AVOIDANCE OF DYSFUNCTIONAL BEHAVIOUR OF COMPLEX COTS USED IN AN AERONAUTICAL CONTEXT

ÉVITEMENT DES COMPORTEMENTS DYSFONCTIONNELS DES COTS COMPLEXES DANS UN CONTEXTE AÉRONAUTIQUE.

Vincent Brindejonc et Anthony Roger
Thales Avionics
18 Avenue du Maréchal Juin
92360 Meudon la Forêt
vincent.brindejonc@fr.thalesgroup.com
anthony.roger@fr.thalesgroup.com
(+33)1 39 45 57 92

Summary
This publication proposes a method in order to study and contain dysfunctional behaviours of complex Commercial Off-The-Shelf (COTS) components used in safety-critical civil aeronautical context. Behaviours will be considered as dysfunctional if COTS expected electronic functions are not performed at the required level. These behaviours consist in limit behaviours unacceptable from safety point of view in certain context, and faulty behaviours caused by design errors which will be considered as dysfunctional whatever the context is.

Résumé
Cette publication propose une méthode d'étude des comportements dysfonctionnels des composants sur étagère (COTS) complexes utilisés dans un contexte sécuritaire aéronautique civil. Un comportement sera considéré comme un dysfonctionnement dans un certain contexte si certaines des fonctions électroniques matérielles attendues du COTS dans ce contexte ne sont pas réalisées au niveau requis. Ces comportements dysfonctionnels recouvrent, d’une part, des comportements aux limites inacceptables d’un point de vue sécuritaire dans certains contexte d’utilisation, et d’autre part des comportements défaillants dus à la présence d’erreurs en particulier de conception et qui apparaîtront dysfonctionnels quel que soit le contexte d’utilisation.

Introduction and context

1. Background
Avionics computers are subject to more or less stringent safety-critical and real time constraints depending on the function in which they are involved: from no safety constraint and high real time constraints in the case of In-Flight Entertainment (IFE) applications to very high constraints both safety and real time in the case of autopilot.

Leaving aside IFE, avionics applications can be split into two domains. In both domains use of complex Commercial-Off-The-Shelf (COTS) components has taken a great importance with some differences among them.

In a first domain, safety-critical and real time constraints are harsh (e.g.: flight guidance, autopilot, etc.). These applications are in general realized through federated architectures namely, architecture in which a computer is dedicated to only one application) and do not require high computing performance. In this domain, use of complex COTS remains moderate even if complex microcontrollers are unavoidable.

A second type of applications has slightly lower real time constraints (e.g. cockpit flight displays, flight warning system, fuel management, etc.) and is more and more allocated to integrated architectures requiring state-of-the-art computing performance. Consequently, last generation of complex COTS and in particular of microcontrollers are used in Integrated Modular Avionics (IMA) architectures.

These complex or highly complex COTS are in general (see Bieth, Brindejonc-2013):
• Bridges and switches among some buses on the board;
• Specialized bus/network interface drivers such as CAN, Arinc 429 or MIL-STD-1553;
• Some memories such as NAND flashes that includes a controller for wear levelling support;
• Complex and highly complex microcontrollers that implement complex peripheral hardware elements and multicore microcontrollers.

During airborne electronic hardware development, DO-254 (EUROCAE&RTCA, 2000) standard shall be followed in order to avoid systematic failures. Complex or Highly Complex COTS are obviously not developed using such standard. Thus, certification agencies led by the European Aviation Safety Agency (EASA) and the Federal Aviation Administration (FAA) have settled additional rules in order to accept complex COTS as a certifiable device in a civil aeronautics context. These rules are respectively reported in EASA Certification Memorandum SWCEH 001 (EASA, 2012) and FAA Order 8110.105 (FAA, 2008). In particular, (EASA, 2012) requests many precise actions in order to master the COTS and to ensure the robustness of the COTS manufacturer development process. Moreover, for the highest safety level, additional clauses are requiring architectural mitigation in order to avoid COTS design errors.

