Computing and Informatics, Vol. 23, 2004, 51–76

DEPENDABILITY EVALUATION OF TIME TRIGGERED ARCHITECTURE USING SIMULATION*

Stanislav Racek, Pavel Herout, Jan Hlavička†

Department of Computer Science, University of West Bohemia
Univerzitni 22, 306 14 Pilsen, Czech Republic
e-mail: {herout, stracek}@kiv.zcu.cz

Manuscript received 31 March 2003; revised 13 February 2004
Communicated by Norbert Frištacký

Abstract. The method presented in this paper uses a generic C-language written simulation model of an embedded distributed computer system aimed for a safety-critical control application. The considered system is built using Time Triggered Architecture (TTA) concepts. The aim of the presented simulation method is to evaluate the system capability to tolerate a chosen category of faults. The model, being written in ANSI-C, is portable and machine-independent. Its structure is modular and flexible, so that the system to be studied and the experiment setting can easily be changed. The functionality of this model is demonstrated on a set of fault injection experiments aimed mainly to evaluate the correctness of the Time Triggered Protocol (TTP/C) that implements the abstract concepts of TTA. These experiments were done within the EU/IST project Fault Injection for Time triggered architecture (FIT).

Keywords: Dependability, simulation, TTA, fault-injection, C-Sim

1 INTRODUCTION

Dependability validation is one of the most important steps in the design of safety-critical embedded computer systems [12]. It includes not only validation of hardware

* The research was in part supported by the Grant Agency of the Czech Republic, project no. 102/03/0672. Another source of funding was the grant of 5th Framework Program Information Societies Technology: IST-1999-10748 Fault Injection for Time Triggered Architecture (FIT).

and software components but above all validation of resilient system behavior, especially the system's reaction to the faults and asynchronous real-time events in its environment.

A broad spectrum of validation methods have been suggested so far. On one end of the spectrum there are exact mathematical methods which enable a formal proof of correctness of the algorithm. A qualified survey of these methods can be found in [16]. The opposite extreme is the testing (probabilistic or deterministic) of the whole system.

Each of these basic approaches has both advantages and disadvantages, therefore methods combining these two approaches are continuously being looked for. The most important disadvantage of the formal proving of algorithm correctness is the syntactical and semantic gap between the verified model and a really implemented system. To be able to create a formal model, we usually need to use many simplifications which increase this gap. Moreover, fully automatic verification environments that cover the complete system from the high-level specification to the hardware are beyond the current state of the art. Program (or the whole system) testing as a validation method is the most popular approach, although it is well known to be incomplete. Unlike the testing of hardware, a theory of the completeness of software tests has not been formulated yet. Moreover, as a rule it is not possible to test the most interesting hazardous situations on a real system (such as an airplane or a power station).

Simulation[1] is one of the standard methods of fault tolerance (FT) evaluation that lies somewhere between the basic approaches mentioned above. The main advantage of simulation is above all the fact that it is flexible and can easily be adapted to an environment which is close to reality. Moreover, it can be (like a method of formal verification) used within one phase of system design. The main disadvantage of simulation is its experimental nature, so the utilization of functional validation for a given purpose (i.e. testing) requires either a lot of simulation experiments with randomly generated parameters or a connected theory how to choose the parameters of the experiments (i.e. test input vectors) in order to reach a sufficient reliability of the final statement of the (modeled) system evaluation.

Clearly, the main problem of a simulation model creation is how to choose the model abstract level to reach a given goal of the functional validation of an FT system. As a next step it is necessary to decide which simulation language should be used. On one side of the spectrum of possibilities there are low-level hardware describing languages, such as VHDL. On the opposite side of the spectrum are high-level languages, such as Simula or Beta. Fault injection (FI) can be performed on a simulation model of the system to be evaluated. For example the VHDL model makes it possible to change a register bit value or to put a short pulse at any signal line. A high-level language, for example, makes it possible to suppress any activity that is modeled as a pseudo-parallel discrete-time

[1] For the sake of simplicity we use the word "simulation" for both the process of the simulation model creation and for the process of the model utilization.

simulation process. Every simulation model should validate itself to be trustworthy.

This article presents a simulation method that was used for an experimental validation of the Time Triggered Architecture (TTA) that is a distributed embedded computer system architecture. It is intended to be used for a complex safety-critical control system of a transportation device, such as a car, an airplane, etc. Most of the work was done within the EC/IST project Fault Injection for TTA (FIT, 2000–2002). It was the objective of the project to validate experimentally, i.e. to test the system concepts of TTA, taking a prototype TTP/C controller chip (C1, see [19]) developed within the ESPRIT project TTA ([3]), as the basis. Here TTP/C means the communication protocol that is used for the interconnection of distributed system nodes. The aim of FI experiments was to determine the specified FT properties of TTA in a realistic application by using different HW and SW based fault-injection methods.

The experimental methods that were used for TTA evaluation can be briefly categorized as follows (the names of the responsible FIT partners are given in parentheses – see [4]):

- methods that use a real system (TTP/C evaluation cluster, see [19]) as the subject of the evaluation procedure:
  - TTP/C controller (C1 chip) pin-level injection (UPV Valencia),
  - TTP/C controller (C1 chip) heavy-ion impacts (Chalmers Univ., Gothenburg),
  - SWIFI (SW Implemented FI), i.e. utilization of a special thread that runs concurrently with the application program and performs a FI activity (TU Vienna),
- methods that use a simulation model of the evaluated system as the subject of the evaluation procedure:
  - VHDL written (i.e. low-level) model of TTP/C controller (CTI Villach, UPV Valencia),
  - C language written (i.e. higher-level) model of TTP/C controller and TTP/C cluster (CTU Prague and UWB Pilsen).

Our approach is based on the use of a C language based simulation model of the TTP/C protocol. Here we consider a discrete-time process-oriented simulation methodology that is used at the level of distributed system processes, messages and/or services (i.e., not at the level of electronic modules and/or signals). The model program code was created using the ANSI C language both as the specification language and simulation language. Language properties necessary to describe simulation (pseudo-parallel) processes were added in the form of the C-Sim [2] library. This library contains basic object types (including processes) and operations on them. Moreover, it contains a control procedure that performs the switching of

pseudo-parallel processes using the discrete model-time concept. C-Sim is in fact an extension of the C language that provides SIMULA-like functions from the SIMSET and SIMULATION library classes.

The simulation model was built on the basis of the TTP/C protocol specification [19]. There are some previous works which provide formal proofs of the correctness of individual specified TTP/C properties and services (e.g., [16], [15]). The simulation-based experimental testing of TTP/C properties can extend the correctness verification to more complex situations, including even the non-stable states of the TTP/C protocol state machine. The variety of methods that have been used within the FIT project enabled us to validate the simulation model (at least partly) using cross-check experiments with other project partners, see e.g. [1].

The rest of the paper is organized as follows: Section 2 describes briefly the properties of the system under test. The structure of the simulation model is presented in Section 3, together with some workload applications. Section 4 describes the fault injection experiment organization. Selected experimental results are presented in Section 5 and conclusions in Section 6.

