HAL
open science

Improved Test Solutions for COTS-Based Systems in Space Applications

Riccardo Cantoro, Sara Carbonara, Andrea Florida, Ernesto Sanchez, Matteo Sonza Reorda, Jan-Gerd Mess

To cite this version:

Riccardo Cantoro, Sara Carbonara, Andrea Florida, Ernesto Sanchez, Matteo Sonza Reorda, et al.. Improved Test Solutions for COTS-Based Systems in Space Applications. 26th IFIP/IEEE International Conference on Very Large Scale Integration - System on a Chip (VLSI-SoC), Oct 2018, Verona, Italy. pp.187-206, 10.1007/978-3-030-23425-6_10 . hal-02321761

HAL Id: hal-02321761

https://inria.hal.science/hal-02321761

Submitted on 21 Oct 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License

Improved test solutions for COTS-based systems in space applications

Riccardo Cantoro[1], Sara Carbonara[1], Andrea Florida[1], Ernesto Sanchez[1], Matteo Sonza Reorda[1] and Jan-Gerd Mess[2]

[1]Politecnico di Torino, Torino, Italy
{riccardo.cantoro,sara.carbonara,andrea.floridia, ernesto.sanchez, matteo.sonzareorda}@polito.it
[2]DLR, Bremen, Germany
Jan-gerd.Mess@dlr.de

Abstract. In order to widen the spectrum of available products, companies involved in space electronics are exploring the possible adoption of COTS components instead of space-qualified ones. However, the adoption of COTS devices and boards requires suitable solutions able to guarantee the same level of dependability. A mix of different solutions can be considered for this purpose. Test techniques play a major role, since they must guarantee that a high percentage of permanent faults can be detected (both at the end of the manufacturing and during the mission) while matching several constraints in terms of system accessibility and hardware complexity. In this paper we focus on the test of the electronics used within launchers, and outline an approach based on Software-based Self-test. The proposed solutions are currently being adopted within the MaMMoTH-Up project, targeting the development of an innovative COTS-based system to be used on the Ariane5 launcher. The approach aims at testing both the OR1200 processor and the different peripheral modules adopted in the system, while providing new techniques for the identification of safe faults. The results show the effectiveness and current limitations of the method, also including a comparison between functional and structural test approaches.

1 Introduction

Space applications are known to be extremely challenging from a dependability point of view, since they are supposed to work in a harsh environment (not only in terms of radiation but also from the point of view of stresses coming from extreme temperature, pression, vibration, etc.) with strong requirements in terms of reliability. In order to reduce cost and especially to increase device availability, there is a trend towards the adoption of Commercial Off-The-Shelf (COTS) components instead of the space qualified ones. Obviously, this trend requires evaluating the costs and efforts for guaranteeing that the resulting reliability still reaches the target threshold [2]. A special niche within the general domain of space applications relates to launchers. In this case, the mission time is more reduced, while the radiation environment corresponds to all the layers from ground up to the geostationary orbit (GEO). The MaM-

adfa, p. 1, 2011.
© Springer-Verlag Berlin Heidelberg 2011

MoTH-Up project [3], funded by the European Commission within the frame of the Horizon 2020 research and innovation program, aims at developing and evaluating a COTS-based system to be used in the telemetry unit of the Ariane5 (A5) launcher. More in details, the MaMMoTH-Up system is composed of several boards targeting data acquisition and processing, power management, and data transmission. All these boards use COTS components, including a flash-based FPGA where several IPs are mapped, including an OpenRISC1200 (OR1200) processor [4] whose design has been properly modified to harden it with respect to radiation effects. The adoption of such processor allows the MaMMoTH-Up system to perform significantly more powerful functions than the system it is going to substitute, e.g., in terms of data analysis and compression. In order to match the strict reliability targets of A5, the MaMMoTH-Up system must be protected not only from the radiation effects, which are mainly responsible for Latch-up and transient fault effects, but also from possible permanent faults arising during both the manufacturing process and the following system life. To target permanent faults several test steps have been identified, which are performed during and at the end of the manufacturing process, at the end of the assembly step, and after the system is mounted in the final position. Some test is also performed during the mission. The fault coverage which can be achieved by these test steps is important, since it directly impacts the achieved reliability level. To estimate the Fault Rate of the different components, we followed the FIDES guidelines [5], taking into account the stress conditions which are applied to the system before and during the mission. The Failure Rate is then derived by applying an FMECA (Failure Mode, Effects, and Criticality Analysis) procedure [11] which identifies the fault effects (and their criticality) and takes into account the timing and effectiveness (i.e., the fault coverage) of the different test steps. Remarkably, some of them have to be performed while the system is already mounted in its final position. Hence, they must basically correspond to a self-test, during which some command is sent to the system, the system performs a test of the hardware, and then results are sent outside. In the previous versions of the target system, which was based on much simpler space qualified hardware, a *functional test* was used for this purpose, where the system was asked to perform some basic operations, and a check on the computed results was sufficient to identify possible faults. Due to the much higher complexity of the MaMMoTH-Up system, this approach can hardly guarantee the achievement of the required fault coverage, especially on the OR1200 core. Hence, a *structural test* has been developed, based on a set of self-test procedures in charge of checking the possible presence of permanent faults affecting the processor core. The key difference between the two approaches lies in the fact that the functional one checks whether the system is able to deliver the expected functions, while the structural one identifies first some fault model related to the implementation of the underlying circuit, and then tries to detect the resulting faults. A major advantage of the structural approach clearly lies in the fact that the adopted fault coverage metric can be more deterministically and quantitatively evaluated than for the functional approach. Moreover, while for simple systems the functional approach (if suitably implemented) can achieve a sufficient testing quality, for more complex systems (e.g., including a CPU, some memory, and peripheral modules) the same is not true, as we will experimentally show in the paper. When

dealing with a CPU-based system, the self-test procedures implementing the structural approach follow the Software-based Self-test (SBST) paradigm [6]. Their code is integrated in the application software and, when activated, forces the processor to execute a proper sequence of instructions. The produced results are compacted into a signature which is returned to the calling program, which can thus check the possible presence of a fault by comparing it with the expected one.

