

Fault Tolerance Evaluation Using two Software Based Fault Injection Methods

Astrit Ademaj
Vienna University of Technology
Real-Time Systems Group
Treitelstr. 3/182-1
A-1040 Vienna, Austria
ademaj@vmars.tuwien.ac.at

Petr Grillinger, Pavel Herout,
University of West Bohemia
Dept. of Computer Science
Univerzitni 8
Pilsen, Czech Republic
{pgrillin, herout}@kiv.zcu.cz

Jan Hlavicka
Czech Technical University
Dept. of Computer Science
Karlovo namesti 13
Prague, Czech Republic
hlavicka@cslab.felk.cvut.cz

Abstract

A silicon independent CBased model of the TTP/C protocol was implemented within the EU-founded project FIT. The C-based model is integrated in the C-Sim simulation environment. The main objective of this work is to verify whether the simulation model of the TTP/C protocol behaves in the presence of faults in the same way as the existing hardware prototype implementation. Thus, the experimental results of the software implemented fault injection applied in the simulation model and in the hardware implementation of the TTP/C network have been compared. Fault injection experiments in both, the hardware and the simulation model are performed using the same configuration setup, and the same fault injection input parameters (fault injection location, fault type and the fault injection time). The end result comparison has shown a complete conformance of 96.30%, while the cause of the different results was due to hardware specific implementation of the Built-In-Self-Test error detection mechanisms.

1. Introduction

Fault injection (FI) is a widely used technique for fault tolerance evaluation. Fault injection can be applied in the simulation model of the system under test or in the hardware implementation (in the prototype or in the final stage). Each of these approaches has its advantages and disadvantages, flexibility and ease of implementation on a simulation, more convincing results on a hardware implementation.

Within the EU-founded project FIT (Fault injection for Time Triggered Architecture) of the 5th Framework Program Information Societies Technology several different fault injection methods are being used for dependability evaluation of the Time Triggered Architecture (TTA) [10]. At the core of the architecture is the communication protocol TTP/C [3]. A prototype

micro-programmable version of a TTP/C communication controller chip has been designed and implemented during EU-founded ESPRIT OMI project TTA [2]. A silicon independent C-Based model of the TTP/C protocol was implemented within the FIT project. The C-based model is integrated in the C-Sim simulation environment. The fault injection methods used in the FIT project include hardware fault injection (pin-level, heavy-ion), and several methods of software implemented fault injection (in prototype hardware, in the C-Sim simulation model and in the VHDL model). The comparison of results of the software implemented fault injection (SWIFI) into the prototype hardware implementation, and the SWIFI in the simulation model is the focus of this work. The fault injection experiments in the prototype hardware implementation of the TTP/C communication controller was performed at the Technical University in Vienna, whereas the fault injection in the simulation model at the Czech Technical University in Prague and University of West Bohemia in Pilsen. These two methods were selected for comparison because the executions of the fault injection experiments are identical.

In the following text, we will refer to the two SWIFI methods using different names:

- *HW-SWIFI* – SWIFI method applied into the hardware implementation.
- *Sim-SWIFI* – SWIFI method implemented in the C-Sim based TTP/C model.

The rest of the paper is organized as follows; section two briefly describes the TTP/C protocol. In the section three the C-Based model of the TTP/C protocol is presented with the workload applications. Section four describes the comparison objectives and description of the experiments. In the section five the tuning of the C-based model is described. The comparison results are presented in the section six and the paper finishes with the conclusion in section seven.

2. TTP/C protocol description

A TTP/C network [4] consists of a set of electronic

modules (called TTPnodes) that are connected by two replicated channels. A TTP node consists of a TTP/C communication controller, a dual-ported RAM called the Communication Network Interface (CNI) and a host controller. The host executes the application software whereas the TTP/C communication controller executes the TTP/C protocol [3].

The CNI (implemented as a Dual Ported RAM) is an interface between the application layer and protocol layer of a TTP/C node. The CNI consists of:

- ❑ *Status Area* – contain the current status of the protocol (read only access from host).
- ❑ *Control Area* – contain host control information for the TTP/C controller.
- ❑ *Message Area* – transmitted messages among the nodes are stored in this area.