2 SYSTEM TO BE EVALUATED

The principles of TTA were developed within the ESPRIT project TTA and they have been described most comprehensively in [12]. A special feature of this distributed computer system architecture is the fixed partitioning of the nodes time slots on the bus, i.e. every node has its statically assigned time slot within the basic bus cycle (TDMA round, see below). The method of access to the bus is then TDMA (Time Division Multiple Access) instead of the more common CSMA (Carrier Sense Multiple Access) used for Ethernet or the CAN (Control Area Network) industrial bus. To utilize the TDMA method of bus access requires the implementation of the so-called *sparse global-time base* measured in *macroticks*. Nodes can measure their local time in microticks (that are produced by an internal timer) and they need to synchronize an internal time value with the global time value with a precision of about one macrotick. TTA is a contradiction to an Event Triggered Architecture represented e.g. by a computer control system that uses the CAN industrial bus to connect its nodes. Both the basic instances of the embedded distributed system architecture have their advantages and disadvantages, see e.g. the qualified comparison of bus architectures in [17]. The following are the most important properties of TTA:

Predictable time responses. Due to the fact that all the system activity is time-driven (and the significant time points are the beginnings of a node slot), it its possible to guarantee system time responses, which is extremely important for so called *hard real-time systems*.

Composability. This property means that the locally evaluated properties of a node do not change after its integration into a time-triggered control system. It is an extremely important property for a system integrating person or

company, because they can rely on the specified properties of the connected device. This is due to the fact that the connected device has its own time space guaranteed at the bus.

Fault-tolerant properties. The time slots based organization of a system activity (or the underlying abstract concept of the sparse time base) makes it possible to implement fault-tolerant properties that are denoted as *Fail Silence in the Temporal Domain* and *Fail Silence in the Value Domain*. An explanation will follow later.

One of the possible implementations of the TDMA method is the TTP/C protocol which is a real-time communication protocol for the interconnection of electronic modules of distributed fault-tolerant real-time systems. Its specification can be loaded from the website [19]. Letter C indicates that the protocol meets the requirements for SAE class C automotive applications, because its utilization for this kind of applications seems to be very promising (see e.g. [6]). Nodes connected to the bus form a system called the TTP/C cluster. Each node (module) is treated as the *smallest replaceable unit* (SRU) of the system. It is also the basic error containment region, i.e. its malfunction should not propagate through the system. All nodes consist of three main parts – see Figure 1:

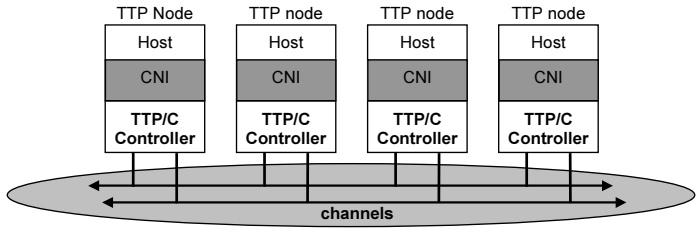


Fig. 1. Structure of a TTP/C cluster

Host processor (or host controller), which executes an application program (and has its own I/O interface with the controlled process). It can be equipped with a real-time operation system (TTPOS) that implements a simple kind of multithreading.

Dual-port CNI memory (Computer Network Interface), which serves as an interface between both the communication and host controller. It contains all the important data items including both the status of communication (Control Area of CNI) and the node "local view" of the application activity (Message Area of CNI). All the CNI data items are periodically updated with the smallest period of cluster cycle (see below).

Communication controller, which executes the TTP/C protocol. The communication activity is statically programmed, for this purpose every controller uses its own EPROM stored table (MEDL table – Message Descriptor List). The

second data structure that influences the protocol activity is the Control Area of CNI.

There is one more important part between a node and the TTP/C bus. This part is denoted as *Bus Guardian* (BG) and one of its functions is to prevent so called *babbling idiot fault*, which means a state when one (failed) node blocks the bus with its activity. At present another version of architecture – a star with a doubled star coupler (i.e. centralized BG) is being developed at TU Vienna and in the TTTech company.

The assumed bandwidth of the TTP/C bus is 2 Mbits/s. Then the basic period of the bus communication activity (TDMA round) has a typical duration value of about 10 ms. Within this period every node has its own slot assigned to transmit its messages. The slot duration can be assumed to be about 1 ms (all the given values clearly depend on the TTP/C cluster configuration). There are two basic kinds of TTP/C messages that a controller can broadcast within its slot:

N-frame – containing a message with application data, the maximum length of the message being 16 bytes,

I-frame – containing a message with cluster communication framework status, the maximum length being 12 bytes. It contains all the necessary information that enables a fallen TTP/C controller to start its normal communication activity immediately, which is called *controller reintegration*.

It is necessary to take into consideration that the TTP/C protocol frames have only 20 bits of additional information. For example, they need no address of receiver, because every message is a broadcast; they have, however, CRC check information (16 bits) that makes it possible to distinguish a distorted message.[2]

The TTP/C cluster activity is repeated in cluster cycles (time-triggered principle). Within the cluster cycle some I-frames that contain the cluster status should be broadcasted to enable fast reintegration of a fallen node communication controller. The most simple organization of the cluster cycle uses two TDMA rounds, where every node transmits, within its slot, either an N-frame (i.e. a message containing application data) or an I-frame (i.e. a message containing the cluster communication framework status). So we can assume approximately 20 ms to be a typical cluster cycle duration (and the control application latency time as well).

The FT properties of a TTP/C cluster that executes a control application can be specified at two basic levels:

Fail silence property in the temporal domain. Every message at the TTP/C bus is either correctly delivered, or is not delivered at all (i.e. the failed node is *fail-silent*). All the correct nodes can recognize the state within the next TDMA

[2] The TTP/C protocol implements only physical and link layers from the ISO/OSI model of a general communication protocol. To implement higher layers does not make much sense for the described architecture and its intended utilization.

round. This property is guaranteed by the TTP/C communication framework. It makes possible to implement an active redundancy of nodes, e.g. duplication.

Fail silence property in the value domain. An application can tolerate every single node failure, i.e. it does not deliver a wrong output value into the controlled object when a single node failure occurs. This property must be supported at the application level, which means that the TTP/C cluster structure should contain an active redundancy (e.g. duplicated nodes) and the application SW (or host controller RTOS) should include a redundancy management.

The fail silence property in the temporal domain is roughly implemented as follows:

- The TTP/C controller's Error Detection Mechanism (EDM) should recognize any protocol error and stop its normal activity (i.e. it should move into the *freeze* state). That means that no controller malfunction should propagate outside (if it does we can speak about an *error propagation*). The other nodes can recognize the state (no transmission within a slot) and remove the fallen controller from the membership. The TTP/C *membership service* guarantees consistency of this operation, i.e. all the correct nodes have the same view of membership).
- The Bus Guardian unit should prevent any unintended controller transmitting activity beyond its slot (it could occur e.g. as a consequence of an internal timing error).
- The TTP/C protocol has a *clique avoidance service*. This utility should prevent an unpleasant category of Byzantine (asymmetric) faults. Such a fault can occur when one group of nodes sees a message differently and excludes the rest of the cluster from the membership (the second group does the same). The clique avoidance service guarantees that the lesser group can recognize the state and (voluntarily) joins the "majority opinion".
- The fallen controller tries "to reintegrate", i.e. it listens to the bus traffic and tries to pick up an I-frame that contains the status of communication framework.