The contribution of this paper lies first in describing a case of study (corresponding to a subsystem including the OR1200 core, some memories, and an I/O peripheral) where the characteristics of a functional and structural approach can be compared (not only in terms of achieved fault coverage, but also of memory footprint and duration). Secondly, it describes a scenario, where SBST can be effectively adopted, matching the several requirements of the qualification, acceptance and in-fly test of a space application. Finally, the target system is expected to perform a well-defined set of functions and in a very specific configuration (e.g., in terms of memory address space). Therefore, the FMECA is in charge of identifying which faults within the considered cores can produce any failure, and which faults will never be able to do so, e.g., because they relate to some hardware part which is not used by the application. While a few techniques have been proposed to automatically identify some categories of untestable faults, we focus here on those faults called *Safe faults*. These faults cannot produce any failure due to the specific (hardware or software) constraints the system matches during its normal operation. The paper shows that the number of safe faults is far from being negligible and uses an improved version of the method proposed in [7] to partly automate their identification, both within the CPU and the serial interface core. Due to the impact of the considered scenario, the fault coverage results reported in this paper are not directly comparable with those in [8], which focus on end-of-manufacturing test, although they refer to the same processor. A preliminary version of this paper appeared in [23], where only faults within the CPU core were considered, while in this paper we extend the analysis to an I/O peripheral core, too. The paper is organized as follows. Section 2 summarizes the main characteristics of the MaMMoTH-Up system, both in terms of underlying hardware and performed functions. Section 3 compares the functional and structural approaches, while Section 4 focuses on the identification of safe faults. Section 5 finally draws some conclusions.

2 The MAMMOTH-UP System

2.1 General architecture and functions

The MaMMoTH-Up system shall provide an experiment and data acquisition opportunity on board the Ariane5 upper stage [3]. It is designed to offer the following functionalities:

1. Acquire measurement data

2. Configure and control the experiment
3. Provide a power supply
4. Perform self-testing and fault management.

To meet these functional requirements, a COTS-based system including one experiment controller (TCM-S), two computing nodes (OBC-S), two data acquisition boards (AQB) and a power supply unit (PSU) was developed. The system is housed in a pressurized and foam-cushioned container to protect it from the harsh environment on board the launch vehicle. In order to collect sensor data and communicate with the Ariane5 upper stage, the system offers analogue acquisition channels for temperature, acceleration, vibration, shock as well as a CAN-interface for digital sensors and pressure sensors and a RS422 interface for data down-link. Synchronization with the launcher timeline and direct status reporting is done using three closed-current loops as inputs and eight discrete output pins. During the mission, the system steps through a number of different acquisition schemes according to the specific mission profile. An acquisition scheme determines which sensors are activated at which sampling rates up to 10 kHz. The data is collected and preprocessed by the computing nodes and then sent to the experiment controller using the internal SpaceWire bus. On the experiment controller, the data is analyzed, compressed and stored on a flash-based mass memory before it is sent to the Ariane5 and downlinked using the launcher's telemetry chain. The complete data flow including its allocation to the different boards is depicted in Figure 1.

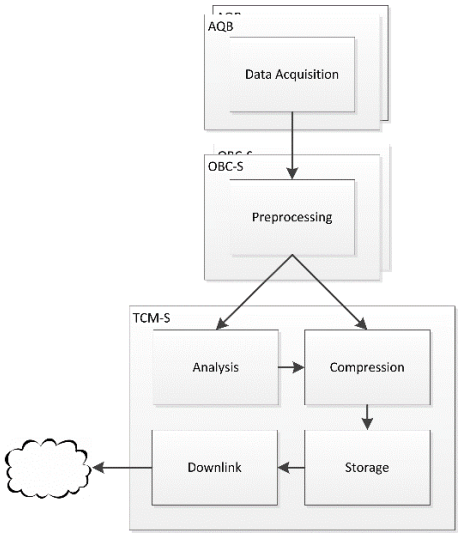

Fig. 1. Data Flow

Each OBC-S board as well as the TCM-S board include a flash-based IGLOO2 FPGA. Each of these is holding an OR1200 soft core as well as accompanying IP

cores, e.g. for SpaceWire communication amongst the boards. The required software images (three updatable images and three write-protected golden images) are kept in a two gigabyte NAND-flash memory that is implemented on each board. For data storage, the TCM-S is equipped with an additional sixteen gigabyte NAND-flash. The data acquisition is performed by a custom IP core that samples ADC channels and returns a block of samples to the software. Preprocessing, analysis and compression are then performed by software run by the OR1200 processors. The data compression algorithm consists of two steps whose computational load is divided between the OBC-S and the TCM-S. The OBC-S boards perform a wavelet transform. The transformed data is then sent to the TCM-S. From the received wavelet transform, certain characteristics of the underlying data (e.g., value range and maximum gradient) are deduced. The transformed coefficients are then encoded into an embedded bitstream. According to the deduced characteristics of a given block, a certain number of bytes in the downstream are allocated for this bitstream. All other bits are cut to save downlink budget. The complete compression scheme is described in [9]. From a reliability point of view, although the OR1200 processors on the FPGAs and especially their memories and registers are hardened by duplication or triplication of some of the underlying flip-flops, there is no redundancy at the unit or system level. If a detected failure is not permanent, the system is able to recover by performing a software reset or power-cycle on the affected board. Should this not be successful because the failure proves to be permanent, the board has to be deactivated, inevitably resulting in a loss of the connected sensor channels. In this case, the MaMMoTH-Up system follows the concept of graceful degradation: although parts of the sensors cannot be acquired anymore, the remaining transfer budget can be reallocated to use it as efficiently as possible.

2.2 The OR1200 processor

The OR1200 is the only major RTL implementation of the OR1K architecture spec. The OR1200 is a 32-bit scalar RISC with Harvard micro-architecture and 5 stage integer pipeline. The OR1200 core is mainly intended for embedded, portable and networking applications. Fig. 2 shows its architecture.

