Experiences during the Experimental Validation of the Time-Triggered Architecture*

S. Blanc, J. Gracia, P.J. Gil
Fault Tolerant Systems Group (GSTF) - Department of Computer Engineering (DISCA)
Polytechnic University of Valencia, Spain
{sablacla, jgracia, pgil}@disca.upv.es

Abstract

During last years, the Time-Triggered Architecture (TTA) has been gaining acceptance as a generic architecture for highly dependable real-time systems. It is now being used to implement the "x-by-wire" concept. A problem for this kind of systems is their validation. Fault Injection has achieved a great acceptance among designers for the experimental validation of systems. This work describes the results and experiences obtained during the validation of the TTP/C controller, a communication controller based on the TTA. Two different fault-injection techniques have been used: VHDL-based fault injection and physical fault injection at pin level. Due to the access that each technique has to different parts of the system, they can complement each other, but moreover, some experiments can be reproduced using both techniques, being very helpful for the analysis of the results.

1. Introduction

Fault injection is used to validate Fault-Tolerant Systems (FTS) and it is being increasingly consolidated and applied in a wide range of fields. Fault injection techniques in the hardware of a system can be classified in three main categories: *Hardware Implemented Fault Injection (HWIFI)*, *Software Implemented Fault Injection (SWIFI)*, and *Simulated Fault Injection* [1, 2].

However not all methods and techniques are giving the same results and coverages. It has been proved very useful to combine different fault injection techniques when validating a FTS [3, 4].

Although the industrial use of fault injection techniques is not very spread, several attempts have been done. One of these attempts is the FIT project [5]. During this project, four fault injection techniques have been used to validate the TTP/C controller, a communication controller based on the TTA. The use of this architecture is growing in areas like avionics or the automotive industry, where the x-by-wire systems are a new challenge. In the case of automotive industry, some electrical and electronic parts are replacing mechanical and hydraulic parts. These electrical and electronic elements must be fault-tolerant and accomplish hard real-time specifications.

This paper is focused in the fault injection experiments done using VHDL-based fault injection and physical fault injection at pin level.

Both techniques have been used to validate two prototypes of the Time-Triggered communication controller, the TTP^{TM}/C-C1 and TTP^{TM}/C-C2. Two fault injection tools developed in the Polytechnic University of Valencia have been used: AFIT (*Advanced physical Fault Injection Tool*), a physical fault injector at pin level [6] and VFIT, a VHDL-based fault injection tool [7].

The distribution of this paper is as follows. Sections 2 and 3 describe the tools used during the fault injection experiments. In section 4 we present the TTP/C controller and we explain the different experiments performed. Section 5 presents the results obtained and section 6 gives some general conclusions and future work.

2. VFIT: VHDL-based Fault Injection Tool

VFIT [7], the tool used during the VHDL-based fault injection experiments, consists of a series of elements designed around a commercial VHDL simulator (ModelSim, the new simulator of Model Technology [8]).

To carry out an experiment, different faults are injected into the model. For every injected fault, the behaviour of the model is analysed. An injection campaign is a set of different experiments, and involves three independent phases:

- **Experiment Set-up**. Here, both the parameters used to describe the fault and the analysis conditions are specified.
- **Simulation**. In this phase, two operations are carried out. Firstly, a set of macros is automatically generated: one macro performs a fault-free simulation of the model while the others have the commands needed to inject the specified faults. Secondly, the VHDL simulator executes the macros, obtaining a set of simulation traces: a fault-free simulation, and *n* with a fault injection performed, being *n* the number of faults injected.
- **Analysis and Readouts**. The *n* faulty simulation traces are compared to the fault-free trace, studying their differences.

* This work has been partially funded by the Project IST-10748- FIT.

About fault timing, transient, permanent and intermittent faults can be injected. It is possible to choose among different probability distributions to determine both the injection instant and the fault duration.

With respect to the fault models used, they depend on the injection technique and the abstraction level of the system model. Table 1 shows the fault models that can be injected with each technique.

Table 1. Fault models of VFIT [7].

