SIMULATION TOOL FOR FUNCTIONAL VERIFICATION OF TTP/C–BASED SYSTEMS

Petr Grillinger and Pavel Herout
Department of Computer Science
University of West Bohemia
Univerzitní 22, Plzen 30614, Czech Republic
E-mail: pgrillin@kiv.zcu.cz

KEYWORDS

SW tool for simulation, fault injection, brake by wire, TTP/C protocol, the C language.

ABSTRACT

This article describes a software tool that implements C-language written simulation model of distributed embedded computer system that is interconnected by means of TTP/C protocol. The aim of simulation is to evaluate specified system's properties when used as a safety critical control system. The method that uses simulated faults to disturb system's activity was developed during the solution of the EU/IST project FIT — Fault Injection for Time Triggered Architecture (TTA). A utilization of the described simulation tool is demonstrated when evaluating the time that a TTP/C cluster that is executing a realistic brake-by-wire control application needs to stop the car while the braking process is disturbed by transient faults.

INTRODUCTION

Verification of dependability is one of the most important steps in the design of a fault-tolerant embedded computer system. This includes testing the system's fault tolerance, i.e. its reaction to real-time disturbances from its environment and to faults inside and outside the system. Finding an appropriate verification method may save a considerable amount of time, expenses and manpower, therefore it is paid ever-growing attention. Different experimental verification methods were suggested so far, mostly using some kind of *fault injection* (FI). Fault injection can be performed on a simulation model of the system to be evaluated, on a prototype, or on the system itself. Each of these approaches has its advantages and disadvantages (flexibility and ease of implementation on the one hand, more convincing results on the other hand).

The method described here is based on digital simulation whose output is used both for qualitative and quantitative evaluation of the tested system properties. It uses a close-to-real code (C-language) describing the FT system function together with the computation dynamics and with a sub-model of the controlled environment. Thus the system behavior (even in the presence of faults) can be studied using a conventional PC workstation. The paper presents some of the results of the presented method utilization obtained by two research groups within the EU project IST-1999-10748, Fault Injection for Time Triggered Architecture (FIT), see (http://www.cti.ac.at/fit).

The goal of the FIT project was an evaluation of time-triggered architecture (TTA). This is architecture of ultra-reliable embedded computer systems aimed to control cars, planes, trains, etc. (Kopetz 1997; Heiner and Thurner 1998). Special feature of TTA is fixed partitioning of node time slots on the bus, which guarantees predictable time behavior of nodes that are connected to the bus. The method of access to the bus is then TDMA (*Time Division Multiple Access*) instead of common CSMA (*Carrier Sense Multiple Access,* used e.g. by the CAN bus) and one of possible implementations of the method yields the so-called TTP/C protocol (Kopetz 1997).

TTP/C is a real-time communication protocol for interconnection of electronic modules of distributed fault-tolerant real-time systems. TTP/C is intended to meet the requirements for SAE class C automotive applications (TTP/C means Time Triggered Protocol — C class of SAE requirements). Every node connected to the bus consists of three main parts:

- communication controller, which executes the TTP/C protocol (existing C1 chip produced by TTTech company in Vienna was tested within the FIT project),
- host processor, which executes a part of an application program (and has its own I/O interface to the controlled process),
- dual-port memory, that serves as an interconnection device between the controller and the host (it is called CNI — *Computer Network Interface*).

Nodes connected to the bus form TTP/C cluster. The basic period of the bus communication activity is called a TDMA round. Within this round every node has its own slot assigned to transmit its messages.

The specified properties of TTA architecture have been evaluated experimentally using the method of fault injection (FI). The FIT project includes several kinds of FI, like hardware induced FI (UPV Valencia), software

Proceedings 15th European Simulation Symposium
Alexander Verbraeck, Vlatka Hlupic (Eds.)

ISBN 3-936150-28-1 (book) / 3-936150-29-X (CD)

implemented FI (SWIFI, used by TU Vienna), heavy-ion impacts on the chip (CTH Gothenburg), etc. The mentioned methods use the real HW (TTP/C evaluation cluster produced by TTTech) as an experimental environment. Another approach is to use simulation model of the communication controller and/or the whole TTP/C cluster. Within the project, two levels of simulation modeling (that differ in abstraction level) were used. The lower level is a VHDL model of communication controller itself (used by UPV Valencia, CTI Karnten). It uses abstractions like *gate*, *pin*, etc. and transient faults *like pin_x grounded for one microsecond,* see (http://www.cti.ac.at/fit).