TTP/C protocol temporal domain services make it possible to implement a group of several nodes that perform exactly the same activity (members of the group act as *replicas determinate*). Such a group is denoted as Fault Tolerant Unit (FTU); it should be able to tolerate every single member fall, assuming that the failed node is fail-silent. The most simple case of FTU is node duplication. An example of the FTU structure that uses duplication can be seen in Figure 2. Depending on an application, it is possible to use either common or separated sensor(s) for duplicated nodes. Here we demonstrate the second possibility.

Both host controllers of the depicted FTU perform a control loop: reading from sensors, computing a result and emitting a statement to the actuator. This loop has the frequency of a TTP/C cluster cycle. The computing procedure can use

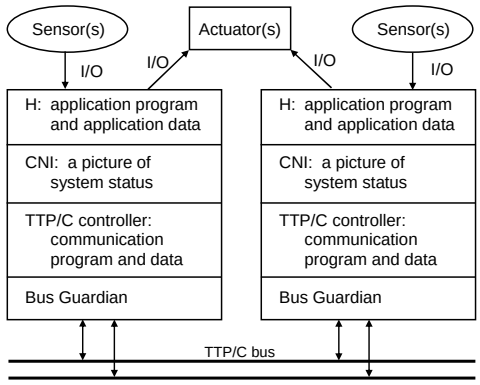


Fig. 2. Fault-tolerant unit that uses duplication

the "state picture" of the system stored in CNI and permanently updated with the cluster cycle frequency[3].

Normally only one node of the FTU group emits a statement to the actuator. When a permanent fault of an active replica occurs, the passive replica can recognize the communication inactivity of its counterpart after one TDMA round (TTP/C membership service) and it can emit a control statement – the same as the active replica should emit because it is its replica determinate.[4]

3 STRUCTURE OF THE SIMULATION MODEL

3.1 Abstract layers of the model

There are three basic abstract layers that the simulation model is composed of:

TTP/C protocol C-reference model. It is a precise (ANSI C-language written) TTP/C protocol specification that describes protocol functionality (i.e. protocol

[3] The TTP/C communication framework implements a kind of "distributed shared memory", where a single part of the memory is a node CNI dual-port memory. The communication activity of a TTP/C cluster is statically programmed using an instance of the MEDL table stored in the local EPROM memory of every TTP/C controller. The content of all the distributed shared memory (CNIs of all nodes) is permanently updated because every node broadcasts data items (which it is supposed to change) with period of the cluster cycle.

[4] This simple reaction is possible in the case when a fail-silent permanent fault semantics of the active replica occurs. For a Byzantine fault semantics, i.e. when the active replica emits misleading actuator commands as a consequence of a transient fault, it is necessary to add time redundancy, e.g. to repeat the application computation and to compare the results before emiting a control statement.

data structures as data types and protocol services as the C language functions). *This layer is TTP/C version dependent.*

Temporal model of TTP/C controller activity. It uses C functions from the previous layer and C-Sim constructions in order to create programs of discrete-time pseudo-parallel processes that implement controller activities.

TTP/C cluster model (i.e. application level). It has to include programs of host controllers, i.e. the application program that is composed of several threads of program code. Thread programs have the form of C functions. A process-oriented model of a host controller operating system is able to do model-time management of threads allocated to a node. Moreover, this layer should include a model of controlled environment that describes (by means of simulation processes) activities of the environment including their interaction with the threads of the application program. *This layer is application dependent*, i.e. it has to be either newly created or at least modified for every instance of the modeled TTP/C cluster.

The current version of the simulation model source code has a clear and well-defined SW modular structure, which is based on two-dimensional layering of program components. This structure is described in greater detail in [4]. Every described abstract layer can be mapped into several program components. Another SW layer (that is not a part of the system model) is the experimental environment. It includes a fault model (and the processes injecting faults as well), a function of recognition and/or registration of the injected fault impact, experiment organization (i.e. the main program function), etc.

3.2 TTP/C Protocol C-Reference Model

The definition of the TTP/C protocol is given in printed form within the specification [19]. This document should serve as a basis for any silicon implementation of the TTP/C protocol, i.e., for a communication chip (the first two chips C1 and C2 have been designed by the TTTech company so far).

In order to obtain a more abstract (silicon implementation independent) description of the TTP/C protocol data and functionality, we first built, in co-operation with TU Vienna and TTTech Vienna, a C-language coded specification, which was denoted as the *TTP/C protocol C-reference model*. This model is not executable by itself, but the data types contain all the necessary information and the defined set of procedures covers as much of the protocol functionality as possible. A C-language based functional specification needs not necessarily to be executable — it is useful as a more precise description of the protocol/controller function than the verbal form. Moreover, it can serve as source data for a software tool for a (semi)automatic chip design from its C-language based functional description.[5]

[5] It leads to the idea "why not to use e.g. SystemC" (instead of C-Sim – see bellow) as an environment where to execute the "C-language based specification". SystemC utilization

The C-reference model describes two main TTP/C protocol data structures:

- CNI – used as data interface between the node communication controller and the node host processor,
- MEDL – used as constant-like information describing the node communication controller activity.

The layout of these data structures is in accordance with [19]; they are, however, treated as abstract data types, i.e. with implementation (data layout) independent interface procedures, so a simple change of data layout has no influence on the description of the C-reference model functionality. The C-reference model includes some more data types, e.g. types of frames, which are transmitted on the bus.

Note: From the methodological point of view, the C-language is used here as a *specification language*. A common requirement for a specification language is to have formalized semantics and to be usable for some of formal proofs (see e.g. [18]) or for some model-checking method (see e.g. [11]). Here we can ignore this requirement because the product of the specification procedure should not serve the purpose of proving formally a property; it should be used for experimental functional validation, i.e. for a kind of execution. The most important reasons leading to the use of the C language were firstly its portability and secondly the fact that most of the embedded systems SW are C language written. Consequently some *parts of application SW can be straightforwardly used as an organic part of the simulation model and tested during the modeled system validation procedure.* A secondary reason was the fact that we had a ready to use C language based simulation tool and also the necessary experience with its utilization.

3.3 C-Sim Based Simulation Model of TTP/C Controller

The simulation model of the TTP/C controller is based on the C-reference model of the TTP/C protocol. The model is based on the principle of process-oriented discrete simulation (i.e., simulation using pseudo-parallel processes) and implemented by means of the C-Sim tool [2]. The principles and software structure of the model are described in [4]. The source code of the C-Sim based simulation model embeds the code of the C-reference model functions into a discrete-time simulation process program that describes the activity of a TTP/C protocol single instance (i.e., the activity of an abstract TTP/C controller). The form of the simulation process enables us:

should e.g. enable to use HW accelerators in order to speed up the simulation. This way is still open but we decided to use C-Sim to have all the computation "deterministically serialized" within one PC station using the model time concept. It generally leads to a better flexibility of the model-construction and model-utilization process. Moreover, the simulation runs quite quickly (see 3.4), so we did not feel any strong need do speed up the computation.