1 Even if not mentioned in the following sections, the term COTS will designate only Complex and Highly complex COTS.
Main advantages of using COTS components are quick availability (for a reasonable period of time), reasonable cost and service experience of other embedded system designers who are also implementing this COTS. Main drawbacks of the COTS are first their lifetime which could be shorter than the lifetime of the aircraft and so imply expensive obsolescence management. The second drawback is continuously increasing complexity of COTS microcontrollers which are leading to additional certification activities which shall be done to master this complexity.

Since several years the development of semiconductor market led to an increasing complexity of these COTS components. This increasing complexity has two folds: firstly a race for pure performance and secondly an increase of integrated functionalities which are leading to additional certification activities which shall be done to master this complexity.

For (EASA, 2011) A microcontroller should be classified as Highly Complex as soon as it has any of the following characteristics:
- More than one Central Processing Unit (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 (for example, a dynamic bus switch matrix).

This increasing complexity has two folds: firstly a race for pure performance and secondly an increase of integrated functionalities in order to be compliant with a maximum of various customers’ needs (for instance, Ethernet frame analyser for the network domain, graphic accelerator or audio decoder for multimedia domain, etc.)

If in one hand, the performance increase can be in line with the needs of avionics applications, in the other hand the complexity increase due to more integrated functionalities makes the safety and certification demonstration more difficult for the avionics computer designer who is embedding such COTS.

2. Example of a complex COTS microcontroller

As already outlined, microcontroller is the cornerstone of avionics computers. This is the reason why we decided to take a recurrent example all along this article. This example is one of a highly complex microcontroller according to (EASA, 2012). The COTS microcontroller will help illustrate our proposed approaches and methodologies.

Thus, from a generic perspective, COTS microcontrollers contain Intellectual Properties (IP) block which can be classified into 3 groups from a transaction point of view (see Figure 1). First, there are IPs which can be initiators of transactions and so we called this group initiator. It includes CPU (Central Processing Unit) cores, DMA (Direct Memory Access) and other initiators such as GPU (Graphical Process Unit) cores and other embedded hardware accelerators. The second group is the target one since they can receive transactions. It can contain any kind of built-in memories and memory interfaces (RAM or ROM) and I/O interfaces (from low-speed interfaces such as SPI, I2C to high-speed interfaces such as PCIe, SRIO and Ethernet). Finally, the last group is the one of the unused IPs which gather all IPs which should be disabled and not used for a specific computer.

In this article, we chose to select a multicore COTS microcontroller, (depicted on Figure 2) implementing 3 initiators (2 CPU cores -Central Processing Unit- and 1 DMA -Direct Memory Access- controller) and 2 targets (1 DDR 3 memory controller and 1 PCIe controller). This kind of microcontroller constitutes the state-of-the-art of highly complex COTS which are used in current avionics computer development.

3. Problematic

COTS usage in avionics domain faces several decisional challenges mainly due to the uncertainty caused by many behavioural possibilities (i.e.: Either functional or dysfunctional). The decision challenge is thus

---

2 Intellectual Property blocks are in general designed independently from a given COTS and integrated by the COTS manufacturer.
3 PCIe is the new standard for intra and inter board broadband exchanges in particular in PCs. It is normalize by PCISig.
4 SRIO generally stand for Serial Rapid Input Output
• Which level of analysis, test, detection and mitigation means should be necessary and sufficient in order to reach an admissible level of confidence?

The present publication aims at classifying dysfunctional behaviours that can be encountered during COTS studies and consequently help the decision making. These dysfunctional behaviours are of several types:

- Random hardware failures occur even if the development, manufacturing or operation respects the state-of-the-art. There is no general solution to model them.
- Systematic failures resulting from a lack of respect for the state-of-the-art during development, manufacturing or operation. By nature, they cannot be modelled by probability laws and thus any coverage is difficult to define at least with respect to a failure occurrence.
- Systematic failures result from a lack of respect for the state-of-the-art during development, manufacturing or operation. By nature, they cannot be modelled by probability laws and thus any coverage is difficult to define at least with respect to a failure occurrence.