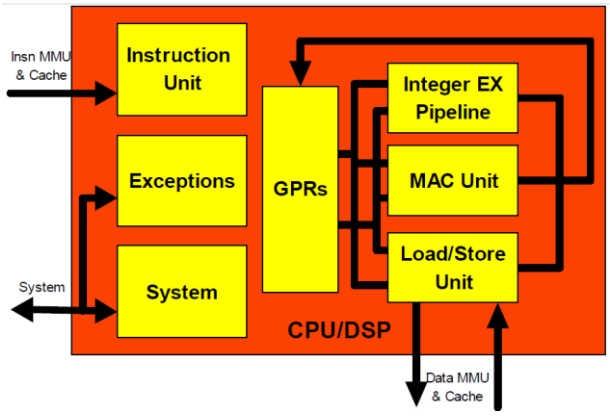

Fig. 2. OR1200 CPU Architecture

2.3 The UART core

The UART 16550 core provides serial communication capabilities, which allow communication with a modem or other external devices, using a serial cable. The peripheral core is designed to be maximally compatible with the industry standard National Semiconductors' 16550A device. It offers an 8-bit-wide Wishbone interface, FIFO only operation and a debug interface. Figure 3 depicts the most relevant modules composing it. FIFOs are not explicitly represented since they are deeply embedded in the transmitter and receiver logic.

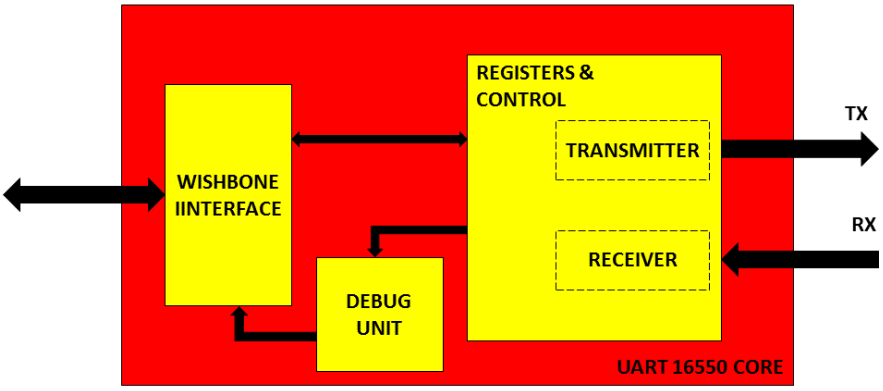

Fig. 3. UART 16550 internal architecture.

3 Comparing the functional and the structural approaches

3.1 Background

In the frame of the actions to evaluate the reliability of the MaMMoTH-Up system and to guarantee that the target figures are matched, a key role is played by the test solutions adopted to identify possible permanent faults. These solutions are activated in different phases of the product life time, since the qualification step until the operational phase (i.e., during the launch). We underline that these test solutions should be usable and effective when applied in a scenario, where the target modules (e.g., the FPGA implementing the processor) have been already mounted on their boards, and each board has been included in the final box corresponding to the telemetry unit, which has been installed in its final location within the launcher. Hence, the whole test should be performed with very limited support from the outside, and should be minimally invasive with respect to the target system. In order to evaluate the effectiveness of these test solutions and to use meaningful fault coverage figures during the reliability evaluation process, a metric must first be identified. Traditionally, a functional metric is adopted. Since the early specification phases, the list of functions that the system must support is defined. For each of them, a *functional test* is then developed, aimed at verifying that the target function is correctly performed by the system. Hence, in this scenario a qualitative metric is adopted, which guarantees that the system is not affected by any fault if all the functional tests for all the functions are successful. When moving to more complex systems including COTS components, a different metric can be considered, which first identifies a structural fault model which is supposed to represent the possible permanent faults in the target device, and then computes the percentage of structural faults which are detected by the considered test solution. One of the goals of the MaMMoTH-Up project is to define new procedures for reliability evaluation, able to match the characteristics of COTS-based systems. Given their complexity, the project partners decided to assess the effectiveness of the functional approach when a structural fault model was adopted. Since the detailed information about the structure of the adopted FPGA were missed, we decided to perform such an assessment resorting to the popular stuck-at fault model, computing first the fault coverage achieved by the functional test when the CPU circuitry mapped on the FPGA was synthesized with a generic gate-level library. This approach is partly sup-ported by the results of [20], showing that the stuck-at fault model, when applied to FPGAs, provides Fault Coverage results which are not far from those which can be obtained resorting to more accurate fault models, based on the knowledge of the internal implementation of the device (which is not available in our case). Moreover, we developed a set of SBST test procedures targeting the stuck-at faults inside the system. These procedures (that we cumulatively call *structural test* in the following) are integrated within the application software of the system and can be easily launched from the outside or by the system itself when required. Each of them returns a signature compacting the results produced by the test code, which can be compared with the expected one. A mismatch means that a permanent fault exists in the CPU core. In the following, we first report some information and figures (Table 1

and Table 2) about the functional test and the structural one (based on SBST procedures) we developed for both the OR1200 processor and the UART peripheral core. We will then report the experimental results aimed at comparing the effectiveness of the two test approaches (Table 3 and Table 4).

3.2 The functional test

The functional test for the OR1200 processor is composed of a compression algorithm that imposes a high workload on the arithmetic units of the processors. It is essential that the processor is fault-free, because even small changes in single bits of the output stream can result in a completely different set of data after decompression. Since it is impossible to predict the exact sensor readings, the processor cannot be checked using live data. Instead, precompiled blocks of sensor data together with expected values for the resulting transformation coefficients and bitstream are used. By comparing the output of the compression algorithm with the expected values, it is checked whether the calculations can be executed as planned. However, in case of an error, no diagnostic conclusions about the affected units within the processor can be drawn. The second line of Table 1 reports the size and duration of the functional test in terms of amount of memory to store the code and test time execution.

Table 1. Characteristics of the test programs for the OR1200.

	Size [Byte]	Duration [#clock cycles]
Functional test	17,360	379,815
Structural test	25,676	74,761
genpc-if	2,896	41,635
ctrl	980	980
rf	10,076	7,281
opmux	544	508
alu	3,184	10,497
multmac	2,996	9,962
lsu	4,244	3,224
wbmux	756	674

Concerning the UART core, the functional test is in charge of verifying whether the peripheral functionalities are fully operational. It is made of three main parts:

first, the peripheral is configured, selecting the appropriate operational mode and BAUD rate; then, a sequence of characters is transmitted out; finally, at the end of the transmission, it reads a sequence of characters. Such sequence is compared with the expected one, checking whether they match or not. It is important to underline that the initial configuration of the peripheral should reflect the one used in field. As for the functional test of the OR1200 processor, in Table 2 the most relevant characteristics of the functional test are shown.