	Fault models	
Injection technique	**Transient Faults**	**Permanent Faults**
Simulator Commands	Stuck-at (0,1), Delay, Bit-flip, Pulse, Indetermination	Stuck-at (0,1), Delay, High impedance, Indetermination
Saboteurs	Same as for Simulator Commands	Same as for Simulator Commands, plus Bridging, Stuck-open
Mutants	Syntactic changes in the VHDL model	Syntactic changes in the VHDL model

3. AFIT: Fault injection in the prototype

Emulation of environmental parameters such as temperature, undesirable capacitance, the precision error of passive components, etc., are better provoked in the real prototype. To validate a new prototype is an effort that can be reduced by a previous pre-implementation testing. But in spite of a good validation of the design by using simulation techniques, the validation of the physical prototype with all their components (hardware and software: OS, application, firmware, etc.) is also important.

Among techniques that can inject faults in the real prototype, physical fault injection at pin level presents some advantages. The technique is based on the perturbation of the logic values of the integrated circuit (IC) pins. It can be used to modify the incoming/upcoming data of the IC. But the main difference with other HWIFI or SWIFI techniques is that signals used by an IC to communicate with other system elements can be perturbed with high controllability [6].

The last version of AFIT [4, 9] was designed to inject faults in distributed embedded systems, where several electronic modules (nodes) share the same communication channel. This external tool does not halt or delay the target execution. Otherwise, it would not be possible to test the system, as the rest of nodes are running. AFIT is able to inject intermittent, permanent and transient single or multiple faults (up to 4), being the fault model *stuck-at (0, 1) line*.

AFIT is a modular tool with a PC handling the interface between user and injector. A specific monitor for distributed systems has been developed to observe the system behaviour in presence of faults. It consists of a hardware monitor combined with software tasks implemented inside each node. These tasks are used to report if any specific event is detected.

Both injection tool and monitor are independent and external to the target. They can be managed by the same personal computer handling the interface between user and devices (see Figure 1).

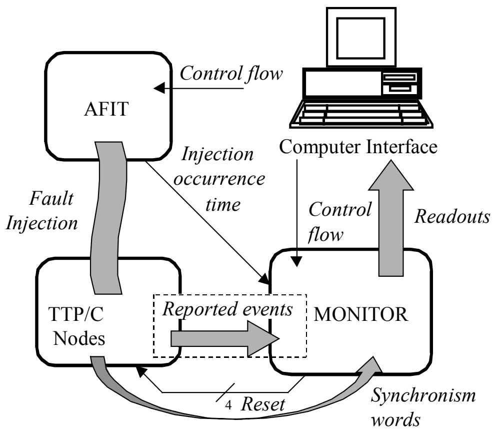

Figure 1. Interconnection AFIT-target system.

Experiments are automated and due to their short duration (less than 1 second), it is possible to obtain a reliable sample, which allows a high number of experiments to be carried out without supervision.

The monitor also provides a physical reset used to restart the process in a normal state. Eventually, the monitor provides an input port per node waiting to receive the *synchronism words*. A node sends a synchronism word when it is ready or it has reached a specified state. This mechanism is very useful if the target can reach several execution states.

4. Fault injection experiments

4.1. The system under test: the TTP/C controller

The TTA [10] implements a real-time communication protocol for the interconnection of electronic modules in a distributed fault-tolerant real-time system. The Time-Triggered Protocol (TTP) is synchronous, and has a static and cyclic scheduling. The main characteristic is the fail-silent behaviour, which assures that the node will work correctly or will stop the communication.

Figure 2 shows the block diagram of the TTP^{TM}/C-C1[1] controller. The communication between the controller and the host computer is carried out through the Communications Network Interface (CNI). The CNI is structured in two main areas, a status/control area and a message area. The host uses the CNI to monitor the controller, while the controller uses the CNI as a buffer for incoming/upcoming messages, as well as for storing control information. The CNI is implemented as a dual ported RAM in the Host Interface.

The VHDL model of the TTP^{TM}/C-C1 controller is organised around the Protocol Control Unit (PCU). Some low-level functional blocks that implement performance critical and hardware related protocol features complete the model. The PCU controls the interaction of

[1] C1 is the 1st controller of the TTP^{TM}/C microcontroller family.

these low-level blocks and executes high-level protocol mechanisms.