New concerns in safety demonstration thus become, firstly to avoid limit behaviours non-compliant with real time constraints and secondly to ensure, up to some level of confidence, the absence of errors in COTS design.

### Dysfunctional modes of embedded systems

1. **General approach**

   Breakdown and abstraction levels

   Dysfunctional modes of embedded systems and complex COTS can be analysed at different levels of breakdown and independently at different levels of abstraction.

   In term of breakdown, three levels can be identified:

   1. **A white box level** where, the considered failure modes are those of internal detailed blocks of the COTS and potentially of each gate. Knowing that such a full white box down to the gate level is never achievable and if achievable, impossible to manage in term of complexity, one difficulty is to determine which level of details would be sufficient to allow adequate mastering of both the functional and dysfunctional behaviour of the COTS and reach acceptable coverage of potential internal failures or errors.

   2. **A grey box level** in which the COTS is modelled with interconnected generic blocks. The COTS model is not fully representative of the component itself. This breakdown level is chosen sufficiently detailed in order to be heuristic and not too detailed in order to be free of property rights. (Bieth, Brindejeons 2013) define a Grey Box as a Black box refined breakdown level, built with fragmentary, non-contractual and potentially under Non-Disclosure Agreement (NDA) information.

   3. **A black box level** where the failure modes are those of the output flows of the COTS. This breakdown level allows identifying a set of transaction initiators and transaction targets within the COTS. Thus COTS failures correspond to the failure of these transactions with respect to some qualification view related to the usage needed by the COTS.

Objects defined at previous breakdown levels can also be considered at various abstraction levels:

1. **A Functional level** at which is considered the Hardware Transfer Function (HTF): for instance delivering a coherent signal to the different Software levels. These "electronic" functional detection/mitigation means are addressed here.

2. **A Logical level** that considers the output of the COTS (at black box level) or of the IP blocks (at grey box level) from their global logical content point of view (For instance a 32 bits output signal). It is the most adapted level on which it is possible to catch the failure modes of the COTS and to identify the possible detection/mitigation means. Hardware Transfer Function (HTF) refers to the description of the service provided by a hardware block to connected hardware blocks or to an environmental media, for instance the software.

3. **A Physical level**. Where COTS output (or IP block outputs) are separated and signals are detailed up to physical characteristics (voltage, current, timing, etc.). This abstraction is certainly too detailed for an effective description of failure modes on a complex COTS due to the number of elementary signals to consider. Nevertheless it can be interesting in some particular cases like clock, power supply or reset distribution and when relevant internal fuse distribution are present within the COTS.

   Failure model at logical level

5 On some modern COTS some IPs can be permanently deactivated by fuse. That can be the case for instance for security-related IP concerned by Export Control clauses.
At logical abstraction level it is possible to define a finite model of failure mode (see Brindejonc et al., 2010) for a transaction at black or grey box breakdown level. Figure 3 presents the various possible behaviours that a transmitted message and carrying some informational content can exhibit.

**Dysfunctional behaviour of a COTS**

The fault model defined rigorously at logical abstraction level can be extended at functional level to the hardware transfer functions (HTF) in order to exhibit its dysfunctional behaviour relatively to a particular usage. For instance, considering a particular Hardware Transfer Function characteristic of a microprocessor: “Realize computation required by the software” corresponding to:
- Get software instruction,
- Get Data for computation,
- Push computation results.

The corresponding dysfunctional behaviour can be defined as:
- Loss of fetched instruction or input data,
- Untimely transition of instruction or data got to another valid instruction or data set.
• Forbidden transition to an erroneous instruction or erroneous data,
• Impossible transition to the subsequent instruction or data set,
• Untimely data delivery (Maximum Execution Time drift),
• Inversion of two outcome instructions or data.

