of the microcontroller
2. Mixed mechanisms relying on detection means internal to the microcontroller and on external mitigation means
3. Mechanisms fully relying on architectural means for detection and mitigation

Establishing a comprehensive failure model is key, as a missed failure mode may cause the invalidation of the analysis at both black-box and grey-box levels, and the judgment on the detection mechanisms' coverage. At grey-box level, the selection of the microcontroller internal architecture also appears to be a tactical choice for the relevance of the failure analysis.

System-level safety analyses apply to microcontrollers and include functional hazard assessment, system safety analysis, fault tree analysis, and failure mode and effect analysis.

7. REFERENCES

1. RTCA Report. (2000). Design Assurance Guidance for Airborne Electronic Hardware. (DO-254).

2. RTCA Report. (2011). Software Considerations in Airborne Systems and Equipment Certification. (DO-178C).

3. FAA. (1988, June). Advisory Circular 25.1309-1A. *System Design and Analysis*. (updated with an Arsenal version). Washington, D.C.: Government Publishing Office.

4. FAA. (2011, November). Advisory Circular 23.1309-1E. *System Safety Analysis and Assessment for Part 23 Airplanes*. Washington, D.C.: Government Publishing Office.

5. FAA. (2014, July). Advisory Circular 27.1309 (within AC 27-1B). *Certification of Normal Category Rotorcraft*. Washington, D.C.: Government Publishing Office.

6. FAA. (2014, July). Advisory Circular 29.1309 (within AC 29-2C) *Certification of Transport Category Rotorcraft*. Washington, D.C.: Government Publishing Office.

7. FAA Report. (2016). AFE75 COTS (AEH) Issues and Emerging Solutions Report. (DOT/FAA/TC-16/57).

8. FAA Report. (2011). Handbook for the Selection and Evaluation of Microprocessors for Airborne Systems. (DOT/FAA/AR-11/2).

9. FAA Report. (2011). Microprocessor Evaluations for Safety-Critical, Real-Time Applications: Authority for Expenditure No. 43 Phase 5 Report. (DOT/FAA/AR-11/5).

10. FAA Report. (2010). Microprocessor Evaluations for Safety-Critical, Real-Time Applications: Authority for Expenditure No. 43 Phase 4 Report. (DOT/FAA/AR-10/21).

11. FAA, Software and Electronic Section – COTS AEH Assurance Methods – Phase 3 (Commodity Memory and Embedded Controllers), Delivery Order #4 Statement of Work, 01 August 2014.

12. EASA Report. (2012). Certification Memorandum. Development Assurance of Airborne Electronic Hardware. Iss. 01, Rev. 01. (CM SWCEH-001).

13. Freescale Semiconductor, (2006, January). MPC7447A Technical Data. Retrieved from nxp.com.

14. Freescale Semiconductor, MPC8610 Datasheet/NXP. (2016, April, 17). Retrieved from nxp.com.

15. FAA Order 8110.105, Simple and Complex Electronic Hardware Approval Guidance, Change 1 (2008).

16. EASA Report. (2013). COTS-AEH – Use of Complex COTS (Commercial Off-The-Shelf) in Airborne Electronic Hardware – Failure Modes and Mitigation. (EASA.2012.C15).

17. EASA Report. (2008). SoC Survey Report – Safety Implications of the use of system-on-chip (SoC) on Commercial Off-the-Shelf (COTS) Devices in Airborne Critical Applications. (EASA.2008/1).

18. ISO-26262:2009: Road Vehicles – Functional Safety.

19. Congguang, M., Canavera, F. G., Cui, Z., & Sun, D. (2016). System-level vulnerability assessment for EME: From fault tree analysis to Bayesian networks – Part II: Illustration to microcontroller system. *IEEE Transactions on Electromagnetic Compatibility, 58*(1), 188–196.

20. FAA Report. (2016). Final Report for Software Service History and Airborne Electronic Hardware Service Experience in Airborne Systems. (DOT/FAA/TC-16/18).

21. FAA Report. (2001). Commercial Off-The-Shelf (COTS) Avionics Software Study. (DOT/FAA/AR-01/26).

22. FAA Report. (2001). Review of Pending Guidance and Industry Findings on Commercial Off-The-Shelf (COTS) Electronics in Airborne Systems. (DOT/FAA/AR-01/41).

23. FAA Report. (2002). Software Service History Handbook. (DOT/FAA/AR-01/116).

24. FAA Report. (2002). Software Service History Report. (DOT/FAA/AR-01/125).

25. Analysis of Fault Types and Common Cause Faults/VeTeSS: Verification and Testing to Support Functional Safety Standards. (2015, February, 2012). Deliverable D5.2. Retrieved from www.vetess.eu.