The Message Descriptor List (MEDL) defines the specific point in time for transmitting a frame, according to the predefined communication schedule. The MEDL is defined before the system starts operating, and it is not possible to change it during run-time.

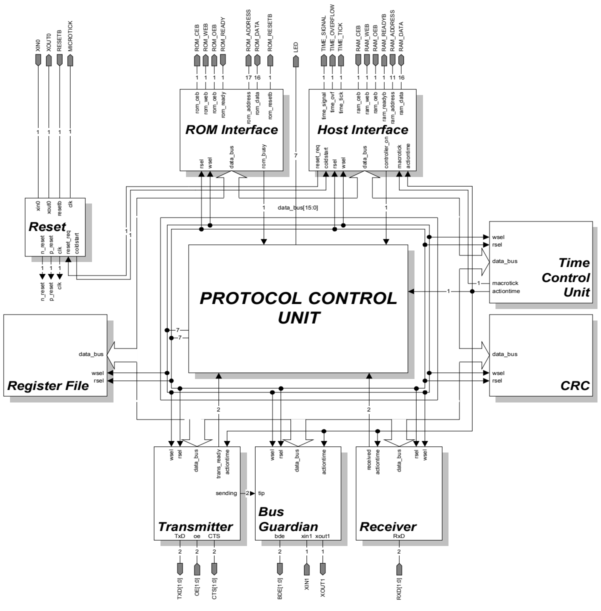

Figure 2. Block diagram of the TTP^{TM}/C-C1 controller [5].

The MEDL resides in an external flash EPROM, which can be downloaded, accessed and programmed by the controller. To connect the low-level blocks, a synchronous internal register bus mastered by the PCU is used. A single register address space is used with the register file and the low-level blocks. Register transfers between units are issued by *move* instructions of the PCU.

The main change in the TTP^{TM}/C-C2 is the addition of a new block, called Bus Interface FIFO. This block is shared between the receiver and the transmitter (these blocks are never active at the same time). It is used as a buffer for incoming/upcoming frames, allowing different types of operations in the system [11].

The TTP/C uses a TDMA scheme for the exchange of messages among the nodes. In a TDMA round, one time slot is assigned to each node in a cluster (the set of nodes sharing a bus in a TTP/C system). An a-priori defined time schedule controls all activities of the cluster. A distributed algorithm establishes the global time base with steady clock synchronisation.

Respect to the prototype, the main elements of a node are the local host controller (Motorola MC68360 in the TTP/C-C1 prototype and Motorola PowerPC PPC555 in the TTP/C-C2 prototype), the TTP/C controller and the Communication Network Interface (CNI), that works as a dual ported RAM between host and TTP/C controller. The host workload consists of the real-time Operating System (TTPOS), the Fault–Tolerant communications layer (FTcom), and the application. The network uses a bus topology with two communication channels called TTP/C bus.

4.2. Fault injection parameters

The hypothesis behind the TTP/C communication protocol is that a single fault will not disturb the operation of the complete distributed real-time system and the fail-silent assumption: the node will work correctly or will stop communication. That means if any given node fails, it will not disturb the others. If there is a replica of the erroneous node, it will take over its functionality. The worst-case scenario in a time triggered distributed real-time system is an erroneous node, which disturbs the communication between the others by sending messages at the wrong instant in time. Such as erroneous node is called "babbling idiot" failure. If this error condition happens, the rest of the distributed real-time system will also fail and it could end in a complete system shutdown (e.g. in automotive or airplane applications). Therefore it is of paramount importance to ensure that the TTP/C protocol and all its building blocks will always behave in a fail-silent manner under all possible conditions.

The main objective of the fault injection experiments carried out has been to test the fail-silent assumption. The faults injected were marked by the specifications of the FIT project [5].

VFIT and AFIT can access to different system parts. Thus, the combination of both techniques [4] or several techniques [12] to validate the same system is a very appreciated resource.