A set of interconnected and synchronized TTP nodes is denoted as a TTP cluster. Access to the bus is controlled by a cyclic time-division multiple access (TDMA) scheme. Each node in the system has a unique transmission slot, which is specified in the static data structure of the TTP/C controller called Message Description List (MEDL) [3]. The sequence of TDMA slots in which each module sends its frames forms a TDMA round. After a TDMA round is completed, the next TDMA round, with the same temporal access pattern but possibly different frames, is started. The number of different TDMA rounds determines the length of the cluster cycle. After a cluster cycle is finished, the transmission pattern starts over again with the start of the next cluster cycle.

3. C-based model

Silicon independent C-code of the TTP/C protocol has been implemented based on the TTP/C specification [3]. During the process of implementation some inconsistent statements in the specification [3] had to be resolved. However, this C code cannot be used as an executable model. To verify the C-based TTP/C model, this model was embedded into the C-Sim [1] simulation environment. The C-Sim enables the execution of several instances of TTP node activity in an interleaved mode using the global simulation time concept, because C-Sim allows a *Simula*-like (co routine) style of pseudo-parallel computing. C-Sim is suitable for creating large and portable simulation programs. It has the form of a library of basic object types and operations on them, which allows enhancing the standard object types with new attributes and methods fulfilling the needs of a concrete model. C-Sim is in fact an extension of the C language that was taken from the programming language SIMULA. The library provides SIMULA-like resources from the system classes SIMSET and SIMULATION.

3.1. Workload applications

The *sine-wave* application is used in the simulation model of the TTP/C protocol as a workload application. The sine-wave application requires four nodes (the minimum number for correct TTP/C protocol operation) and performs all protocol-required tasks. Each node performs the same operation, they measure the value of an external sine wave signal and the measured value is transmitted on the bus so that the nodes may compare their measured values with each other.

The nodes are grouped into two separate FTUs (*Fault Tolerant Units*). Each of the FTUs provides one measured value. The first FTU is formed by a single node, so-called the reference (**R**) node. It computes the reference value of sine wave. The second FTU comprises three nodes, forming a TMR (*Triple Modular Redundancy*) FTU. Such an FTU ensures correctness in the value domain even if one of the nodes fails. The result is determined by majority voting on all valid results from the nodes. The node under test is denoted as F node. G1, G2 are replicas of F. All three nodes F, G1, G2 vote and output voted final value.

In fig. 1 a screenshot of the visual interface of the sine-wave application is presented. The interface enables to quickly setup an experiment and to observe the model behavior. To perform the Sim-SWIFI experiments, this interface is not needed, although it provides useful insight when tracking any misbehavior of the model.

Brake-by-wire (BBW) is used as application workload for the HW-SWIFI setup. BBW is a distributed simulation program implemented from Volvo Technological Development [8]. The BBW requires 4 TTP nodes: two replica nodes acting as *wheel* nodes, one *pedal* node and one additional node which generates data that would be send by other nodes in a realistic environment. A realistic fault tolerant brake-by-wire application requires 10 nodes; two nodes forming the fault tolerant unit for each wheel denoted as front left (FL), front right (FR), rear left (RL) and rear right (RR) and two nodes reading the pressure applied on the brake pedal. The hardware workload application has the same node configuration setup (same number of nodes, same TDMA slot length) as the sine-wave application.

4. Comparison objectives

In the simulation environment it has been proven that the new C-based model of the TTP/C protocol behaves according to the specification [3] in the fault-free condition. The aim of this work is to check whether the model behaves in the same way as the hardware also in the presence of faults. The main objective of this work is to verify the C-Based model implementation of the TTP/C communication protocol and remove any found discrepancies. Thus, the targets of the both SWIFI tools