When several applications are embedded in the same microprocessor, another HTF appears, related to the non-interference of these applications (partitioning), that will not be detailed here. This dysfunctional behaviour can have two origins:
• A failure of the COTS either random or systematics. It appears through analysis that failure of the COTS may generate all dysfunctional behaviours listed before;
• An inadequacy between the normal behaviour of the COTS in some particular functioning modes and the minimum performance required by the application. This cause of dysfunctional behaviour is present in our safety-critical context only for "Untimely data delivery (Maximum Execution Time drift)". Indeed COTS selected for this type of usage are not authorized by their specification to lose or corrupt data. In another hand since many years the COTS market development is characterized by significant improvements in computation characteristics for most of the applications. These improvements are statistic and not deterministic. It is thus very important in real-time, safety-critical applications to be able to characterize the real-time performances of the COTS in a deterministic way and not only statistically.

From previous discussion on causes of dysfunctional behaviours, it runs out that in order to use a particular COTS in real-time safety-critical applications, use of the COTS should be restricted to some domain called “Usage Domain” that have the following requirements:

1. Real time performance of the COTS restricted to its usage domain shall be guaranteed and consistent with the overall real-time constraints associated to the real-time constraints of embedded applications;
2. COTS restricted to its usage domain shall be exempt of systematic failures;
3. The usage domain shall be stable –i.e. the COTS shall not run away from this usage domain--.

These requirements assure a safe and deterministic usage of the COTS.

It can be noticed that requirements 2 and 3 are relative to safety and consequently are absolute. The first requirement has not the same status because it is partly relative to embedded applications needs: A COTS may be suitable for an IMA type usage because its real-time performances are consistent with embedded applications response time but it can be unacceptable for Flight guidance or autopilot usage because its real-time performances are not sufficient in these harsh real-time applications.

In the forthcoming sections COTS are studied at black-box and grey-box breakdown level through tests and analyses. Black box studies are expressed in term of an initiator-target formalism that allows in particular a simple classification of cases to be studied and tested at this breakdown level in order to guarantee real-time performances (req. Nb. 2 here before). In order to study the potential failures of the COTS a grey-box approach is developed that systematically scan the potential failure modes.

Figure 5 : A typical Initiator-Target model

Figure 6 : Two examples of single-path test classes (in red)

Usage domain: limitation of usage of a COTS to
• Only a part of the COTS IPs;
• Restriction of capacities of the selected IPs.

6 As already mentioned authors considers that random failures are covered by other techniques (e.g. FMEA) and are out of scope of the present communication.
2. Black-box approach based on the initiator-target model

The primary goal of the initiator-target model is to bring a theoretical method on which the developer can rely in order to define a deterministic performance usage domain of the system under consideration. The performance usage domain is a domain where the predictability of the system performance has been validated and is respecting pre-defined limits. Outside this domain, performance figures are no more guaranteed since low to high performance variations have been observed. We call this phenomenon, performance contentions, and can be observed as soon as more than one initiator is running in the system. Of course, this model can be applied onto any kind of systems where entities (that is to say initiators or targets) are exchanging information. In our case, we have applied this model onto COTS processors in order to master the growing complexity of such chips. Indeed, inside a unique chip, a lot of functionalities and components are embedded and can interact between each other. And also not all these embedded components are fully documented which make the analysis more difficult.

In first development steps, the initiator-target model was applied on the COTS processor considering a black-box approach. By saying black-box approach, one can argue immediately that it is not possible to detect the precise location of possible performance bottlenecks. This is true, nevertheless correlations between various tests which detected performance contentions could be done in order to investigate potential common high-level components. In this perspective, it is interesting to note that we go down from the black-box approach to the little more detailed grey-box approach in order to better understand and investigate the dysfunctional behaviours.

However, as current COTS processors have very limited points of observability and controllability, the black-box model should be kept as much as possible. But for investigation purpose of possible contentions and bottlenecks, the grey box model should be privileged in order to locate the possible limiting high-level components of the COTS (For instance, high-level components are I/O controllers, memory controllers, interconnects, etc…) or identify design failures (see next section).

Formalization of the initiator-target model

First, let’s define some terms. A test class is a group of unit tests which have the same initiators and targets activated. Now, let’s formalize the initiator-target model. The first goal is to enumerate all the possible test classes which can be considered by an implementation of a specific COTS processor.