- to add the temporal properties of an instance of the TTP/C protocol computation using a global model-time concept usual with discrete simulation,
- to run several instances (processes) of the TTP/C protocol with their activity "interleaved" in the global model time with regard to the local time flow of protocol instances.

The process-oriented simulation environment makes it possible to add other activity processes (without any influence on the temporal properties of the protocol instances, i.e. without so called *intrusion effect*), e.g. one instance (per node) of bus guardian activity, or some instances of the FI process. The FI process generates disturbances of the "normal" TTP/C activity in order to test its correctness and/or robustness.

3.4 Testing Applications

In order to evaluate the TTP/C protocol based system dependability, two main categories of testing applications were used (i.e. applications executed on the TTP/C cluster undergoing the test – see Section 3.1, the third abstract level of the simulation model):

- A *synthetic application* which is constructed to represent a class of TTP/C cluster utilization cases which enable experimental verification of a given abstract hypothesis and generalization of the test results.
- A *realistic application* which is as close as possible to a chosen safety-critical real-time embedded application. Such an application clearly enables us to test thoroughly some specific property (e.g. a chosen output variable behavior under the influence of transient faults), but a generalization of the test results is more difficult.

So far the following set of applications has been developed and is ready for use for fault tolerance testing purposes:

Dummy application (synthetic) , which only keeps the cluster communication activity "alive" and does no useful work. It only broadcasts data messages containing the node ID. This type of application is quite sufficient when testing only the TTP/C protocol temporal domain properties (see tested hypotheses in part 4.2) and can be configured for any number of TTP/C nodes between 4 and 64. Slots layout for the number of nodes $n = 8$ is shown in Figure 3.

Sine-wave application (synthetic). This application uses the TTP/C cluster composed of four nodes, all of them performing the same activity: repeated reading of an external (sine-wave) signal value and making instances of their own "image" of the read value. Three nodes form a TMR-type (Triple Modular Redundancy) FTU that uses SW voting to emit one common output value. The fourth node serves as a reference (golden) node. This application is suitable for

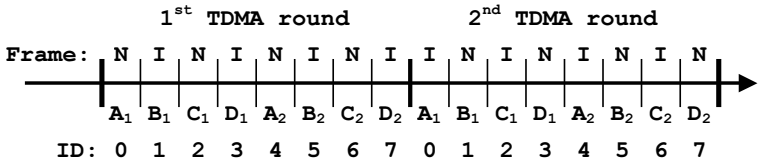


Fig. 3. Slots layout for Dummy application

testing all the hypotheses stated below (see Subsection 4.2). With some caution, the test results can be generalized for a class of stateless applications (i.e., applications without H-state – see [12]) which uses a similar organization of the TTP/C cluster and performs a similar endless read-compute-write cycle. The visualised Sine wave testing application screenshot is given in Figure 4.

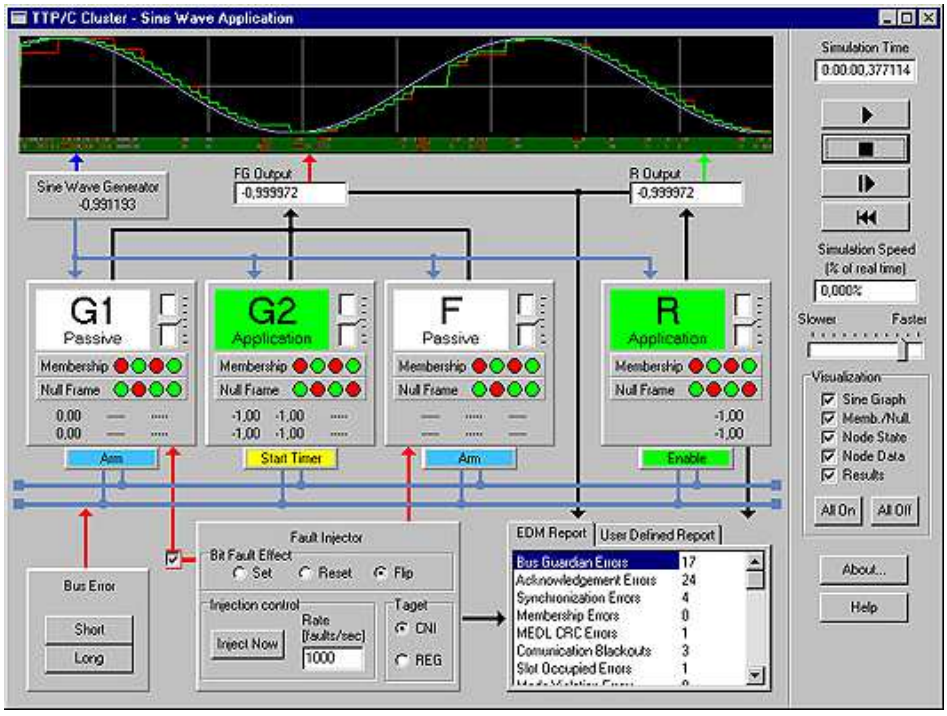


Fig. 4. Sine wave screenshot

Single-wheel brake-by-wire (BBW) application. This application belonging to a "semi-realistic" category was designed by the Volvo company and accepted by the FIT project partners as a standard TTP/C cluster testing workload. The BBW source code was delivered in the form of C-language modules, so it can be either compiled and loaded into real HW or incorporated into a C-language

based simulation framework (as we have done). A run of the BBW application emulates a car braking ABS control system for one wheel.

Four-wheels brake-by-wire (BBW-4W) , also designed by Volvo, uses up to ten TTP/C nodes. This application belongs to the "true realistic" category in the sense that after thorough testing using the TTP/C evaluation cluster and/or the simulation model, the application program code can be used with minor changes to a real car control system. The BBW4 testing application structure uses 10 nodes according to Figure 5.