Table 2. Characteristics of the test programs for the UART core.

	Size [Byte]	Duration [#clock cycles]
Functional test	212	17,760
Structural test	2,996	651,687

3.3 The structural test

The structural test for the OR1200 core is based on a suite of test procedures that target the different modules of the processor: program counter generator (*genpc*), instruction fetch (*if*), control unit (*ctrl*), register file (*rf*), operand muxes (*opmux*), arithmetic logic unit (*alu*), multiply and accumulate unit (*multmac*), load and store unit (*lsu*) and write back multiplexer (*wbmux*). Each test program executes a sequence of instructions aimed at stimulating as much as possible the target unit. At the end of the test, a signature is stored in memory: if the produced result is different than the expected one, it means that the CPU module is affected by a fault. All the test procedures have been written manually following the guidelines provided in [8]. In the following, we provide the most important characteristics of every one of the developed test programs.

The *genpc* and *if* modules are tested together using a single program. Any type of instruction from the Instruction Set must be tested. The program is written in such a way, that each type of instruction is followed by an unconditional jump to a procedure to the bottom of the code that updates the value of the signature and then it jumps back again to the top. In this way, the program counter adder inside the *genpc* is well tested, since it continuously jumps backward and forward, so performing addi-tions and subtractions. In order to test the *ctrl* module, it is necessary to give it as inputs all the possible instructions from the Instruction Set: arithmetic, logic, branch, jump, compare, multiply, load and store, immediate or register-to-register. Since the *ctrl* module also generates signals to freeze some selected stages of the pipeline or to activate the forwarding when data hazards occur, it is important to include some instruction sequences with suitable data dependency in order to stimulate those signals. The

values of the operands are not so important in this case, so random values are chosen for the operations. The *rf* module is tested using register to register operations. Basically, the test consists in writing a value into a register and then reading it. The test is divided into four parts. In the first part of the test, the stack pointer and the link registers are tested. In the second part, the first half of the registers (r2-r15) is tested, assuming the other part is not faulty and using one among these registers to hold the signature. In the third part of the test, the second half is considered in turn, assuming the first part is not faulty. The values written in the registers are 0x55555555 and 0xAAAAAAAA. To protect the CPU core against temporary faults the register file has been duplicated and the first operand is read from one register, while the second operand is read from the second register; the write back operation updates both registers. Hence, it is necessary to perform each instruction twice, swap-ping the two operands, in order to read the values from both registers. The *opmux* module selects the operands for the execution units, choosing between values coming from the register file or from the various pipeline registers when forwarding is needed. The idea to test this module is to choose arithmetic, logic, load/store and multiply instructions in such a sequence that causes data dependencies in different stages of the pipeline. The *alu* module test addresses all the possible arithmetic/logic instructions of the instruction set. Some special values generated resorting to an Automatic Test Pattern Generation (ATPG) tool launched on the combinational part are chosen as operands to better test its functionalities and all the operations are performed choosing as operands all the possible combinations between the values above. The test of the *mult_mac* module depends significantly on the values chosen for the operands. Therefore, an ATPG tool has been used again to generate proper input values. The test program consists in a series of multiplications (also with immediate, signed and unsigned operands), multiply-accumulate and multiply-subtract instructions of the computed random operands. Division instructions also involve the *mult_mac* module to operate and it has also been tested. Since the mac instruction uses special purpose registers to accumulate, it is necessary to read the values written in these registers after each multiply-accumulate instruction. For testing the *lsu* module, all kinds of load and store instructions are considered: load/store byte or word, extended to zero or signed. The program is constituted of a sequence of instructions to write and read contiguous locations in memory; each block is composed of instructions performing the following three steps: a) Storing a value in a memory location, b) Reading the value from the same location, c) Updating the signature. The values chosen to be written in memory are random and the offset to be added to the base address is a large value (from 16,380 to 17,380). The *wbmux* module chooses the value to be written back into the register file, whether it comes from the memory system (for a load instruction) or from the execution units. Since this module basically corresponds to a mux, the program is very similar to that developed for the *opmux* module. Table I summarizes the characteristics of the Functional and Structural tests for the OR1200 processor in terms of size and duration. For the Structural test, we detailed these figures for each test procedure.

The structural test of the UART consists of a unique test program, which was developed following the guidelines presented in [22]. Differently than in the aforemen-

tioned work, the UART was exclusively configured in one operational mode, namely the very same that the application software is using within the MaMMoTH-Up system. In fact, there is no reason to test the peripheral in all the possible operational modes: the only operational mode that matters is the one used when the device is in mission mode. After a first initialization phase, a loopback connection is activated. This feature is normally available in most serial communication protocols, and it is essential for devising any in-field testing strategy. The structural test can be split in different phases as well, each of them targeting the test of a specific module.

First the transmitter and receiver FIFOs are tested. FIFOs can be tested resorting to any test algorithm developed for a register file, as the one proposed in [21]. The two FIFOs can be considered as a unique register file, since whenever an entry is written in the transmitter FIFO, the same entry is written in the receiver FIFO as well. The significant difference with respect to a register file lies in the fact that the entries are not randomly addressable, since the access to the FIFO is performed through specific pointers. As specified in [21], first the entries are grouped into two groups (group 0 and 1), depending on the hamming distance of their encodings. Then, the patterns 0x55 and 0xAA are applied (that is, written and then read) to the FIFO. Patterns are written to the transmitter FIFO by initiating a transmission, and read out when the transmission ends. On the transmission, the same patterns are also applied to the receiver FIFO. It is worth noting that since the entries are not randomly addressable, the sequence of write operations to the FIFO should be fixed and carefully constructed, in order to write the intended pattern to the specific entry. Specifically, the algorithm requires that the value 0x55 is written to group 0 and 0xAA to group 1.Then, the test is repeated, but with the patterns inverted.