VHDL-based fault injection focuses the experiments on the PCU and the Instruction Register (IR) of the PCU, the Time Control Unit (TCU) and the CRC unit [13]. The parameters of the fault injection experiments have been:

TTP/C-C1 model:

- *Fault Location*: all atomic signals and variables of the PCU, the IR, the CRC and the TCU module.
- *Number of injections*: 3000 in the PCU and IR, 2000 in the CRC and TCU.
- *Injection instant*: randomly selected, distributed uniformly during the 1st TDMA round.
- *Fault duration*: transient faults with a random duration in a range from ½ time slot to 1 time slot.
- *Fault models*: bit-flip (in storage), pulse (in combinational logic), indetermination and delay.

TTP/C-C2 model:

- *Fault Location*: all atomic signals and variables of the IR, the CRC and the TCU module.
- *Number of injections*: 1000 in each module.
- *Injection instant*: randomly selected, distributed uniformly during the 1st and 2nd TDMA round.
- *Fault duration*: transient faults with a random duration in a range from ½ time slot to 1 time slot.
- *Fault models*: bit-flip (in storage), pulse (in combinational logic), indetermination and delay.

Physical fault injection at pin level focuses the experiments on the TTP/C controller pins, being divided into two main groups: the CNI and the TTP/C Controller bus connections. The parameters for the different experiments have been:

TTP/C-C1 controller:

- *Fault Location*: controller bus connections, CNI interface, MEDL interface and the hard Reset line.
- *Number of effective injections*: 3726 in the bus connections, 7432 in the CNI busses and control lines, 12342 in the MEDL busses and control lines and 859 in the Reset line.
- *Injection instant*: randomly selected during the execution of the application.
- *Fault duration*: transient faults from 1 µs to 600 µs.
- *Fault models*: single stuck-at 0 and stuck-at 1.

TTP/C-C2 controller:

- *Fault Location*: controller bus connections, local Bus Guardian (BG) [14] oscillator and CNI interface.
- *Number of effective injections*: 7283 in the bus connections, 1000 in the BG oscillator and more than 10000 in the CNI busses.
- *Injection instant in the bus connections and BG oscillator*: synchronised with a frame transmission or reception.
- *Injection instant in the CNI busses*: synchronised with a write or read host access to memory.
- *Fault duration*: transient faults from 500 ns to 12 µs.
- *Fault models*: single stuck-at 0 and stuck-at 1 and double stuck-at (0,0) and stuck-at (0,1).

5. Results

5.1. VHDL-based fault injection results

Using the VHDL model we can test the correct execution of the algorithms involved in the TTP/C protocol. In fact, during the experiments it was shown the usability of the technique due to its high precision in deciding the fault location and the fault model. A summary of the main results obtained with the TTP/C-C1 controller is presented in Table 2. As can be seen, the percentage of activated errors is similar for all modules. Those activated errors that cause no-effect in a target are denoted as *non-effective errors*. One of their more frequent reasons is the target redundancy. Although the percentages of detected errors are very high, we can see that the CRC module is the most sensitive, as it presents the lower detection percentage. An error not correctly detected will derive in a failure (denoted as *undetected errors* in Table 2).

Once the error has been detected, it can be *recovered* or it can cause a failure (denoted as *non recovered* in Table 2). These failures can be produced because the detected error has been incorrectly isolated or the target cannot be recovered after the detection.

Table 3 resumes the percentage of failures produced in all experiments respect to the propagated faults.

Table 2. TTP/C-C1 results.

	PCU	**IR-PCU**	**CRC**	**TCU**
Activated errors	38.17 %	38.10 %	37.15 %	39.15 %
Non effective errors	6.89 %	14.00 %	22.88 %	10.35 %
Detected errors	93.02 %	85.13 %	75.10 %	89.14 %
Recovered	98.87 %	98.25 %	99.28 %	99.14 %
Non Recovered	1.13 %	1.75 %	0.72 %	0.86 %
Undetected errors	0.09 %	0.87 %	2.02 %	0.51 %

Table 3. Percentage of failures respect to activated errors [13].

	IR	**PCU**	**CRC**	**TCU**
CORRECT	97.64 %	98.86 %	97.44 %	98.72 %
FAILURE	2.36 %	1.14 %	2.56 %	1.28 %

As can be seer, a non-negligible percentage of failures were found. These failures are provoked by a single fault injected in a node. This failing node affects all the system as it causes a cluster shutdown. That is, all nodes of the system stop transmitting frames, violating the fault hypothesis.