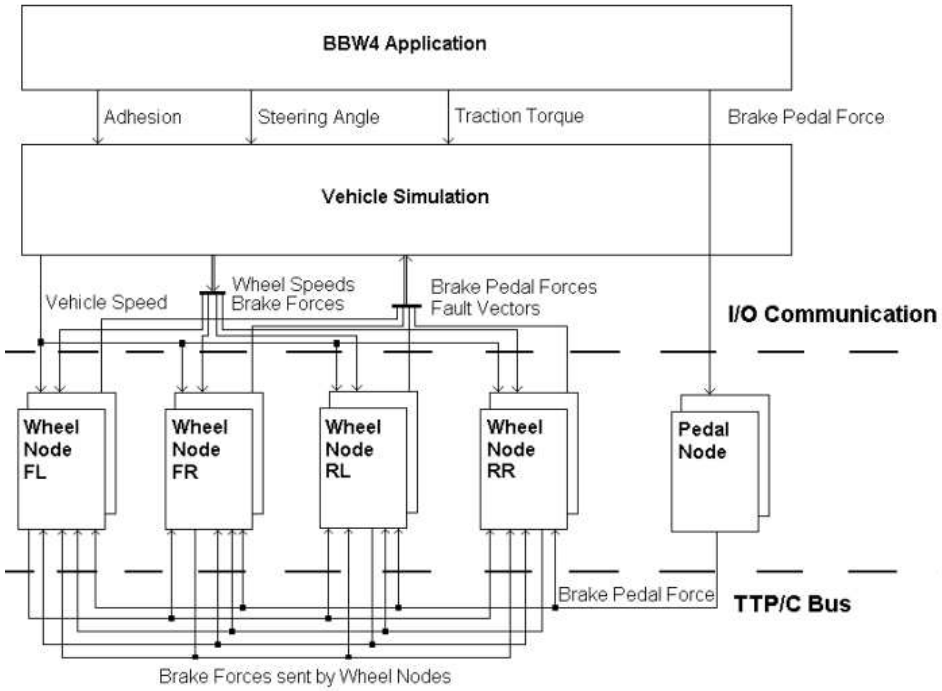


Fig. 5. BBW4 testing application structure

Note: The speed of simulation is, surprisingly, quite good. It is probably influenced by several factors. Firstly the C language translation process is generally very effective. Secondly a good PC station, which is the normal[6] tool for simulation experiments runs much faster then processors intended to be used within the TTP/C node structure. Thirdly the C-Sim library deterministically switched

[6] We have used a supercomputer as well. An instance of the simulation model that does not use any visualization component is ANSI C portable. Moreover, FI experiments can be straightforwardly parallelized, using e.g. farmer-workers model with PVM. Every worker can perform an independent simulation experiment from a set to be performed.

threads (i.e. simulation processes) have a negligible overhead. The resultant effect is that a synthetic testing application, such as a sine-wave runs (in the model time) approximately two times faster than is the (modeled) real time of the life of the application. Realistic application, such as BBW-4W, runs several times more slowly than its "real life".

4 FAULT INJECTION EXPERIMENT ORGANIZATION

4.1 Fault Model

Due to lack of space, all the possible kinds of faults of the assumed distributed system structure cannot be analyzed in this paper. Basically there are *permanent* and *transient faults*. Surprisingly, a permanent single node fault is not so dangerous as the transient fault. The TTP/C controller construction guarantees that the fallen node is fail-silent. This property was tested and verified in many ways within the FIT project. The node-replica can recognize an active node fault (after one TDMA round) and immediately emit a (correct) control statement. Moreover, for a mission oriented application (control systems of planes, cars, etc.) a proper initial testing can reveal most of the permanent faults. The second basic kind of faults is the transient fault. Physical cause of a transient fault can be electromagnetic (EMI) or heavy-ion radiation. The internal consequence of such an impact is normally a (partial) loss of data items stored in an erasable (RAM) memory.

Sometimes the fault has no effect (*ineffective fault*), e.g. a damaged data item is correctly updated before its utilization. An *effective fault* causes an *error*. Such an error should be detected by an error detection mechanism and in the case of an FT system, a redundancy management action should follow. When it is not the case (e.g. a node error within the TTP/C cluster was not detected and propagated into other nodes) a *failure* of the control system specified service can follow with a (possibly) catastrophic consequence.

Note 1: We did not investigate the mapping between an external transient fault and its internal consequence. In the following text we will denote as a transient fault the consequence, i.e. an unintended data change.

Note 2: In the terminology of the TTP/C protocol specification a *single fault* means, expressed in a simplified fashion, either a single node permanent fault or a transient fault that lasts no longer than the TTP/C bus time slot (the shortest one), e.g. occurs in the steady state[7] of the node activity (i.e. it does not disturb the process of node reintegration).

[7] It is the "application" state of the TTP/C protocol state machine, see specification in [19].

Within the FIT project, a transient fault model was accepted. It is also possible to inject permanent faults due to the flexibility of the used simulation method. As the simulation model was built from the TTP/C protocol specification, we injected transient faults attacking only the data structures which are exactly defined in it (e.g. CNI, RAM copy of MEDL). A distortion of the transmitted frames can be injected either into the Message area of a given CNI or directly into the data structure of the transmitted frame.

We introduced a hierarchical model of information-damaging transient fault types (for definitions see also [13]), where a high-level type can degenerate (for specific values of the parameters) into a lower-level type:

Single-bit fault is *a random setting of a bit within a RAM located bit array.* We recognize three types of single-bit faults: setting to logical 0, setting to logical 1, and bit flip (setting to the opposite value). The attributes of a single-bit fault are: memory array identification (process, array, bit offset), fault mask indicating which bits should not be affected, discrete probability function of fault type (e.g. $\{1/3, 1/3, 1/3\}$ meaning that every type of a single-bit fault can occur with the same probability).

m-bit fault is *a set of m single-bit faults within a given bit array* which is located inside a single error containment region – see [12]. All single-bit faults are injected at the same time. Every bit of the array has the same probability to be (successively) chosen as the subject of fault injection. The value of m should be less than or equal to the length of the given bit array; for $m = 1$ we get a single-bit fault. The attributes of the m-bit fault are: single bit fault attributes and the number m of bits, which are influenced by the fault, denoted as *dimension of m-bit fault.*

Burst of m-bit faults is *a stochastic stream of k m-bit faults*, where the time between single events (faults) within the stream has a given probability density function. The attributes of a burst are: simultaneous fault attributes, mean frequency of faults within the stream (λ_c), and number k of faults in the burst, denoted as *length of the burst of faults.* For $k = 1$ we get one m-bit fault, etc.

Stream of bursts is *a stochastic stream of events*, where one event means one burst of faults. The number of events within a stream is not limited. The attributes of a stream are: burst attributes, mean frequency of events within the stream (λ_s), and minimal gap g between two successive events within the stream. For $k = 1$, $m = 1$ we get a stream of single-bit faults.

4.2 Tested Hypotheses

For a better transparency of the simulation-based testing process, we designed a hierarchical set of hypotheses. Every "high level" hypothesis assumes that the "lower level" hypotheses are valid. It is an advantage of the presented method (compared to other FI methods used in the FIT project) that all the stated hypotheses and test

of transient fault robustness can be effectively evaluated. The used set of hypotheses is as follows.

4.2.1 TTP/C Single-Fault Hypothesis in the Time Domain

The TTP/C protocol has been designed to tolerate any single permanent physical fault in any one of its constituent parts, such as the TTP/C cluster nodes denoted as SRUs in [19]. *Tolerating a single fault means that a failed node is "fail silent", i.e., that the failure does not propagate outside of the node and, moreover, all other nodes within the TTP/C cluster are able to recognize the failed node not later than after one TDMA round.* As for the transient fault influencing one SRU, specification declares that for its duration shorter than one SRU slot, the TTP/C based system should tolerate such a fault, assuming that the next fault does not occur before the attacked SRU reintegrates.

4.2.2 Single-Fault Hypothesis in the Value Domain

When the TTP/C time domain hypothesis is valid, it is possible to design TTP/C systems which fulfill the so-called *value domain single-fault tolerance hypothesis*, which means that *the system never delivers a wrong value at any of its outputs when a single-fault occurs.* The basic way to achieve this property is to use node-level redundancy, i.e., to construct groups of nodes representing Fault Tolerant Units (FTUs).