Here is the formula which gives the total number of test classes that can be tested:

\[
\sum_{k=0}^{n_{si}} \binom{n_{si}}{k} \cdot \sum_{k=0}^{n_{nsi}} \binom{n_{nsi}}{k} (n_{t}^2)^{1-k} - 1 = (1 + n_{nsc})^{n_{nsi}} (1 + n_{t})^{n_{t}} + 1
\]

(1)

Where,

- \( n_{si} \) is the number of smart initiators that is to say initiators which have built-in memories (for instance, processing cores);
- \( n_{nsi} \) is the number of non-smart initiators that is to say initiators which do not have built-in memories (for instance, DMA needs both a target source memory and a target receiving memory to perform a transaction);
- \( n_{n} \) is the number of targets.

This formula is valid under the following hypotheses:

- Each target must be reachable by each initiator both smart and non-smart
- One smart initiator addresses no more than one target. In other words, the multicast / broadcast from one smart initiator is not allowed (see Figure 6);
- One non-smart initiator only has a couple of target at a given time: no multicast/broadcast allowed;
- Non-smart initiators are directional: we distinguish both directions transfers from target 1 to target 2 and from target 2 to target 1;

Finally, it is interesting to note that some simplifications may be brought by symmetry principles: when at least two initiators or targets are of the same type. For instance, two initiators can be two instantiations of the same processing core component and so symmetric axes can be leveraged in order to not run all test classes.

Moreover, it is important to note that this formula enumerate all possible test classes from single-path transactions (only one couple of smart initiator-target) to multiple path transactions (double, triple …).

Example of a initiator-target model with 3 initiators and 2 targets

For a 3 initiators-2 targets model where we consider 1 non-smart initiator, we calculate the total number of test class to consider:

\[(1 + 2)^2 \cdot (1 + 2)^2 - 1 = 9 \cdot 5 - 1 = 44\]

In this use case, there are 44 test classes which need to be considered in order to be exhaustive.

Among these 44, 4 test classes are called single-path transactions (Figure 7). These test classes are those where only one (smart) initiator-target couple is active.
These single-path transactions serve at setting the maximum reachable performance (latency, throughput) when no performance contentions are present.

Eight (8) test classes are called double-path test classes. Four (4) double-path test classes are the performance reference test classes for the non-smart initiator (Figure 9) whereas 4 double-path test classes are couples of single-path test classes (Figure 8). These latter test classes are those necessary to test if performance contentions are present for smart initiators.

Next, 16 test classes are called triple-path test classes (Figure 10). These test classes try to identify possible interference channels when one smart initiator and one non-smart initiator are simultaneously activated.

Finally, 16 test classes are called quadruple-path test classes. These test classes occur when both smart initiators and the non-smart initiator are activated at the same time. These quadruple-path test classes are the second order priority test classes necessary in order to test if performance contentions are present. In this example, quadruple-path test classes are the maximum order of path since there are two smart initiators and one non-smart initiator and since no multicast/broadcast are allowed. The generalization of this observation is that the maximum path order is equal to the number of smart initiators plus 2 times the number of non-smart initiators.

3. Grey-box approach: Analysis of COTS failure modes

Initiator-Target method allows restricting the use of the COTS in a usage domain in which the functional performance of the COTS are guaranteed in all conditions. Such a domain can be impaired by COTS failure. Among them, systematic design failures are in general:

- **Unpredictable** like all systematic errors;
- With some threshold effect because internal COTS mechanism may mask the prior effects that can only be perceived as weak signals up to some accumulation of errors that then spread out the complete COTS;
- **Catalectic** in the sense that after the threshold crossing, all IPs of the COTS can be contaminated with erroneous data and then generate wrong outputs.

It is thus strategic to manage these potential failures and first to identify them.

The impact of design systematic failures can be two folds:

- In a first case, failures can occur that violate the frontier of the usage domain. They are mainly failures that modify the COTS configuration in order to define the usage domain or spurious failures, at physical level, leading to discrepancy between COTS configuration observed through configuration registers and the configuration observed through COTS behaviour. Figure 11 presents these two mechanisms.
- In a second case some failures may cause loss, delay and repeated sending of messages or untimely, forbidden or impossible transition of information even if message is correctly sent. These failures occur without violation of the usage domain that remains unchanged but the behaviour of the IPs included in the domain is out of specification.