26. Dependability Benchmarking project, Fault Representativeness report ETIE2, IST-2000-25425, June 2002.

27. SAE International (1996). Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment. (ARP4761).

28. Lee, I., Iyer, R.K., “Software Dependability in the Tandem GUARDIAN System,” IEEE Transactions of Software Engineering, 21(5): 455-467, 1995.

29. DOT/FAA/AR-02/118, Study of Commercial Off-The-Shelf (COTS) Real-Time Operating Systems (RTOS) in Aviation Applications.

30. DOT/FAA/AR-03/77, Commercial Off-The-Shelf Real-Time Operating System and Architectural Considerations.

31. Rajabpour, N., Segaghat, Y., “A Hybrid-based Error Detection Technique for PLC-based Industrial Control Systems,” 2015.

32. SAE, “Guidelines for Development of Civil Aircraft and Systems,” ARP4754A, December 2010.

33. FAA Report. (2017). Commercial Off-the-Shelf (COTS) Airborne Electronic Hardware (AEH) Assurance Methods – Phase 3 – Commodity Memories. (DOT/FAA/TC16/40).

34. Espinosa, J., Hernandez, C., Abellla, J., De Andres, D., & Ruiz, J.C. (2015). *Analysis and RTL correlation of instruction set simulators for automotive microcontroller robustness verification.* Proceedings from the 2015 DAC Conference, San Francisco, CA.

APPENDIX A—GLOSSARY

This report uses the following definitions of technical terms and architectures:

Commercial off-the-shelf component (COTS) [RTCA/DO-254]

Component, integrated circuit, or subsystem developed by a supplier for multiple customers, whose design and configuration is controlled by the supplier's or an industry specification

Complex hardware item [RTCA/DO-254]

A hardware item is identified as simple only if a comprehensive combination of deterministic tests and analyses appropriate to the design assurance level can ensure correct functional performance under all foreseeable operating conditions with no anomalous behavior. When an item cannot be classified as simple, it should be classified as complex.

Complex COTS microcontroller [EASA CM-SWCEH-001]

Any integrated circuit or electronic hardware item that executes software in a specific core area (central processing unit) and implements complex peripheral hardware elements, such as input/output bus controllers

Highly complex COTS microcontroller [EASA CM SWCEH-001]

A microcontroller should be classified as highly complex as soon as it has any of the following characteristics:

- Multiple central processing units are embedded, and they use the same bus (which is not strictly separated or which uses the same single-port memory).
- Several controllers of complex peripherals are dependent on each other and exchange data.
- Several internal buses are integrated and are used in a dynamic way (e.g., a dynamic bus switch matrix).

Simple COTS microcontroller [EASA CM SWCEH-001]

Any electronic item that executes software in a specific core area and implements simple peripheral hardware elements, such as universal asynchronous receiver/transmitter (UART), analog to digital (A/D) converter, and digital to analog converter (D/A).

COTS controller [derived from COTS microcontroller]

Any integrated circuit or electronic hardware device that does not execute software in a specific core processing unit but implements miscellaneous electronic hardware functions.

Note: A COTS controller can be assessed as simple (e.g., UART, A/D, or D/A converters; pulse-width modulator) or as complex (e.g., controller area network bus interface, direct memory access, memory management unit, input/output controllers).

Intellectual property [derived from EASA System-on-Chip report]

In electronic devices, an intellectual property (IP) or intellectual property core (IP core) is an electronic function designed to be reused as a portion of a device (e.g., COTS, application-specific integrated circuit, or programmable logic device).

The following terms are introduced based on discussions with specialists of packaging who emphasized the need to be precise.

Chip – Synonym of die

Component – The complete component embedding the die in a package

Die – The piece of semiconducting material constituting the integrated circuit

Device – Elementary structures such as transistors, capacitors, etc.

Electronic module – Electronic module, or simply module, for which there is no ambiguity; refers to a part of airborne electronic hardware equipment that contains electronic components. An electrical module may be a mezzanine card, an electronic board, a set of related electronic boards, a computer, etc.

Interconnect – Metal layers connecting the devices within the die

The following definitions of failure-related terms are used in this report:

Hard error – A hard error implies that the affected element is definitively broken, regardless of the effects.

Intermittent effect – An effect that is sometimes present, sometimes absent, with no limit in time

Permanent effect – An effect that is continuous in time

Soft error – A soft error means that the affected element currently exhibits a bad behavior but is not broken and will resume working after being refreshed.

Transient effect – An effect that exists only for a limited duration (it will exist again only if it is triggered again)