This fail silence violation is caused by the special configuration of the message schedule in the model. The system supplies two communication channels that allow to replicate messages, or to improve the communication bandwidth. It is up to the user to decide if one frame is sent by one channel or by both channels.

In the particular case analysed here, during the first slot, the first node sends only a frame in one channel. The injection of a transient fault at this moment causes an erroneous frame to be received by the rest of the nodes.

The fault causes an incorrect execution of the CRC calculus of the next frame to be transmitted. The frame with an erroneous CRC is transmitted. All the nodes in the cluster receive the frame that will be considered as an invalid frame. That is the expected behaviour. However, an erroneous implementation of the clique avoidance algorithm [15] produces a whole cluster shutdown if the erroneous frame is the first transmitted frame. With the help of these experiments, the manufacturer company that builds the TTP/C controller (TTTech [14]) was able to find the problem and solve it in the TTP^{TM}/C-C2. In order to check if the problem has been solved, several fault injection experiments have been done in the TTP/C-C2. The results of these fault injection experiments are shown in Table 4.

The main conclusions obtained in these last experiments were:

- No fail silence violations have been found (all detected errors are recovered).
- The TCU is the most sensitive point, as it has the biggest percentage of activated errors.
- In the IR experiments, a relatively high number of activated errors (in both columns) are detected by the error detection mechanisms of the TTP/C-C2.
- The biggest part of errors in the TCU and CRC are covered by the intrinsic redundancy of the system.

Table 4. TTP/C-C2 results.

	IR-PCU (1st TDMA)	IR-PCU (2nd TDMA)	CRC	TCU
Activated errors	18.30 %	18.40 %	14.80 %	60.60 %
Non effective errors	22.40 %	20.65 %	72.30 %	51.16 %
Detected errors	77.60 %	79.35 %	27.70 %	48.84 %
Recovered	100 %	100 %	100 %	100 %
Non Recovered	0 %	0 %	0 %	0 %
Undetected errors	0 %	0 %	0 %	0 %

5.2. Fault injection at pin level results

5.2.1. TTP/C bus connection lines

The babbling-idiot scenario at the drivers' level can be provoked with faults in the controller bus connections [4]. That means basically in the transmission and reception lines of one channel (single faults) or both channels (double faults). The injection is synchronised with the frame transmission (or reception), being possible to inject the fault before, during or after the transmission. The following issues were observed:

1. Although the error appears before, during or after the transmission, the frame will be considered as invalid by the receiver nodes.
2. The reinstatement of the node in the cluster is quicker if the error appears during a transmission instead of during a reception.
3. Drivers must assure the physical isolation of the transmission lines of the assigned transmission time slot. Otherwise, the error will be propagated causing the exclusion of a non-faulty node.
4. In spite of the precision of the clock synchronization algorithm, the perfect synchronization of the local clocks is not always possible due to bad oscillators or degraded performance. Thus, it is high recommended to validate the communication protocol in the presence of babbling idiot failures that occur in the limits of a transmission window because in a degraded performance they will have the condition of an asymmetric fault [4].

5.2.2. Bus Guardian Oscillator

The role of the local bus guardian in a bus topology is to avoid that out-of-time frames are transferred into the bus. The clock frequency is independently generated to the main TTP/C clock frequency. In fact, our working nodes supply a 16 MHz oscillator just to be used by the local bus guardian. 1000 faults were injected in the local bus guardian oscillator, synchronised with the transmission and reception of frames.

The bus guardian has detected the perturbations on its oscillator. It takes several TDMA rounds to reintegrate the node in the cluster again, but no collisions or violation of the fault hypothesis were observed during these reintegration actions.

5.2.3. CNI Address and Data Busses

There are two types of data exchanged between the TTP/C controller and the host controller: messages and status/control information. Data are exchanged through the CNI. This section describes the observations obtained with those faults oriented to test the different memory areas.

Status/control area

Injections in this area show the collaboration between a physical fault injection technique and simulation.

Among the status/control registers in the CNI are the life-sign registers. As long as the host is active, it has to periodically update its life-sign register in order to notify the TTP/C controller that messages are still being read/written (it is alive).