Applying the failure pattern to the communication between IPs selected in the usage domain help to discover the potential failures that the COTS may develop.
Considering the microcontroller of Figure 2, we will concentrate on an example of an interconnect IP that is the most buried block in the COTS in the sense that it is never observable directly by external probes. Typical interconnects appear to be constituted of switched matrix of buses connecting initiators and target through several different possible paths and with internal buffers and memories allowing to store messages when the data-path is still occupied and to retry transactions when it appears that they did not reach their target. Due to this structure, interconnects may develop all kind of failures exposed in section 2.1.2.

- **Loss of Messages** can occur in buffers and internal memories. It can take the two following forms:
  - Loss of data type messages: In addition to direct loss of message, it can occur if addresses are corrupted or lost;
  - Loss of instruction type message: Interconnect transfer program instruction from flash memory to main memory during the boot and between main memories and caches during operational phase. It is possible to lose instruction during these transfers.

- **Untimely transfer of message.** Typical retry processes implemented in order to avoid message loss could lead to babbling. Interconnect contains buffers that could store transaction during few clock pulses in order to wait that a way is available. The presence of such buffer could also generate untimely resending of messages in case of pointer error.

- **Abnormal sequence of messages.** The two afore mentioned mechanisms, which could lead to untimely transfer of information, could also lead to possible abnormal sequence. In particular a loss of message compensated by a retry could make the corresponding message arriving after a message sent before it. In another way, a buffer pointer error could invert two messages.

- **Untimely or forbidden transition of information:** An information (address, data, instruction) stored in a buffer could suffer from a bit flip and then to an untimely transition.

- **Impossible transition of information:** Equally to the bit flip phenomenon, a stuck-at bit phenomenon may occur in buffers or directly in data-paths leading to an impossible transition of information.

Such abnormal behaviours shall be avoided and/or detected / mitigated in order to guarantee a safe functioning of the computation platform implementing the COTS. The approach is then twofold. First, explore the way to guarantee a qualitative high level of Reliability\(^7\) of the COTS. Second to detect and mitigate failures that may still occur.

### Avoidance, Detection and Mitigation principles

#### 1. Reliability demonstration

The first approach tries to improve the confidence level that the embedded system designer can have in the COTS. By definition, the COTS development quality cannot be controlled by the embedded system designer or associated to any assurance level (DAL, SIL, etc.) as it is not in general developed in a safety-critical context. However COTS chosen in safety-critical context offer some assurance based on the credit experience from multiple usages and the development quality of COTS manufacturers. Certification organisms have formalized these aspects (see for instance EASA, 2012) by reinforcing a posteriori controls and analyses of COTS documentation including errata sheets. These analysis activities can improve confidence but does not bring full guarantee that the COTS is free from design errors.

In the common state-of-the-art, no method exists that can bring a sufficient level of guarantee on the global COTS. Nevertheless some progresses and some leads can be mentioned.

- Firstly, some IPs can be covered by systematically performed functional tests. It is the case in particular of:
  - CPU cores which failures can be considered as covered in civil aeronautics domain by DO-178 (EUROCAE & RTCA, 2010) tests performed on basic and applicative software (see for instance EASA, 2012);
  - Normalized IPs, like PCIe interfaces, are delivered with normalized test plans (Bieth, Brindejonc, 2013 in section 9.2.3.1) so that a complete test of the IP can be performed in initiator-target context.

- Secondly, it is possible to extend initiator-target tests improving the demonstration of COTS reliability regarding design errors. The method consists in generation of random scenarios built from elementary initiator-target tests chained in long sequences in which the failure modes could appear\(^5\). The coverage of these tests can be increased through a randomisation of the test scenarios. These random endurance tests explore homogenously the space of possible scenarios.