4.2.3 Transient-Fault Robustness in the Time Domain

To test the sensitivity of the TTP/C protocol for a kind of *arbitrary transient fault* (and to reveal as many inconsistencies in the TTP/C specification as possible), we extended the tests "beyond specification". For this purpose the so-called transient fault robustness hypothesis in the time domain was formulated in the following way: *No sequence of transient faults which disturb the controler level volatile (register or RAM located) information can result in a permanent cluster fault, i.e., the attacked node always reintegrates once the sequence of faults is stopped and no other node is influenced (no fault propagation occurs).*

4.2.4 Transient-Fault Robustness in the Value Domain

The TTP/C specification states that: *The resilience of a TTP/C cluster with respect to multiple failures depends on the specific configuration of the application. Many multiple internal faults will be tolerated under normal operating conditions.*

In order to be able to test the stated property, we formulated the following hypothesis: *No sequence of transient faults which disturb the TTP/C level cluster-wide volatile (i.e., register or RAM located) information can result in a permanent cluster fault. Thus the cluster executing an application always recovers its correct*

output(s) once the sequence of faults is stopped. This hypothesis should be made valid (by a proper cluster design) with a sufficient probability, especially for stateless applications[8].

4.3 FI Experiment Methodology

The described simulation method of TTP/C based system fault tolerance evaluation is flexible enough to enable a broad spectrum of FI experiments organization. Using the testing applications characterized above (which are usually capable of non-stop activity) and the transient fault types described above, we have adopted the following FI experiment scenario:

- One experiment is a run of the simulation program for a set of chosen values of the parameters.
- The experiment simulates a given duration of TTP/C cluster activity (e.g. 1 million TDMA rounds).
- Cluster activity is disturbed by a chosen stream of transient faults, and statistics on the influence of the faults are collected. The stream of faults can be either synchronized with the modeled cluster activity (when testing hypothesis 1 or 2) or quite asynchronous (hypothesis 3, 4).
- A violation of the tested hypothesis can bring the simulation experiment run to an end (depending on the hypothesis).

The execution of the program that simulates the TTP/C system activity is deterministically serialized (it uses pseudo-random number generators which are always started from the same value). That is why it is possible to analyze what caused the violation of the tested hypothesis during the given simulation experiment.

4.4 FI Tools

Due to the nature of the experimental testing it is necessary to perform many simulation experiments to reach a sufficient reliability of results of the type "*the tested hypothesis is valid*". Obviously, a single experiment with negative results suffices to confirm the inverse statement: "*the tested hypothesis is not valid*". To be able to repeat many simulation experiments, we designed and implemented a special SW tool.

The Fault Injection Module (FIM) is a flexible tool for FI in the C-Sim simulation environment. It offers a wide range of options for preparing the simulation-based fault injection experiments, run them and analyze their results. The main characteristics of the tool are as follows:

[8] The presented method could be the right tool to estimate such a probability value (i.e. the reliability parameter related to a certain kind of faults). Clearly, we would have to have a proper instance of the fault model, i.e. we would have to know (at least roughly) the internal consequence (data distortion) of an external impact, such as e.g. "lightning".

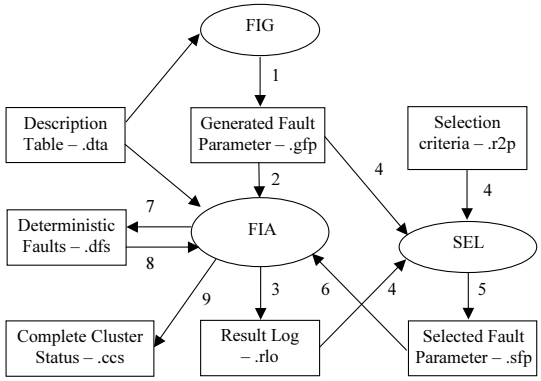


Fig. 6. FIM – structure of modules and sequences of utilization

- It enables a huge set (thousands) of experiments to be prepared (described), each with one or several streams of faults. The experiment is one run of the simulation program using one set of parameters.
- It controls the run of the whole set of experiments. It uses pseudo-random fault injection disturbing the RAM-located information according to the description of the set.
- It enables experiments with "interesting results" (i.e., those leading to an arbitrary error) to be selected automatically.
- It runs these selected experiments for a second time. During this run it also continually writes each fault record (number of the affected node, time, type of faults and fault location).

With this tool the entire RAM-located information can be induced very effectively and quickly, and the results can be analyzed. Results revealing some problem can be used as a basis for repeated experiment runs. Figure 6 presents the structure of modules including generated file types (e.g. .dta, .gfp, ...) and sequences of utilization (e.g. 1, 2, ...).

5 FAULT INJECTION EXPERIMENTAL RESULTS

A large set of FI experiments has been performed within the FIT project, testing **all the hypotheses** stated above in 4.2. As the limited space of this paper does not allow us to report on all of them, we will try to give at least a comprehensive overview. Moreover, all the tests have not been finished yet, because a development of new testing applications is still in progress.

We started with tests aimed to reveal the simulation model bugs and to validate the simulation model. This process was repeated many times during the C-Sim based simulation model building, so the C-Sim based simulation model of the TTP/C

cluster activity was tested thoroughly. It is necessary to emphasize that from the methodological point of view every simulation experiment means a test both of the given hypothesis and also of the testing tool. Due to the deterministic nature of the model itself it is possible to analyze whether an unexpected result was caused by a program bug or by an out-of-specification modeled system behavior. Several inconsistencies in the protocol specifications were discovered and subsequently corrected during construction of the model.

To evaluate the model validity, the most important set of comparable FI experiments was executed in parallel in Vienna and in Pilsen, during which two different forms of SWIFI (software implemented fault injection) were used. TU Vienna used a real TTP/C system as the object of FI, whereas at UWB Pilsen the faults were injected into its model coded in C. The organization of the FI experiments was the same, with the exception of some slight differences which were due to different implementation. All of the 526 bits of the CNI control area were subject to bit-flip injection. The results of the comparison have shown a conformance of 98.9 %, while the cause of the unmatched results can be easily explained by certain differences in experiment organization [1].

5.1 Systematic CNI Bits Injection

For this group of experiments we used the injected node (F-node) CNI data area as the target of FI. The goal of the experiments was to validate the first and second hypotheses given above in 4.2.1 and 4.2.2. The choice of CNI has at least three good reasons:

- The CNI structure is exactly defined by the TTP/C specification, so the model built directly according to this specification makes the FI into specified bits easily.
- We can assume that the majority of internal errors (caused either by the TTP/C controller or by an application) will propagate as a corrupted CNI item.
- Application data exchanged among the TTP/C cluster nodes are stored in the F-node CNI Message area, so the coordination at application level can be disturbed as well.

All bits of the control area of CNI were successively tested, using mainly the sine-wave application. *Systematic injection was synchronous in the sense that the fault injection process was synchronized with the TTP/C cluster activity.* Thus the injected fault is applied within a chosen time point (slot) of the TDMA round, always only within the "application" state of the TTP/C protocol state machine activity. It means that the assumption of single-fault is valid, i.e. the fault influences only one SRU and the next fault cannot occur before F-node reintegration.