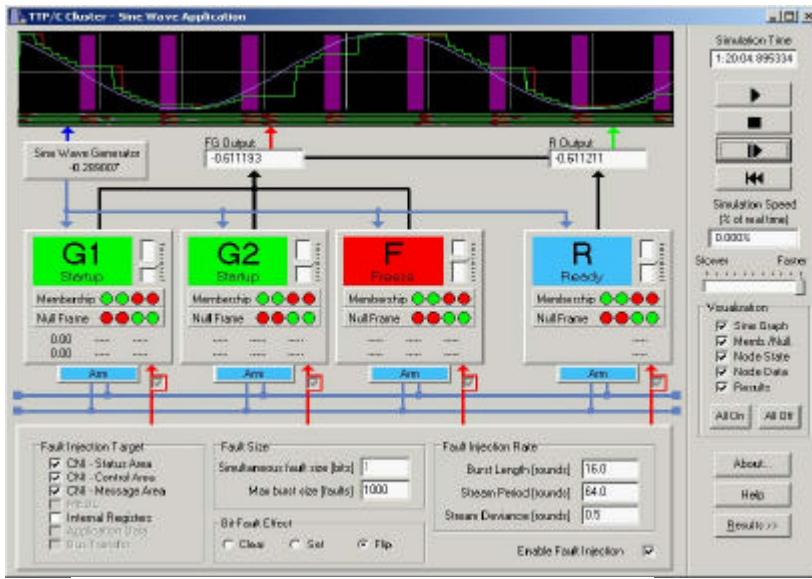


Figure 1 – A snapshot of the Sine-Wave Application

are not application data, but the status and the control data in the interface (CNI) between the host and the TTP/C protocol layer. The second objective is to confirm the fault injection results obtained by the HW-SWIFI method and possibly point out any incorrect behavior of the protocol implementation in the hardware prototype. The tuning of the model to obtain the best possible behavior (according to the specification) is the third objective of this comparison.

4.1. HW – SWIFI

Software implemented fault injection tool emulates physical faults into the CNI. Fault injections are performed at run-time. The HW-SWIFI emulates transient bit flip faults, which are considered to be the most common physical faults in computer systems. This model is in accordance with most SWIFI tools [5, 6, 7]. The fault injection experiments are performed in the fourth node. Faults are injected in the *application* state [3] and before the node's sending slot. If the fault does not affect the behavior of the node (the fault has not been activated or the fault has been masked), the node performs restart.

The fault injection experiments into the CNI are conducted from the TTP/C communication controller – TTP/C-C1 (CNI Status Area), and from the host controller MC68360 (CNI Control Area). Therefore, we have two fault injection campaigns. The fault injector code for the first campaign is implemented as a part of protocol code and for the second case a part of the host application. The decision for separating the fault injection into two campaigns is due to the read-only access of the host controller to the CNI Status Area [3]. This design decision

(host has read-only access to the CNI Status Area) is made in order to prevent the malicious host to write in the CNI Status Area.

The fault injector is implemented using the code insertion technique. The HW-SWIFI perform systematic bit-flip fault injection over the list of predefined CNI memory addresses, and is able to perform 2 fault injection experiments per second. Results of HW-SWIFI experiments are stored in a text file. Each single fault injection is described by the following parameters: i) the CNI address ii) the bit-mask which specifies a single bit for FI and iii) the EDM, which has detected an error (if any error is detected).

4.2. Sim-SWIFI

The results of the HW-SWIFI experiments are used as input for Sim-SWIFI experiments. The process of fault injection goes through following steps:

- ❑ read the fault injection parameters, i.e., CNI address and bit-mask from the HW-SWIFI output file
- ❑ perform fault injection (according to parameters from HW-SWIFI),
- ❑ retrieve result from the fault injection experiment (error code / no error),
- ❑ write fault injection results in an output file,
- ❑ continue with the next experiment.

If no error is detected for a specified time period (4 TDMA rounds), the experiment is considered complete, the node is restarted and the Sim-SWIFI continues with the next experiment. The method used for Sim-SWIFI is the same as in the HW-SWIFI. A single task is scheduled for a chosen time to perform fault injection (single bit flip).

The HW-SWIFI method requires a different approach for the CNI Status and CNI Control Area [3], since the CNI Status Area is read-only for the application layer (host) in the hardware, whereas in the C-Sim model there is no such restriction. The Sim-SWIFI injects faults in all fields of CNI in the same way. The simulation speed of the TTP/C cluster simulation is near to the real-time execution speed (the simulation executed in PC workstation AMD 800 MHz gave the simulation speed higher than the real-time execution in the hardware implementation).