In case the TTP/C controller detects an error in the host life-sign updating, it changes from an active state into a passive state waiting the host to be again ready to execute the application. Moreover, the controller triggers the only existing external signal between the controller and the host: the TTP/C Interrupt signal. The interrupt informs the host that the controller has detected an error.

During the experiments pertaining to the life-sign algorithm, the injected faults modified the value stored in the life-sign register. Figure 3 show the analysed cases.

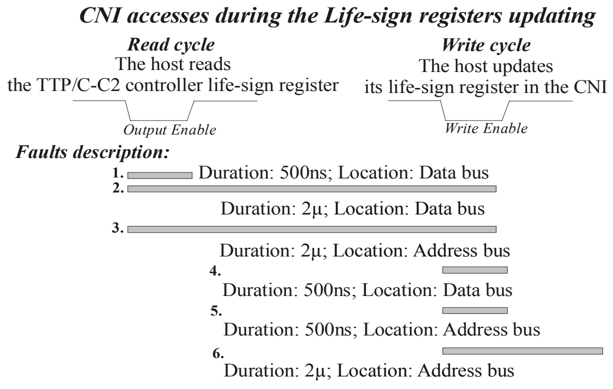
CNI accesses during the Life-sign registers updating
Read cycle
The host reads
the TTP/C-C2 controller life-sign register
Output Enable
Write cycle
The host updates
its life-sign register in the CNI
Write Enable
Faults description:
1. Duration: 500ns; Location: Data bus
2. Duration: 2µ; Location: Data bus
3. Duration: 2µ; Location: Address bus
4. Duration: 500ns; Location: Data bus
5. Duration: 500ns; Location: Address bus
6. Duration: 2µ; Location: Address bus

Figure 3. Injections during the Life-sign updating algorithm in the TTP/C-C2 controller.

Because the host busses are disturbed with faults, host exceptions are frequent in cases 2 and 3. Operating system exceptions are observed specially in case 6. Cases 1 and 4 cause a bit-flip in data and consequently, the host updates its life-sign register with a wrong value. In case 5, the host does not update the register. In order to carry out a detailed analysis of cases 1, 4 and 5, the VHDL-based fault injector was used to reproduce them and to compare the results with the pin-level fault injector.

During the monitoring of the transmission lines of the faulty node, it was observed that the node does not transmit any frame (both channels were observed) in the following TDMA round after the fault injection. Due to this silence of the faulty node, the rest of the cluster members judge this node as inactive.

That means that the TTP/C controller is aware of the host error in updating the life-sign and follows the condition of fail-silence. But the TTP/C interrupt should be raised, warning the host about the detection of the error. However, it was observed that the interrupt status flag register is not updated and the interrupt is not raised, mismatching the specifications [14]. It is considered as an error in the implementation of a specific algorithm that could be early detected in the VHDL model using fault injection, before the defect will be propagated into the prototype.

Message area

A fault injection campaign was carried out altering the message values, resulting in host exceptions and fail-silence violation in the value domain.

Two different fault effects are observed that lead the system to a value failure. Due to a fault that forces (0 or 1) a line during some time, part of the data written in the CNI becomes erroneous, but the node sends them to other node.

Firstly, a fault causes a bit-flip in one or several words, depending on the number of accesses during which the line remains stuck. Secondly, the fault causes a whole ineffective write access. Transferred data do not get to update the content of the selected memory address. Both cases imply multiple errors.

The frame to be transmitted by the node is built using the contents of the CNI message area. But, as these contents are wrong, the frame will be syntactically correct but semantically incorrect. A good solution to detect semantic errors is the use of Error Detection Codes able to detect multiple errors [9].

6. Summary. Conclusions and future work

The combination of two different fault injection techniques has brought us the opportunity of a deeper validation of the TTP/C communication controller.

The PCU and its Instruction Register, the TCU and the CRC Unit have been put to the test using VHDL-based fault injection, which achieves a high control to define the fault location and duration as well as the injection instant. During the experiments, an error in the clique avoidance algorithm implementation was observed. This error leads the TTP/C-C1 to a complete shutdown under specific conditions. Thanks to the validation test, the error has been solved in the second version of the controller (TTP/C-C2).