The second part of the test focuses on the circuitry embedded in the receiver, in charge of detecting possible transmission errors. For some error types (e.g. break indicator error, overrun error), it is required to simply force the error during the transmission and then check whether it was acknowledged by reading out the peripheral register that keeps track of such events. For example, the overrun error is tested by simply transmitting a number of characters higher than the number of entries in the receiving FIFO. For other types of error (namely, parity and frame error), they are caused by physical faults (e.g., noise) in the communication channel. Hence, it is required to deactivate the loopback connection for emulating these errors. A transmission of a certain number of characters is first initiated. While the transmission is still ongoing, the loopback connection is disabled. Depending on how long the connection is interrupted it is possible to emulate both parity and frame (i.e., missing STOP bit in the received frame) errors, and consequently excite the logic that checks incoming data for these types of errors. Short periods of interruption cause parity errors, while longer one can trigger the frame ones. The characteristics of the functional and structural test procedures of the UART are reported in Table 2. It is noteworthy that the structural test requires a considerable higher number of clock cycles than the functional one due to the higher complexity of the latter. Specifically, they originate from the fact that the structural test fully tests the FIFOs and the error detection logic,

which require several transactions to be completed and (for the error detection test) interruption of the transmission. Moreover, it is important to underline that the peripheral transmits data at lower frequency (i.e., the BAUD rate) compared to the one used by the CPU.

3.4 Results

For the purpose of our experiments, we created a simulation setup where both the OR1200 processor and the UART cores lie in a system composed also of a 64 MB RAM, as in the MaMMoTH-Up OBC-S boards. They were synthesized and mapped to the CMOS NanGate 45nm Open Cell Library. The obtained netlists are used to perform the fault simulation experiments with a commercial EDA tool. Using this setup, we evaluated the stuck-at Fault Coverage obtained by both the functional and the structural test described above.

Results are reported in Table 3 for the CPU core, detailing also the results achieved on each component module. As the reader can notice, the Fault Coverage achieved by the Functional test on the CPU core is far lower than the one of the Structural test. This supports the claim that a Functional test cannot be effectively used when complex COTS-based systems are used. It is also worth underlining that the comparison between the two tests provides very different results depending on the considered module. For some of them (e.g., genpc) the fault coverage achieved by the Functional test is slightly higher than by the Structural test. This is basically due to the fact that some modules can be tested in a good way by executing long programs, and the Functional test is much longer than the Structural one. However, for modules that include large combinational parts (e.g., alu and mult-mac) or require a specific sequence of operations to be tested (e.g., rf) the Structural approach is far more effective.

Table 3. Stuck-at (SA) fault coverage of functional and structural test on OR1200.

Module	Total SA faults	Functional Test	Structural Test
Whole CPU	124,612	32.09 %	81.89 %
genpc	4,906	60.80 %	57.97 %
if	2,268	50.57 %	71.12 %
ctrl	4,320	71.53 %	80.25 %
rf	39,056	33.97 %	90.93 %
opmux	2,530	90.51 %	96.05 %
alu	14,532	46.04 %	78.50 %
mult_mac	39,398	13.91 %	95.77 %

sprs	5,522	8.31 %	37.61 %
lsu	2,708	67.61 %	65.99 %
wbmux	2,286	69.29 %	78.83 %
freeze	126	75.40 %	76.98 %
except	6,716	15.86 %	18.92 %
cfgr	232	0.00 %	0.00 %

Table 4 reports the results of the fault simulation campaigns concerning the UART. By comparing columns three and four of such table, it can be noticed that the fault coverage follows the same trend as for the OR1200, i.e., the functional test achieves a significantly lower fault coverage than the structural one. In particular, the structural test clearly outperforms the functional test for all the modules composing the UART. Indeed, for most of the units (e.g., the circuitry of receiver and transmitter) composing the peripherals the structural test at least doubles the fault coverage of the functional one.

This is a further evidence that strengthens our claim for the necessity of a more complex and detailed structural test when dealing with these systems. Specifically to this case study, it is evidently not enough verifying whether the peripheral is actually able to deliver the intended functionalities. Considering for example the transmitter and receiver FIFOs (*Tx_fifo* and *Rx_fifo*, respectively), if one would rely on the result of the functional test solely, 80% of the faults for each FIFO would be missed.

Table 4. Stuck-at (SA) fault coverage of functional and structural test on the UART core.

Module	Total SA faults	Functional Test	Structural Test
Whole UART	17,876	36.38%	68.00%
wb_interface	1,400	53.36%	63.36%
regs	15,962	35.38%	69.83%
transmitter	3,956	41.86%	94.08%
Tx_fifo	2,196	20.40%	99.73%
receiver	7,794	32.13%	70.45%
Rx_fifo	2,196	20.17%	99.36%
dbg	338	0%	0%

4 Safe Faults

We denote as *Safe Faults* those faults that can never produce a failure in the considered system[1]. One of the goals of the FMECA process is their identification, since they do not contribute to the Failure Rate, and should thus be removed from the Fault List to be considered when evaluating the Fault Coverage achieved by the test procedures.

Moving from the space domain to the automotive one, we could mention that the ISO 26262 standard for automotive applications also considers Safe Faults, and defines them as "application dependent". Clearly, Safe Faults include untestable faults, i.e., faults for which no test exists. Hence, it can be useful to review the different categories included in the set of Safe Faults for a given system:

- *Structurally (or combinationally) untestable faults,* i.e., faults for which a test does not exist even if the combinational block where the fault is located is fully controllable and observable (e.g., via scan test). Examples of faults belonging to this category include faults that cannot be tested due to some redundancy in the combinational logic. Structurally untestable faults identification is a by-product of the combinational ATPG process, for which very effective algorithms are known [24]. Hence, if a gate-level description of the block is available, an ATPG tool can identify these faults.
- *Sequentially untestable faults*, i.e., faults that do not belong to the previous group, but cannot be tested due to the sequential behavior of the circuit, for example, because the circuit cannot reach any of the states required for their test. Several works proposed techniques to automatically identify these faults, either in a generic circuit [12][13][14][17][18] or specifically in a CPU [15]. Sequentially untestable faults identification is a sub-problem of sequential ATPG, which is known not to be a practically solvable problem for generic real circuits. Design for Testability techniques (e.g., scan) have been introduced to circumvent this problem reconfiguring the circuit under test at test time.
- *On-line functionally untestable faults,* [10], i.e., faults that do not belong to the previous groups, but cannot be tested in a functional manner (i.e., without resorting to Design for Testability) in the operational conditions the target device works in. On-line functionally untestable faults can be related for example to the specific memory configuration adopted by the system [16]. Several bits in the processor Program Counter or in the registers storing the addresses in the Load-Store Units become untestable if the memory area storing the code and the data is less than the maximum one.