5. Tuning the C-based TTP/C model

Any TTP/C cluster configuration contains a number of parameters that affect its performance. The specification

[3] allows a wide range for these parameters and so even two exactly the same applications may run under quite different conditions. In order to get compatible results from two experiments, both experiments must run on a *cluster* that uses the same parameters. Some of them may be set freely whereas some of them are physical properties of the hardware implementations, which may not be easily altered. The list of these parameters [3] which had to be tuned in the simulation model to match the HW parameters is:

- ❑ *IFG Duration* – implementation specific.
- ❑ *Microtick/Macrotick ratio*.
- ❑ *Internal Clock Drift* – the physical drift of controller’s internal clock - random value.
- ❑ *Bus Transmission Speed* – one of the predefined values.
- ❑ *Transmission Delay* – a small delay introduced when transmitting on long wires.
- ❑ *Interrupt Latency* – a small latency between the raising and handling of interrupt.

6. Results

In table 1 the results of the fault injection experiments of the HW-SWIFI and the Sim-SWIFI are presented.

A specific injected fault can trigger a specific error, which can be detected by a specific EDM. The relation

Table 1. HW-SWIFI and Sim-SWIFI Fault injection experiments

	Complete		Partial		Unmatched	
	573		17		5	
<i>No. of Exp.</i>	HW	Sim	HW	Sim	HW	Sim
<i>Error detection mechanism</i>						
<i>Protocol</i>	19	19	-	16	-	-
<i>Host</i>	4	4	1**		-	-
<i>Membership Loss and Change</i>	82	82	-	1	-	-
<i>Mode Change</i>	1	1	-	-	-	-
<i>BIST</i>	2	2	16*	-	5***	-
<i>No Error activated/detected</i>	465	465	-	-	-	5

between a specific injected fault, causing a specific EDM to react, has been compared in the HW-SWIFI and in the Sim-SWIFI model. We consider HW-SWIFI and Sim-SWIFI experiments results to be conformant if an error detection mechanisms detects an error, caused by the same injected fault, in both methods.

Complete conformant results denotes the part of the experiments where the same injected fault in the HW-SWIFI and the Sim-SWIFI, caused the same error detection mechanisms (the list of the EDM is given in [3]) to react.

Partial conformant results means that a specific injected fault has triggered an error which has been

detected different error detection mechanism in the hardware and the simulation environment.

Unmatched result means that a specific injected fault triggered an error in one whereas it has triggered no error in other method.

The final result comparison has shown a total conformance of 96.30 %. The remaining differences can be explained by a different *level* of experiment organization. As can be seen from the table 1 most of fault injection result differences occurred in the fault injection experiments that triggered the Built-In-Self-Test (BIST) error detection mechanism. BIST mechanisms in the prototype implementation of the TTP/C communication controller include a list of self-checks, which are hardware specific (excluding a software *watchdog*) and cannot be *implemented* in the simulation environment.

6.1. Partial matched results

Detailed analysis of the 16 experiments (*) in the *partial* part, has shown that the cause of the result difference was due to hardware specific implementation of the Built-In-Self-Test error detection mechanisms. Errors triggered by these fault injection experiments in both methods have been detected, but the in the simulation environment there was no hardware BIST, and therefore error has been detected by another error detection mechanism. The remaining difference (1** experiment) was due to the specific implementation of the error detection mechanisms in the hardware and the simulation. The EDM performed different check order for special error condition in hardware and different check order for specific error conditions in the simulation. Both EDM implementations are not in contradiction with the TTP/C specification [3].

6.2. Unmatched results

In the hardware implementation, the host can set an request for a BIST self check from the communication controller. The cause of different experimental results in the *unmatched* part (5*** experiments) was due to the invalid BIST self-check request at the hardware implementation. Such a self-check (like RAM check, register file check) request is *invalid* in the simulation environment (since it is hardware specific, and does not have any sense to implement it in the simulation environment) and therefore no error was detected.