- During these tests some parameters have to be checked at least periodically or through some check sums. There are
  - The COTS outputs in order to guarantee that the COTS does not develop hidden data corruption;
  - COTS internal detection/Mitigation mechanisms status in order to guarantee that a COTS internal detection/mitigation mechanism does not hide some failures;
  - Usage domain perimeter: in particular the coherency between COTS configuration and observed features, in order to guarantee that for instance a deactivated IP cannot be reactivated although its configuration remains at deactivated.

---

\(^{5}\) Here reliability has to be taken in its prime sense: the ability to perform an intended function for a specified interval under stated conditions.

\(^{6}\) Analysis of COTS errata sheets shows that in general the failures appear after some sequence of particular instruction.
2. Detection and mitigation mechanisms

Considering the partial character of reliability demonstration improvement methods, detection and mitigation mechanisms have the following functions:

- To guarantee that the functioning of the COTS stays in a domain of controllable determinism, determined and tested through initiator–target tests.
- To guarantee a controllable functioning in the defined controllable determinism domain.

Mechanisms belonging to the first category aim at avoiding that the COTS enter in a zone where it could develop limit behaviours unacceptable from safety point of view in the context it is used. Failures leading to a modification of usage domain have been described in Figure 11. If endurance tests have shown that the IP outside of the usage domain are always coherent with their configuration, it remains to guarantee that this configuration is stable.

This test is performed in general by a CPU core. Various techniques may be implemented depending upon the level to be reached and the available resources:

- Comparison of COTS internal configuration with a reference configuration stored in an external memory;
- Addition of the internal configuration with a mirror configuration stored in an external memory;
- Comparison of a checksum computed by the CPU core on the internal configuration with a checksum stored externally.

These techniques may be difficult to implement because usage domain definition is not limited to deactivation of some IPs but also to the fine configuration of used IPs. Thus configuration checks are not limited to verification of correct inhibition of some IPs but also verification of correct configuration of used ones. Detailed analysis of configuration shows that configuration parameters corruption does not lead systematically to severe effects. It is then possible to classify parameters in several categories to which a periodicity of test is assigned depending upon:

- the maximum allowable elapsed time from the failure to the effect on COTS outputs and
- the criticality of the parameter.

This categorization is particularly useful for complex COTS on which the configuration table are huge (from 1MB up to 15-20 MB).

Mechanisms belonging to the second category aim at protecting the computation platform against failures that could occur within the usage domain (see failures of category “2” on Figure 12).

It appears, and that is a general rule also valid for mechanisms protecting against violations of the usage domain, that detection / mitigation mechanisms are of three types:

- Internal mechanisms: Mechanisms relying on internal mitigation means of the COTS;
- Mixed mechanisms: Mechanisms relying on internal COTS detection means and on external mitigation means;
- Architectural mechanisms: Mechanisms fully relying on architecture means for detection and mitigation.

Internal mechanisms are implemented by COTS manufacturers in general in order to compensate known weaknesses in their designs. For instance an Error Correcting Code (ECC) on memories in order to avoid information corruption in memory transfers. Such corruption is mainly due to bandwidth transfer increase and signal-to-noise decreasing ratio. If tests show that such mechanisms are not fully used to compensate these effects, then credits can be gained from them in the detection / mitigation of design failures. Example of internal detection / mitigations that can be encountered on an interconnect (see (Bieth, Brindejonc, 2013)) are summarised in Table 1.

<table>
<thead>
<tr>
<th>Mechanisms</th>
<th>Failure modes covered</th>
<th>IP covered by mechanisms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local access error detection</td>
<td>• Untimely or forbidden transition of address</td>
<td>Initiators</td>
</tr>
<tr>
<td></td>
<td>• Impossible transition of address</td>
<td>Interconnect (partially)</td>
</tr>
<tr>
<td>Retry Mechanism</td>
<td>• Loss of message</td>
<td>Initiators</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Interconnect (partially)</td>
</tr>
<tr>
<td>Transaction ordering mechanism</td>
<td>Abnormal sequence of message</td>
<td>Interconnect</td>
</tr>
<tr>
<td>ECC on buffers</td>
<td>• Untimely or forbidden transition of information</td>
<td>Interconnect</td>
</tr>
<tr>
<td></td>
<td>• Impossible transition of information</td>
<td></td>
</tr>
</tbody>
</table>