Safe faults include and extend the previous categories. In the following we report some examples of safe faults:

---

[1] When performing FMECA, it is common to also categorize faults depending on the criticality of the resulting failures, i.e., on how serious their effects are. Reliability figures typically depend only on critical safe faults. For the purpose of this paper we ignore any distinction within the set of safe faults.

- The debug circuitry possibly existing in a processor generates safe faults, since debug facilities are not used during the normal behavior, and most faults within it cannot impact the system behavior and produce any failure.
- Several faults in the Design for Testability hardware (e.g., the scan chains) used for end-of-manufacturing test also correspond to safe faults: for example, faults on the scan-in input of the scan flip-flops are safe faults.

In [7], we reported some results concerning the identification of safe faults in the openMSP430 processor. In that paper, we also considered those safe faults that cannot produce any failure due to the specific application code executed by the CPU. As a simple example, if the system application only uses integer arithmetic, faults in the Floating-Point Unit become untestable.

4.1 Safe Faults identification

The typical approach for safe faults identification is based on manual analysis. In many project teams, the designers, test engineers, and reliability/functional safety experts systematically cooperate to categorize faults and (based on their effects) identify safe faults. Clearly, this process is extremely time consuming (and hence expensive), as well as prone to possible errors. For this reason, in [7] we proposed an approach, which aims at partly automating the safe faults identification process taking into account all the constraints coming from the application scenario, including the application software to be run by the CPU. Some preliminary results coming from the application of the same method to the OR1200 CPU have been reported in [19]. In this paper, we improve the procedure used in [7] and [19] and extend the work reported in [23] to peripheral modules, not necessarily CPU only. The improved method is now able to identify a larger number of safe faults, thanks to a mechanism allowing to exploit the power of a commercial ATPG tool. In the following, the proposed method for safe faults identification is detailed. For sake of generality, it is presented assuming a generic sequential circuit as Device Under Analysis (hereinafter, DUA). However, as it is explained further in this section, the applicability of the proposed method spans DUAs of different kinds, namely from CPUs (as in the above-mentioned works) to peripheral cores (as described in this paper). Our method for safe faults identification operates on the gate-level netlist of the DUA, and it is based on the following steps:

1. We identify the set of all inputs of the DUA which will remain at a fixed value during the system operation (e.g., the Normal/Test signal). Let us call PI_{fixed} this set.
2. We perform several simulation experiments on the DUA running the actual workload that the circuit will experience in the operating conditions and with different but realistic data input sequences. Then, we use the toggle activity to identify the set $FF_{possibly\text{-}fixed}$ of flip-flops which never toggle.
3. We focus on $FF_{possibly\text{-}fixed}$, and manually check whether any of the flip-flops in this set may possibly toggle if a different sequence of input data and events is considered. The remaining set of flip-flops, called FF_{fixed} is composed of those flip-flops that will never toggle in the operating conditions.

4. We resort to an ATPG tool to identify the faults in the combinational logic of the DUA that become untestable once the constraints coming from the fixed values of the PI_{fixed} and FF_{fixed} signals are applied. In other words, the combinational logic is extracted from the whole gate-level netlist and then we specify the inputs of such portion of circuit whose values always remain fixed during the operational phase (i.e., connecting the inputs either to ground or to V_{DD}). Finally, given these conditions on the inputs, the ATPG tool is asked to generate test patterns for all the possible faults within the combinational logic. At the end of the generation process, the ATPG tools identifies a certain number of untestable faults due to the constraints on the combinational logic inputs. These faults correspond to safe faults for the system.

The method is also applicable to combinational circuits. In this case, the procedure described above can be simplified since it is not necessary to consider the values of flip-flops. Moreover, the reader should note that in [7] and [19] the last step was performed resorting to a simple topological analysis of the effects of the fixed values in the combinational logic: the analysis identified for each gate the possible safe faults caused by any fixed value on the inputs of the fault. To perform the same step, in this paper we resort to an ATPG tool, so that a larger number of safe faults can be identified, taking into account the constraints on the input signals of the combinational logic. The reasoning behind the usage of an ATPG tool originates from the test pattern generation process itself. As an example, let us consider the portion of combinational logic shown in Fig. 4, where I_0 to I_4 represent the inputs of the logic cone for the output Out. Let us assume that as a result of the analysis in steps 1 to 3, we realize that I_4 always remains to a fixed value (i.e., zero, therefore connected to ground). We can now force the ATPG tool to derive a test pattern for the fault stuck-at-0 on the output of gate U3. For testing the stuck-at-0 on that signal, the ATPG tool has to force U3 inputs both to 1 and then propagate up to the Out signal. The only input pattern that satisfies these requirements would be 01111. However, I_4 is connected to ground. Therefore, the value of U3 is always at 0 *independently from the other inputs* of the logic cone. If the ATPG tool is not able to derive a test pattern given the constraints on the inputs, it means that the fault can be marked as untestable. Since the constraints on the inputs derive from a workload that mimics the operating condition of the DUA, the occurrence of that fault will never cause a failure and can be labeled as safe.

Leveraging an ATPG tool allowed us to increase by about 3% the number of safe faults identified by this step with respect to the results presented in [19]. Moreover, the usage of a commercial tool also makes the applicability of the proposed method easier. It is important to underline *that our method cannot identify all safe faults in the system*. However, we claim that it can identify a significant number of them and represents a first step towards the automation of the whole safe fault identification procedure, thus contributing *to significantly reducing its cost*.