Table 2. Result comparison

	CNI Status & Control Area	
Number of experiments	595	100 %
Complete	573	96.30 %
Partial	17	2.86 %
Unmatched	5	0.84 %

7. Conclusions and future work

Fault injection is usually used to assess the effectiveness of the error detection mechanisms and to observe the behavior of the system under test in the presence of faults. In this work we have used fault injection to verify the implementation of the simulation model. Two software implemented fault injection techniques have been used to improve and verify the C-Based TTP/C model, by comparing the fault injection experiment results of the prototype hardware implementation and the model executed in the simulation environment. The experiments in the hardware and the simulation environment are performed using the same setup, and the same fault injection input parameters (the fault type, fault location and the fault injection time). In addition, we proved that the modified C-based model behaves the same way in the fault-free conditions and in the presence of faults. The comparison of two different FI methods has several additional effects. It helped to validate the C-Based TTP/C model, discover several bugs within the model, and to improve its behavior. The results of the comparison has shown a complete conformance of 96.30%, while the cause of the different results was due to the hardware specific implementation of the Built-In-Self-Test error detection mechanisms, which are not present in the simulation environment.

So far, the Sim-SWIFI methods have been used for fault injection into the CNI memory area. The Sim-SWIFI method may be used also for injection into other memory areas. The future work would be to compare the results of the HW-SWIFI and Sim-SWIFI experiments targeting node configuration data contained in the Message Description List (MEDL).

It should also be noted that the experiments performed using the SWIFI methods are limited by the following facts:

- The fault injection has been performed only at a single point during the cluster cycle. This way only a portion of the protocol implementation is covered and the validation of the C-based model cannot be considered complete. Both the HW-implementation and the C-Sim implementation allow fault injection at multiple points in time, so the experiments could be extended to cover a wider area of possible faults in the future.

- The compared SWIFI experiments targeted only the application-independent items of the CNI. Fault injection into application data requires a fault tolerant application that would handle all recoverable faults and report correctly all critical faults.

Finally, the TTP/C C-Based model can be used in the future also for the validation of the protocol improvements, and investigation of the behavior of future changed versions of the TTP/C protocol.

Acknowledgment

The research was in part supported by the grant of 5th Framework Program Information Societies Technology: IST-1999-10748 Fault Injection for Time Triggered Architecture (FIT).

8. References

- [1] *C-Sim*, Available at <http://www.c-sim.zcu.cz>
- [2] *Time-Triggered Architecture*. <http://www.vmars.tuwien.ac.at/projects/tta/>
- [3] Kopetz, H.: *TTT/C Protocol Specification*. Available at <http://www.tttech.com>.
- [4] Krüger, A., Kopetz, H.: *A Network Controller Interface for a Time-Triggered Protocol*, Symposium on Future Transportation electronics: Multiplexing and In-Vehicle Networking, Dearborn, Mich. SAE International, 1995, (pp. 1–10).
- [5] Carreira, J., Madeira, H., Silva, J. G.: *Xception: Software Fault Injection and Monitoring in Processor Functional Units*, IEEE Transactions on Software Engineering, vol. 24, no. 2, Feb. 1998.
- [6] Kanawati, G. A., Kanawati, N. A., Abraham, J. A.: *FERRARI: A Tool for the Validation of System Dependability Properties*, (FTCS-22), Boston, Massachusetts, 1992.
- [7] Barton, J., Czeck, E., Segall, Z., Siewiorek, D.: *Fault Injection Experiments using FIAT*, Transactions on Comp, Vol. 39, No. 4, 1990.
- [8] Lennevi, J.: *Simple vehicle model with ABS control for the FIT project*. Volvo Technological Development Memorandum 01-01-11
- [9] Kopetz, H.: *Real-Time Systems, Design Principles for Distributed Embedded Applications*, Kluwer Academic Publishers, 1997, 338pp.
- [10] Kopetz H.: *Time-Triggered Architecture*. IFIP International Workshop on Dependable Computing and Its Applications (DCIA 98), January 12 - 14, 1998, Johannesburg, South Africa
- [11] Manzone, A., et al.: Fault tolerant automotive systems: An overview. Proc. IEEE International On-Line Testing Workshop 2001, pp.117-121