Table 1: examples of internal detection/mitigation mechanisms on interconnect

Mixed mechanisms are shared among internal COTS resources and architectural means. They use in general PIC (Programmable Interrupt Controller) or debug interfaces (not represented on Figure 1 and Figure 2) in order to detect failures and in general rely on an external components such as simple COTS or PLD (Programmable Logical Device) for the mitigation. More and more microcontrollers tend to integrate such features in their strategy (see for instance lockstep technology) so that in this case of mixed mechanisms tend to become internal mechanisms.

External mechanisms are of a great variety (see (Bieth, Brindejonc, 2013)):

- COTS Output monitoring where COTS output is compared to some threshold or to some information redundant pattern in order to detect dysfunctional behaviour;

9 This parameter is typically the Fault Tolerant Time Interval of ISO-26262, 2011 or Time To Alert in Aeronautics.
• End to end protection where information is encoded in a way that strongly reduce the risk of corruption and allow detection of dysfunctional behaviours;
• External independent monitoring on the principle data-path where an independent PLD checks the coherency of processor behaviour with respect to some given pattern encrypted in the PLD;
• Watchdog;
• Etc.

These mechanisms detect and mitigate failures that could directly impact safety (first mechanisms).
These first mechanisms should be tested during development by fault injection techniques and be periodically monitored in operation in order to avoid latent failures. These second mechanisms are defined to prevent first mechanisms from adverse deactivation caused by systematic or random failures. As first mechanisms, second mechanisms should be tested during development by fault injection tests.

Way forward

Methodologies presented in this publication tackle systematic design failures by relying on a black box description interpreted and consolidated at grey box level.
At each level the same failure model is used in order to list the potential failures.
The black box level is formalized through the initiator-target model that allows to enumerate the tests to be performed. These initiator-target tests have been already conducted on some modern microprocessors, leading to isolate and characterize contention domains and thus restraining the safety-critical usage of the COTS to a deterministic performance usage domain.
The grey box analysis aims at interpreting black box results and having some insights in COTS constitution and intrinsic failure modes. Both approaches enrich each other in order construct a dysfunctional model of the COTS with respect to systematic design failures. Detection and mitigation means are set to protect against these failures.
These activities are rather time consuming and the question arise of a decision criterion allowing to stop them at a necessary and sufficient level of confidence in the avoidance of systematic failures. Indeed, for random failures, the decision criterion is provided by probabilistic targets, for in-house developed component the decision criterion is provided by some development effort linked to standards (e.g. DO-254, ISO-26262, IEC-61508). However, no standardized criterion exist in order to determine if the study of a COTS design is sufficient or not for a given safety level.
Such a decision criterion could consist in the level of proof to be brought in order to ensure that a COTS usage does not present unacceptable risk in a particular industrial context.
The present paper is not intended to bring a solution to this problem, it allows however to draw some leads:
• Reliability demonstration, when it is possible to guarantee that no error can occur and
• Detection / mitigation of the errors when not possible.

These concepts have been studied in an aeronaautical context but are fully applicable to similar safety-critical industries: automotive, railway, medical industries. Some key concepts (in particular initiator-target model) can be beneficial to less safety-critical domains such as communication infrastructure and consumer electronics where COTS components are also used.

Acknowledgement
Authors thank participants to the COTS-AEH project on both EASA and Thales side for their contribution to this work and Joël Bosson (Thales Avionics Design Authority) for valuable discussions during the redaction of this article.

References
• Brindejonc, V.; Marcuccilli, G. and Petit, S., 2010 «System FMECA in the framework of ISO 26262,» in Lambda-Mu 17, La Rochelle, October 2010 (in French).
• FAA, 2008 “FAA Order 8110.105 Simple And Complex Electronic Hardware Approval Guidance, July 16th, 2008
• FIDES, 2019 UTE-C80-811 Reliability Methodology For Electronic Systems FIDES Guide 2009 Issue A