The task of our team was *to build a silicon implementation independent description (C-language based) of the TTP/C protocol* in order to have a precise and flexible description of specified TTP/C data structures and functions. In co-operation with TU Vienna and TTTech, we first built a C-language written TTP/C specification (so-called C-reference model of TTP/C protocol). This model was to simulate the behavior of TTP/C protocol at the level of message transmission and basic process interaction. Low level activity simulation (e.g. gates, pin signals, etc.) was not required — such model was already available in VHDL (due to its complexity, a VHDL description is not suitable as reference model).

However, being just a specification, this model is not executable itself. To verify its correctness and to use it for the FI purpose, we embedded the C-language written protocol specification into the C-Sim simulation environment that allows a Simula-like (coroutine) style of pseudo-parallel computing (http://www.c-sim.zcu.cz; Hlavička et al. 2000). The simulation model of TTP/C was verified by comparison of results with the other FI techniques — namely the SWIFI method — see (Ademaj et al. 2002).

This paper describes only in the most basic form the simulation tool and the TTP/C model itself. It concentrates more on the possible use of this kind of model in high-level verification of dependable systems. As a case study, a real-world application — Brake by wire 4 (BBW4) — is presented. It is important to notice that this application is very specific, but the simulation tool itself, the applied methods and the lower layers of the software (see table 1 in section 3) are general. This paper does not intend to propose a new original approach to simulation; rather it presents a complete utilization of one particular technique.

C-LANGUAGE BASED SIMULATION MODELING

The discrete-time simulation modeling principle enables to run several instances (i.e. processes in simulation terminology) of TTP/C protocol with their activity "interleaved" in the global model-time with regard to local-time flow (microticks/macroticks counters) of protocol instances. The protocol/controller instances execute the same program but they have their own data (i.e. CNI instances). Other processes (e.g. threads of a control program executed by a host processor, threads modeling controlled objects, threads of simulation experimental environment, etc.) can be added without any unintended mutual time-intrusion effects — only the speed of simulation depends on the number (and complexity) of executed processes. The computation is completely deterministic, so the obtained results are fully reproducible.

The method needs no special hardware. Typically a PC station with a C-language development environment (C++ can be used as well) can be used to develop and to perform (visualized) FI experiments. Moreover, massive non-visualized FI experiments can be automatically performed using “batch mode” execution on a powerful mainframe-like supercomputer.

The method is general enough (Hlavička et al. 2000), but we especially present its implementation that was used within EU/IST FIT project solution for an experimental validation of TTP/C based system properties (Herout et al. 2002). The lessons learned within the FIT project solution identify two main application areas of the SW implemented fault injection using C-language based simulation model:

- **TTP/C specification level** (an earliest and most abstract development phase of a class of distributed embedded computer systems architecture): The presented simulation methodology enables an evaluation of communication protocol specified properties, e.g. fail-silence property for a single TTP/C controller permanent fault, measurement of time of controller recovery that follows after transient fault, etc.

- **Application level** (the final TTP/C system development phase): 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 and PC station executed development (evaluation) system instead (or as a counterpart) of a HW based TTP/C evaluation cluster. The FIT project results confirm the usefulness of utilization of the TTP/C cluster simulation model for a realistic FT application design including fault injection and a visualization of faults influence.

This article especially concerns the second item stated above.

SOURCE CODE MODULAR STRUCTURE

In order to achieve a clear and consistent SW structure with a sufficient degree of portability and reusability,

we introduced a two-dimensional layering of SW modules.

The first dimension (i.e., the first way of SW layering) reflects the C-code stability, and consists of three layers (the letters assigned to individual layers are used as first and second letters in file names, so the purpose of a module can be easily identified by looking at its name):

- **L** — stable code (i.e. libraries, protocol independent code),
- **P** — protocol version dependent code,
- **A** — application-dependent code (e.g., empty application, sine wave application, brake-by-wire application etc.).

The second dimension reflects the degree of extension of C-language code properties:

- **F** — pure functionality,
- **S** — simulation, i.e., multithreading ability and time properties,
- **V** — visualization of a TTP/C cluster activity.

The second dimension determines the code portability — **F** and **S** layers are ANSI-C written (and use only ANSI-C standard libraries), so they are generally ANSI-C portable. Modules of the **V** layer are C++ Builder v.5.0 coded and assume the use of 32-bit Windows operating system.

The modular layering scheme is graphically displayed in Table 1.

Every module from the structure depicted below may export its services via an interface to the modules on the right or below. More details can be found in (Hlavička et al. 2001; http://www.cti.ac.at/fit).