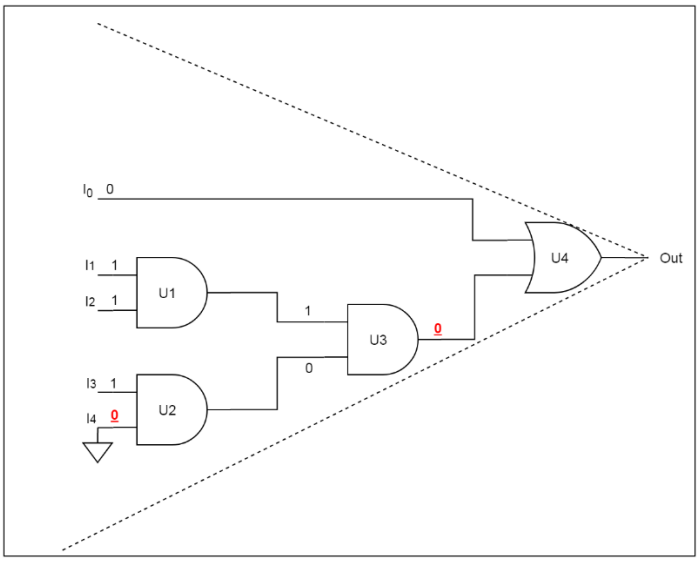

Fig. 4. Safe Fault Identification process with an ATPG tool.

4.2 Results

We implemented a tool based on a set of TCL scripts interacting with a logic simulator and an ATPG to implement the procedure described in the previous sub-section. The required time to run the simulation campaign to gather the data for the Toggle analysis and to process them to extract the list of Safe Faults (including the ATPG step) is in the order of a few hours. In the following we will present and discuss the experimental results for the same setup used in Subsection 3.4, including the OR1200 CPU and the UART.

By using the same commercial ATPG tool we also identified the number of structurally untestable faults in the OR1200, which amounts to 80. Following the proposed procedure and referring to the environment and application code of the OBC-S board, we identified a set of safe faults in the OR1200 processor, as reported in the second column of Table 5. We also computed the Safe Fault Coverage (*SFC*) for the Functional and Structural tests (columns 4 and 5), defined as:

$$SFC = \frac{\#detected\ faults}{\#faults - \#safe\ faults}$$

The reported results show that:

- The number of safe faults is relevant, accounting for about 13% of the whole stuck-at fault list.

- The percentage of safe faults varies widely from one module to another. It is about 20% for modules such as *mult_mac* and *sprs* (dealing with special purpose registers, which are not significantly used by the application). It is also significant for modules such as *if*, *genpc* and *rf*, which are not fully used by the application code.
- The SFC figure achieved by the structural test procedures is quite high (taking also into account the observability constraints of the test environment) and allows (combined with some further test techniques implemented at a higher level) to fully match the reliability requirements for the MaMMoTH-Up system.

Table 5. Safe stuck-at fault coverage (SFC) of functional and structural tests for the OR1200.

Module	Safe faults	Safe faults w.r.t. total faults	SFC	
			Functional Test	Structural Test
CPU	16,183	12.98 %	36.88 %	84.41 %
genpc	425	8.66 %	66.57 %	63.24 %
if	204	9.00 %	55.57 %	76.74 %
ctrl	13	0.30 %	71.74 %	80.42 %
rf	5,550	14.21 %	39.60 %	92.89 %
opmux	41	1.62 %	92.00 %	96.47 %
alu	75	0.51 %	46.28 %	78.54 %
mult_mac	7,861	19.95 %	17.38 %	97.04 %
sprs	1,070	19.38 %	10.31 %	44.41 %
lsu	7	0.26 %	67.79 %	66.16 %
wbmux	0	0.00 %	69.29 %	78.83 %
freeze	7	5.55 %	79.83 %	79.51 %
except	912	13.58 %	18.35 %	21.85 %
cfgr	18	7.76 %	0.00 %	0.0

Concerning the UART core, we followed the same procedure and the results are gathered in Table 6. Preliminarily, as for the OR1200 analysis, the ATPG tool identified 13 structurally untestable faults. Compared to the previous case study, when dealing with a peripheral it can be observed that the percentage of safe faults is considerably higher (29% against 13%). This mainly stems from the fact that the functionalities offered by a peripheral often depend on a set of configuration registers whose value is written in the initialization phase. Thus, because of these registers, non-negligible

portions of the design can be actually unused. As a consequence, most of these faults belongs to the *regs* unit, which embeds such configuration registers and the control part. Specifically to the considered UART:

- The *receiver* is the module which is directly affected by the configuration registers. Among the other duties, this unit is in charge of generating interrupts as response to several events. In our application, the UART was used in polling solely. As a consequence, the interrupt facilities are not used at all. The same reasoning applies also to the Modem configuration. The peripheral can be connected to an external model through a dedicated interface, which is not needed by the application. Finally, depending on the format of the transmission format, some portions of the circuit become inactive as well. All these factors affect partially the FIFO as well, for which some safe faults were identified.
- Safe faults can be found also in the *transmitter*, although their number is significantly smaller compared to the receiver. These faults mainly originates from: 1) modem not being used and 2) the configuration of the transmission format (e.g., the selected BAUD rate).
- The debug unit (*dbg*) is not used at all, since this unit is mainly intended for checking the correctness of the operation during the development phase and clearly not for the in-field operations.
- The remaining safe faults are part of the control circuitry, included in the *regs* module.

By comparing the fourth column of Table 6 with the fourth of Table 4, it is possible to observe that removing the safe faults significantly increases the fault coverage achieved by the structural test, reaching quite high SFC figures. Furthermore, removing safe faults considerably reduces the effort for the development of the self-test procedures. Indeed, the test engineer should not target all the possible faults affecting the UART, but instead exclusively the ones considered critical for the application (i.e., not safe) However, it is worth noting that 16% of the remaining undetected faults might still include safe faults. From the experiments, it emerged that most of them could be linked to the interrupt circuitry. However, they cannot be clearly identified with the current methodology, thus we decided not to include them in the set of safe faults.

Table 6. Safe stuck-at fault coverage (SFC) of functional and structural tests for the UART 16550.