On the other hand, physical fault injection at pin level has been very useful to generate babbling idiot errors in delivered frames through the communication bus, validating the fault hypothesis of the distributed system in the presence of arbitrary faults. Moreover, some injections have been carried out directly on the local bus guardian oscillator, assuring that the reinstatement of the controller does not provoke frame collisions.

Faults injected at the CNI cause that the faulty node sends a semantically incorrect message but syntactically correct and on time, which is considered a failure in the value domain. These experiments are useful to experimentally obtain the coverage of different error detection codes and protection strategies.

Finally, different fault injection techniques can be used to study specific faulty scenarios, in order to put a high level of confidence in the validation results. That is the case of the life-sign algorithm, where a set of physical faults at pin level reveals a mismatch of the prototype with the specifications. The case was reproduced in the VHDL model obtaining the same results.

In the future, we will continue with the fault injection experiments, in order to test the effectiveness of different detection and recovery mechanisms added to the TTP/C controller.

References

[1] J. Clark, D. Pradhan, “Fault Injection. A method for validating computer-system dependability”, IEEE Computer, June 1995.

[2] M. Sueh, T. Tsai, R. Iyer, “Fault Injection Techniques and Tools”, IEEE. Computer, pp. 75-82, April 1997.

[3] P. Folkesson, “Assessment and Comparison of Physical Fault Injection Techniques”, Technical Report nº 377, 1999.

[4] S. Blanc, J. Gracia, P.J. Gil, “A Fault Hypothesis Study on the TTP/C using VHDL-based and Pin-Level Fault Injection Techniques”, Procs. of 17th International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT 2002), pp. 254-262, November 2002.

[5] C. Madritsch, “Fault Injection for the Time-Triggered Architecture (FIT)”, Supplement of the 2001 Int. Conference on Dependable Systems and Networks (DSN 2001), Special Track: European Dependability Initiative, Göteborg, Sweden, pp.D-25-D-27, July 2001.

[6] J. R. Martínez, P.J. Gil, G. Martín, C. Pérez, J. J. Serrano, “Experimental Validation of High-Speed Fault-Tolerant Systems Using Physical Fault Injection”, Procs. of 7th International Working Conference on Dependable Computing for Critical Applications (DCCA-7), pp. 233-249, 1999.

[7] J.C. Baraza, J. Gracia, D. Gil, P.J. Gil, “A Prototype of a VHDL-Based Fault Injection Tool: Description and Application”, Journal of Systems Architecture, I.S.S.N. 1383-7621, 47(10):847-867, 2002.

[8] Model Technology, “ModelSim SE/User’s Manual, Version 5.5e”, Septiembre 2001.

[9] S. Blanc, P.J. Gil, “Improving the Multiple Errors Detection Coverage in Distributed Embedded Systems”, Procs. of Symposium on Reliable Distributed Systems (SRDS2003), pp. 303-312, October 2003.

[10] “TTP/C C1 Controller. Specification of the TTP/C C1 Controller”, TTTech Computertechnik GmbH, Available at http://www.tttech.com.

[11] “AS8202. Functional Description of the AS8202 (preliminary) D-032-S-10-026. Version 1.0”, TTTech, Available at http://www.tttech.com, September 2001.

[12] S. Blanc, A. Ademaj, H. Sivencrona, J. Torin, P. Gil, “Three Different Fault Injection Techniques Combined to Improve the Detection Efficiency for Time-Triggered Systems”, Procs. of IEEE Design and Diagnostic of Electronic Circuits and Systems (DDECS2002), pp. 412-415, April 2002.

[13] J. Gracia, J.C. Baraza, D. Gil, P.J. Gil, “Using VHDL-Based Fault Injection for the Early Diagnosis of a TTP/C Controller”, to be published in IEICE Transactions on Information and Systems, Vol. E86-D, Nº 12, December 2003.

[14] Specifications of the TTP/C Communications Controller, Available at http://www.tttech.com.

[15] H. Kopetz, TTP/C Protocol, TTTech 1999, Available at http://www.ttpforum.org.