5.1.1 Single-Fault Hypothesis in the Time Domain

This group of FI experiments was aimed to test the first given hypothesis (fail-silence in the time domain). The following reactions were observed:

a) Fault propagation from the injected F-node: *no propagation, i.e. no violation of the TTP/C single fault hypothesis in the temporal domain was observed.*

b) Malfunction of the affected F-node: *no malfunction was observed, the node always recovered* (returned into the "application" state).

It means that no violation of the TTP/C single-fault hypothesis in the time domain was observed within this group of extensive tests (for more details see [4]), which is not surprising, because the correctness of the TTP/C protocol during its steady-state processing was proven formally, e.g. [15, 16].

The following table can serve as an example of experimental results. All the faults were injected into the Active state, i.e. the state where the controller is fully synchronized with the TTP/C cluster activity.

File of parameters	rts2u1g10.gfp		State of the injected node	# cases	%
Processing time	7:25:27		Active	10 020 826	100
Cluster time	29:13:12		Init	0	0
Total faults	10 020 826		Listen	0	0
Bus guardian	enabled		Cold start	0	0
Result	# experiments	%	Ready	0	0
No error	526	100	Passive	0	0
Fault propagation	0	0	Freeze	0	0

Table 1. Results of FI into the Active state of TTP/C controler

5.1.2 Single-Fault Hypothesis in the Value Domain

When the first tested hypothesis is valid (or at least not denied), a fault-tolerant TTP/C application can be designed using node replication. Such an application, if properly constructed, can fulfill the single-fault hypothesis in the value domain. At first we used the sine-wave testing application, where three nodes (including the injected F-node) form a TMR-based FTU[9]. We repeated all the tests (as within the previous group) comparing the FTU and R-node (reference node) output. No

[9] The Sine-wave application is not the best one for the given purpose, because the used TTP/C cluster structure is not very representative. The sparse time base concept that is underlying the TTP/C protocol implementation makes it possible to reach the same effect as HW triplication using HW duplication. It is necessary to use an additional mechanisms, such as repeating the computation and providing the transmitted data with

mismatches were observed during these tests, which means that *the single-fault hypothesis in the value domain was not violated*. As the test results are application dependent, only the following conclusions can be drawn:

- The TTP/C cluster is a framework, which makes it possible to construct an application that tolerates a single fault (i.e. does not give a wrong output value).
- The sine-wave testing application is constructed properly in the sense that it does not violate the single-fault value-domain hypothesis.

Moreover, we tested the value-domain hypothesis using both BBW applications. These tests are still in progress, see [7].

5.2 Randomly Induced CNI Bits Injection

Random CNI bits injection can be described as a stationary stochastic process, which at randomly chosen time points randomly chooses a CNI bit (or several bits for multiple bit-flips) whose value is to be injected. *This means that the injection process is asynchronous with the process of TTP/C cluster activity.* When a chosen rate of events (the mean frequency of single injections) within the injection process is dense enough (say, approximately the same as the basic frequency of repeating the TDMA rounds), a fault can be injected before the simulated controller has recovered from a previous fault, so faults are generally incoming in every state of the TTP/C protocol state machine activity. Thus the ability of the controller to integrate itself into the cluster activity (when faults are disturbing the process of integration) has also been verified.

5.2.1 Transient-Fault Robustness in the Time Domain

We performed a large set of experiments with randomly induced CNI bit injections using the FIM tool. These tests were mostly aimed to test the ability of TTP/C abstract controller reintegration even when the process of reintegration is disturbed with a successive fault. The most important result is that *the transient-fault robustness hypothesis as stated in 4.2.3 does not generally apply.* It is not surprising, because we are "out of TTP/C specification", especially out of the single-fault assumption. In a case of a "heavy" injection with a dense stochastic stream of CNI bit injections we observed some percentage of fault propagations and cluster errors.

An example of this is one set of experiments carried out using the sine-wave testing application. Each of the 526 CNI Control area bits was injected with several fault types (0, 1, bit-flip) and the used bit-injection stream parameters were changed. The total number of experiments executed was 6192, the length of each was 200000

redundant information (e.g. end-to-end checksum), which makes it possible to detect data distortion (that occurs after transmission; during transmission the CRC serves this purpose). Development of a better (i.e. more representative) testing application for the given purpose is in progress.

TDMA rounds (one round has 4 slots per 2.5 ms). Of the 6192 experiments only 112 experiments led to fault propagation (1.8 %).

Results of the experiment, such as those given above, can be utilized in two ways. Firstly the robustness property related to a certain fault type can be evaluated by means of a probability value. Thus, e.g. the estimated probability that the modeled cluster will not "survive" a transient fault of the type "dense stream of CNI bit injections, of the duration of 1 second" is[10] about 1.4×10^{-6}.

Secondly results of the experiments can be used to analyze the reasons which led to a cluster fault. Such analysis may help improve the TTP/C protocol state machine (the abstract TPP/C controller) behavior in order to prevent fault propagation.

5.2.2 Transient-Fault Robustness in the Value Domain

This property (as defined in 4.2.4) relies on the previous one (4.2.3). It is not surprising that both properties can be achieved only with a certain probability (e.g. reliability value), because they are "out of TTP/C specification". We tested the transient-fault robustness property using the sine-wave application whose all four nodes were "attacked" with a dense stream of CNI bit-flip bursts; every bit of the CNI was endangered with the same probability. The tests done so far were described in [5]. The preliminary results show that e.g. for the length of burst 16 TDMA rounds, the mean period of bursts 128 TDMA rounds and "dimension" of burst varying from 1 to 1000 bit-flips, the probability is about 10^{-3} that the modeled cluster output does not recover, i.e. that the burst of transient faults is transformed into a permanent fault. This value is clearly application dependent (and fault-type dependent as well), which means that a model-based test of this type should be done in the final stage of application design.

6 CONCLUSIONS

The method presented in this paper is based on a generic C-language written simulation model of an embedded distributed computer system used for a safety-critical control application in order to evaluate its fault-tolerant properties. The method enables us to inject various kinds of faults and evaluate their influence, i.e. to test different hypotheses concerning the fault tolerance of the system. The simulation model can be executed on an ordinary PC station and the simulation model-time speed is comparable with the modeled system real-time speed.

The results obtained so far are based on the tests of the TTP/C protocol specified properties (the corresponding hypotheses were confirmed). The tests using the "abstract controller" (SW based, built according to specification) also significantly contributed to an improvement of TTP/C protocol specification quality. The simulation model validity was tested in cooperation with TU Vienna by comparison with

[10] When taking into consideration the modeled 350 hours of the cluster function time.

the SW implemented FI method (SWIFI) applied to real HW. This comparison revealed a very high degree of coincidence of the results.

Further results were obtained by testing the TTP/C protocol robustness (i.e. tests beyond the TTP/C specification) verifying the resilience of a TTP/C cluster with respect to multiple faults. We found that many multiple internal faults will be tolerated under normal operating conditions, and a TTP/C based system can successfully survive even in very severe conditions.