Module	Safe faults	Safe faults w.r.t. Total SA faults	SFC	
			Functional Test	Structural Test
UART	5,192	29.04%	49.96%	84.02%
wb_interface	340	24.29%	69.62%	74.81%

regs	4,484	28.09%	47.85%	84.99%
transmitter	217	5.49%	43.89%	97.17%
Tx_fifo	0	0.00%	20.40%	99.68%
receiver	2,213	28.39%	42.36%	81.04%
Rx_fifo	288	13.11%	20.89%	99.42%
dbg	338	100.00%	100%	100%

5 Conclusions

This paper deals with the adoption of COTS components in the design and manufacturing of electronic systems to be used on a launcher. We focused on the MaMMoTH-Up system to be used on board the Ariane5 launcher, which represents a testbench for developing a suitable design and manufacturing flow compatible with the adoption of COTS components. In particular, we focused on the test of a couple of modules within the whole system (i.e., the CPU core and a serial peripheral core), showing first that a functional test is not able to achieve a sufficient test quality, while structural SBST test procedures can be much more effective. We also focused on the identification of safe faults, i.e., those faults that cannot produce any failure due the hardware and software constraints provided by the application environment. We proposed a semi-automated method able to significantly reduce the cost and effort for safe faults identification, showing that the method can identify a significant number of safe faults. We reported experimental results on the OR1200 processor core used within the MaMMoTH-Up system. The same experiments were also performed on a peripheral for serial communication embedded in the system (namely the UART 16550). Although the proposed method has been experimentally evaluated referring to stuck-at faults, only, the same approach can be adopted to deal with other fault models (e.g., transition delay faults, or bridges), if required. We are currently working towards the development of improved techniques for safe faults identification and towards a new and more effective release of our SBST procedures.

Acknowledgments. This work has been supported by the European Commission through the Horizon 2020 Project No. 637616 (MaMMoTH-Up).

References

1. S. Avramenko; M. Sonza Reorda; M. Violante; G. Fey; J. -G. Mess; R. Schmidt, “On the robustness of DCT-based compression algorithms for space applica-

tions", 2016 IEEE 22nd International Symposium on On-Line Testing and Robust System Design (IOLTS)
2. M. Pignol, "COTS-based applications in space avionics", 2010 Design, Automation & Test in Europe Conference & Exhibition (DATE 2010)
3. http://www.mammoth-up.eu/
4. https://openrisc.io/
5. UTE FIDES guide 2009, Edition A, September 2010
6. M. Psarakis et al., "Microprocessor Software-Based Self-Testing", IEEE Design & Test of Computers, vol. 27, no. 3. May-June 2010, pp. 4-19
7. R. Cantoro, A. Firrincieli, D. Piumatti, E. Sanchez, M. Sonza Reorda, M. Restifo, "About functionally untestable fault identification in microprocessor cores for safety-critical applications", IEEE Latin-American Test Symposium (LATS), 2018
8. N. Kranitis; A. Merentitis; G. Theodorou; A. Paschalis; D. Gizopoulos, "Hybrid-SBST Methodology for Efficient Testing of Processor Cores", IEEE Design & Test of Computers, 2008, Volume: 25, Issue: 1, pp. 64 – 75
9. J.-G. Mess, R. Schmidt, G. Fey, "Adaptive Compression Schemes for Housekeeping Data", 2017 IEEE Aerospace Conference
10. P. Bernardi, M. Bonazza, E. Sanchez, M. Sonza Reorda, and O. Ballan, "On-line functionally untestable fault identification in embedded processor cores", Proc. Design, Autom. Test Eur. Conf. Exhibit. (DATE), Mar. 2013, pp. 1462–1467
11. W. M. Globe, "Control Systems Safety Evaluation and Reliability", third edition, ISA, ISBN 978-1-934394-80-9
12. J. Raik, H. Fujiwara, R. Ubar, A. Krivenko, "Untestable Fault Identification in Sequential Circuits Using Model-Checking", Proc. IEEE Asian Test Symposium, 2008, pp. 21-26
13. Syal, M.; Hsiao, M.S., "New techniques for untestable fault identification in sequential circuits", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 5, no. 6, 2006, pp. 1117 – 1131
14. H.-C. Liang; C. L. Lee; Chen, J.E., "Identifying Untestable Faults in Sequential Circuits", IEEE Design & Test of Computers, Vol. 12 , No. 3, 1995, pp. 14-23
15. W.-C. Lai; Krstic, A.; Kwang-Ting Cheng, "Functionally testable path delay faults on a microprocessor", IEEE Design & Test of Computers, vol. 17, no. 4, 2000, pp. 6-14
16. A. Riefert; R. Cantoro; M. Sauer; M. Sonza Reorda; B. Becker, "A Flexible Framework for the Automatic Generation of SBST Programs", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2016, Volume: 24, Issue: 10, pp. 3055 – 3066
17. David E. Long, Mahesh A. Iyer, Miron Abramovici, "FILL and FUNI: algorithms to identify illegal states and sequentially untestable faults", ACM Transactions on Design Automation of Electronic Systems (TODAES), v. 5, n. 3, pp. 631-657, July 2000
18. Daniel Tille, Rolf Drechsler, "A fast untestability proof for SAT-based ATPG", 12th International Symposium on Design and Diagnostics of Electronic Circuits&Systems, pp. 38-43, April 15-17, 2009

19. S. Carbonara, A. Firrincieli, M. Sonza Reorda, J.-G. Mess, “On the test of a COTS-based system for space applications”, 24th IEEE International Symposium on On-Line Testing and Robust System Design, 2018, poster session
20. Jaroslav Borecky; Martin Kohlik; Pavel Kubalik; Hana Kubatova, “Fault Models Usability Study for On-line Tested FPGA”, 14th Euromicro Conference on Digital System Design, 2011, pp. 287-290
21. D. Sabena, M. Sonza Reorda and L. Sterpone, "A new SBST algorithm for testing the register file of VLIW processors", 2012 Design, Automation & Test in Europe Conference & Exhibition (DATE), Dresden, 2012, pp. 412-417
22. A. Apostolakis, M. Psarakis, D. Gizopoulos and A. Paschalis, "Functional Processor-Based Testing of Communication Peripherals in Systems-on-Chip", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 15, no. 8, pp. 971-975, Aug. 2007
23. R. Cantoro, S. Carbonara, A. Floridia, E. Sanchez, M. S. Reorda and J. Mess, "An analysis of test solutions for COTS-based systems in space applications," 2018 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC), Verona, Italy, 2018, pp. 59-64.
24. M. Bushnell, V. Agrawal, Essentials of Electronic Testing for Digital, Memory, and Mixed-Signal VLSI Circuits, Kluwer Academic Publisher, 2000