The described software tool is not designated for wide non-initiated public as a ready-to-use program. It should serve programmers in the C programming language to prepare their own (realistic) application by adding the C-code of an application. Some parts of the tool can serve as libraries and it is possible to use these parts as components in a visual programming environment. Moreover, there is a possibility to unify the development process for hardware and software-based evaluation clusters.

CASE STUDY: BRAKE BY WIRE APPLICATION

The first brake-by-wire application (BBW1) was proposed in (Lönn 2001) to enable fault injection testing with transient faults, which can attack arbitrary volatile information at the TTP/C level, especially controller's local data and CNI data. An extended version of this application, that is closer to reality because it simulates four wheels instead of just one (BBW4) has been developed later and this later application is presented here.

Both BBW application cores were developed by Volvo Technological Development Corporation in Matlab/Simulink for the FIT project and then converted to ANSI C source files.

The BBW application consists of wheel simulation with ABS controller and vehicle simulation. The current four-wheel BBW simulation cannot run on the HW cluster, because it uses floating point variables and operations. It is possible replace the floating point arithmetic routines by fixed point routines — this means however an additional overhead and the development time of such application would be longer. This shows an advantage of simulation method that has no restrictions in this direction.

The Structure and the Principles of Four-Wheel BBW Simulation

The four-wheel application is very close to reality (the model of the car provides nine degrees of freedom). All wheels are simulated, the external vehicle model

		j → *code properties extension*		
		F	**S**	**V**
↓ *code instability*	**L**	ANSI-C library	C-Sim kernel	C++ Builder library
	P	FPx	SPx	VPx
	A	FAxy	SAxy (main)	VAxy (main) ↓
		ANSI-C portable ←	→ *OS dependent*	*executable*

x-axis — TTP/C version, here assumed 1.0, i.e., x = 1
y-axis — cluster configuration + application

Table 1: Source code modular structure

simulates vehicle reactions and the distribution unit redistributes brake force when a brake failure is detected. Figure 1 schematically explains the interconnection of individual modules.

Individual parts of the simulation:

a) **Vehicle simulation** — the vehicle simulation is based on the behavior of the Volvo V70 car and except when moving with low speeds its behavior is practically the same. The vehicle simulation runs with a period of 500 microseconds. The simulation uses one parameter (initial vehicle speed) at the beginning of the simulation and four input variables in each simulation step. These variables are: *Brakel_Force* vector, *Traction_Torque* vector, *Steering_Angle* vector and *Adhesion* vector. All vectors have four members, one for each wheel. The *Brake_Force* is given in *N* units, *Traction_Torque* in *Nm* units, the *Steer_Angle* in radians and the *Adhesion* is a non-dimensional value from 0.0 (ice) to 1.0 (dry tarmac). These four vectors are used to control the experiment during the simulation. The outputs from vehicle simulation are: *Wheel_Speed* vector and *Vehicle_Speed* vector (in heading direction, *m/s*). Heading angle in radians and global x-axis and y-axis position (*m*). The ABS controller uses the output values to compute the final *Brake_Force*. All output and input values are 8 byte floating-point variables or vectors.

b) **Distribution unit** — the distribution unit is introduced to reduce the effect of brake faults. This unit uses as its input the requested *Brake_Torque* sent by the pedal node and *Fault_Vector*, which is derived from TTP/C model (using membership service provided by the TTP/C protocol complemented at application level with comparison of messages from both replicas). Each bit in this array corresponds to one wheel — a value of 1 signalizes that the brake system at this wheel failed. The distribution unit computes the output *Brake_Force* from the requested *Brake_Torque*, the multiplier factor (set for each wheel in the actual situation), and from the fact whether the wheel is front or rear (the brake force distribution is 65% for front wheels and 35% for rear wheels).

c) **ABS controller** — The ABS controller does not compute the final brake force, it only uses the input brake force obtained from the Distribution unit and then decides whether the *Brake_Force* will be zero or the input value. Its decision depends on the second and the third input values — the *Wheel_Speed* and the *Vehicle_Speed*; the difference between these two values starts or stops the function of the ABS controller. These two modules (distribution and ABS controller) run with period of 4 milliseconds, it means that these simulations have eight times longer period than the vehicle simulation. This is convenient for our TTP/C simulation because the cluster cycle has the same length, which makes scheduling easier.