A generalized experience gained in the FIT project identifies three main application areas of the C-language based simulation model:

Evaluation of protocol specification correctness (and its possible modifications). Thus we can test a TTP/C cluster communication framework using instances of an abstract TTP/C controller (C-language coded functional model of a real controller) and a synthetic testing application.

Evaluation of protocol specification robustness. In this way we can test a TTP/C cluster communication framework using arbitrary kind of fault (i.e. not only single fault). Using the simulation method, it is possible to estimate the reliability parameters related to a given kind of arbitrary fault.

Evaluation of a TTP/C based real-world application, using its C-language source codes. As the C-Sim based model enables C-language coded application SW modules to be incorporated, it is generally possible to use the C-Sim based model and PC station executed evaluation system instead of a real TTP/C cluster. A given fault hypothesis can be tested by using a model-implemented FI either into the CNI Message Area structures or into a specific application data item. It is possible to use a broad spectrum of fault models and/or a broad spectrum of tested hypotheses or system properties.

List of abbreviations

BBW Single-wheel brake-by-wire application

BBW-4W Four-wheels brake-by-wire application

BG Bus Guardian

CAN Control Area Network

CNI Computer Network Interface

C-Sim software tool

CSMA Carrier Sense Multiple Access

EDM Error Detection Mechanism

EMI electromagnetic radiation

FI Fault Injection

FIM Fault Injection Module

FIT Fault Injection for Time Triggered Architecture

FT Fault Tolerance

FTU Fault Tolerant Unit

MEDL Message Descriptor List

PVM Parallel Virtual Machine

RTOS Real Time Operating System

SAE Society of Automotive Engineers

SRU smallest replaceable unit

SWIFI Software Implemented Fault Injection

TDMA Time Division Multiple Access

TMR Triple Modular Redundancy

TTA Time Triggered Architecture

TTP/C Time Triggered Protocol

TTPOS real-time operation system for TTP/C

TTTech Austrian software firm

REFERENCES

[1] Ademaj, A.—Grillinger, P.—Herout, P.—Hlavicka, J.: Fault Tolerance Evaluation Using Two SWIFI Methods. In: Proceedings of IEEE IOLTW 2002, Isle of Bendor (France), July 2002, pp. 21–25.

[2] `http://www.c-sim.zcu.cz`

[3] `http://www.cordis.lu/esprit`

[4] `http://www.fit.zcu.cz`

[5] Grillinger, P.—Racek, S.: Transient Faults Robustness Evaluation of Safety Critical Systems Using Simulation. In: Proceedings of BEC 2002 (Baltic Electronic Conference), Tallinn, Oct. 2002, pp. 257–260.

[6] Heiner, G.—Thurner, T.: Time-Triggered Architecture for Safety-Related Distributed Real-Time Systems in Transportation Systems. In: Proceedings of FTCS-28, Munich, Germany (1998), pp. 402–407.

[7] Herout, P.—Grillinger, P.: Simulation Tool for Functional Verification of TTP/Cbased Systems. In: ESS2003 – 15th European Simulation Symposium and Exhibition, Delft, The Netherlands, Oct. 2003 (submited).

[8] Hlavicka, J.—Racek, S.—Smrha, P.: Functional Validation of Fault-Tolerant Asynchronous Algorithms. In: Proceedings of Euromicro, Prague, Czech Republic (1996), pp. 143–150.

[9] Hlavicka, J.—Racek, S.—Herout, P.: Analysis and Testing of Process Controller Dependability. In: Proceedings of Ninth IEEE European Workshop on Dependable Computing, Gdansk, Poland (1998), pp. 7–11.

[10] Hlavicka, J.—Racek, S.—Herout, P.: Evaluation of Process Controller Fault Tolerance Using Simulation. Simulation Practice and Theory, Vol. 7, Nr. 8, March 2000, pp. 769–790.

[11] Holtzmann, G. J.: The Model Checker SPIN. IEEE Transaction on Software Engineering, Vol. 23, No. 5, May 1997.

[12] Kopetz, H.: Real-Time Systems, Design Principles for Distributed Embedded Applications. Kluwer Academic Publishers, 1997, p. 338.

[13] Laprie, J. C. (ed.): Dependability: Basic Concepts and Terminology. Springer-Verlag Wien, New York, 1992, p. 265.

[14] Manzone, A. et al.: Fault Tolerant Automotive Systems: An Overview. In: Proceedings of 7th Int'l On-Line Testing Workshop, Taormina, Italy, 9.–11. 7. 2001, pp. 117–121.

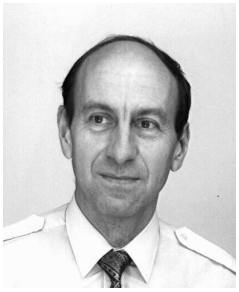
[15] Pfeifer, H.—Schwier, D.—Henke, F. W.: Formal Verification for Time-Triggered Clock Synchronization. Published in Dependable Computing and Fault-Tolerant Systems, Vol. 12, C. B. Weinstock and J. Rushby, eds., pp. 207–226, IEEE Computer Society.

[16] Rushby, J.: Systematic Formal Verification for Fault-Tolerant Time-Triggered Algorithms. IEEE Transactions for SW Engineering, Vol. 25, No. 5, Sept/Oct 1999, pp. 651–661.

[17] Rushby, J.: Bus Architectures For Safety-Critical Embedded Systems. Published in proceedings of EMSOFT 2001: First workshop on Embedded Software, October 2001, Lake Tahoe CA. Springer-Verlag Lecture Notes in Computer Science.

[18] Sveda, M.—Vrba, R.: Executable Specifications for Distributed Embedded Systems. IEEE Computer, Vol. 34, 2001, pp. 138–140.

[19] `http://www.tttech.com`



Stanislav Racek was born in 1946. He graduated at the Faculty of Electrical Engineering, Technical University of Pilsen, 1969. Currently he works at the Department of Computer Science and Engineering, University of West Bohemia in Pilsen. His research interests include reliability and performance modeling of computer systems, fault-tolerant computer systems, modern programming tools and methodologies.

Pavel Herout was born in 1961. He graduated at the Faculty of Electrical Engineering, Technical University of Pilsen, 1985. This time he works at the Department of Computer Science and Engineering, University of West Bohemia in Pilsen. His research interests include fault-tolerant computer systems, modern programming tools and methodologies, simulation methods and tools.

Jan Hlavička (1942–2002) graduated from Faculty of Electrical Engineering, Czech Technical University in 1964. He received his PhD and DSc degrees from the same university in 1971 and 1987, respectively, and was appointed Associate Professor and Professor by the same institution in 1985 and 1991, respectively. During his fruitful scientific career, he was with the Research Institute for Mathematical Machines in Prague, Siemens Munich, and Department of Computer Science and Engineering of the Faculty of Electrical Engineering, CTU Prague. He was the visiting professor at TH Ilmenau (Germany), Universit de Montréal (Canada) and Hochschule fr Bauwesen Cottbus (Germany). He is the author and co-author of numerous scientific papers. His research interests included fault-tolerant computing testing and diagnostics of digital circuits an systems,computer architecture, error coding, self-checking circuits.