TTP/C Model of Four-Wheel Brake by Wire

The simulation is run on 10 nodes divided into two categories. At every wheel brake are two replicated controlling nodes (8 nodes altogether), and at the brake pedal different two nodes measure the pedal's position. The vehicle motion, brake force distribution and ABS controller status is updated externally in a separate thread. This structure of the application was obtained

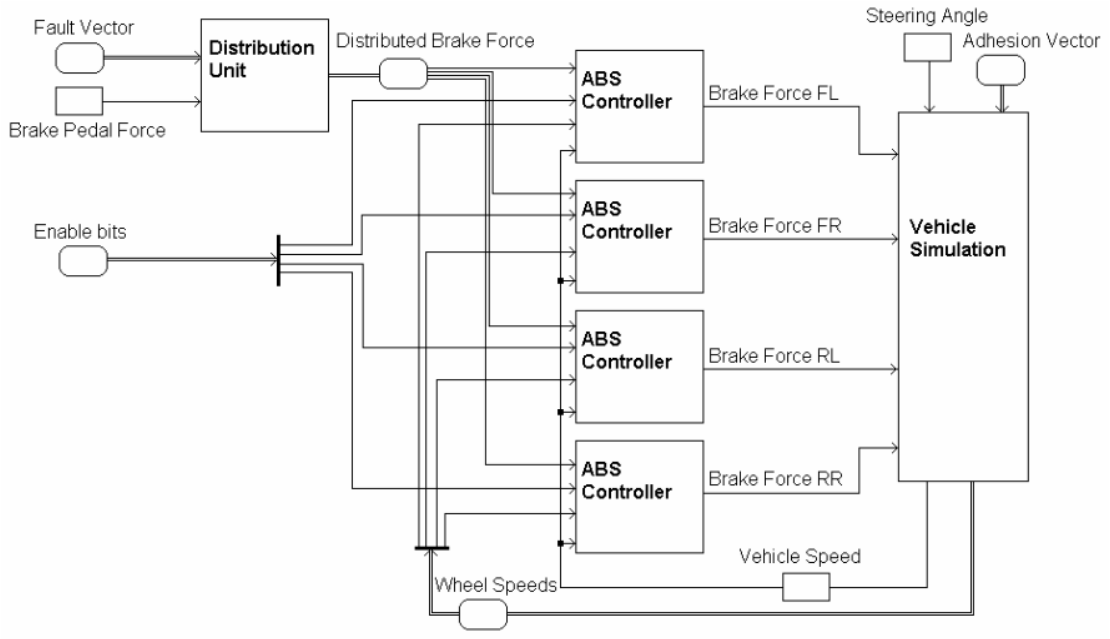

Figure 1: Detailed Structure of Four-Wheel BBW Simulation

from Volvo Company and it is difficult to divide into four (or eight) parts. The distribution unit and the ABS controller routines can run on every wheel node. This division means that the simulation status (internal variables of the model) is updated only at one place in the simulation, but the function that computes the *Brake_Force* for a wheel is distributed among all (eight) wheel nodes.

The application task of a wheel node (executed in real-world directly by the node's host computer) can be divided into four steps:

- The first step is reading of the values of *Brake_Force* sent by other wheel nodes during the last TDMA round and their comparison. The node compares corresponding values sent by wheel nodes of other wheels. This comparison is used to set up the fault vector for wheel node — i.e. its own view of the cluster situation.

- The second step is reading the *Brake_Torque* sent by two pedal nodes. In case that the read values are identical, their value is used for the new *Brake_Force* computing function; otherwise brake force equal to 0 is used.

- The third step is *Brake_Force* computing. In this phase the node uses the functions developed by Volvo. These functions use *Brake_Torque,* their individual image of *Fault_Vector,* their *Wheel_Speed* and *Vehicle_Speed* as an input.

- In the last step the application writes the actual value of *Brake_Force* into the CNI for the next transmission slot.

Cluster and Message Structure

As was mentioned before, the model runs 10 nodes coupled into five pairs. For better understanding of the structure we can look at figure 2.

We divide the cluster cycle into two TDMA rounds. In the first TDMA round all nodes send N-frames (frames with application data content) on the bus; in the second round all nodes send I-frames (these frames contain no application data, they contain current cluster state that can be used for reintegration). Each sending slot is 200 μs long.

Multiple I-frames help the nodes to reintegrate after a breakdown in situation when multiple nodes break down as a consequence of a stream of faults. We could send I-frames by pedal nodes or by wheel nodes only. However, it is quite dangerous to send I-frames only by the pedal nodes, because both pedal nodes could fall out and then no node could reintegrate into the cluster. The second option is irrelevant due to a different reason: the danger that there will be no running node that will send I-frames is very low, but we need much more information about how the brakes really work, than the information about the brake pedal status. In a real

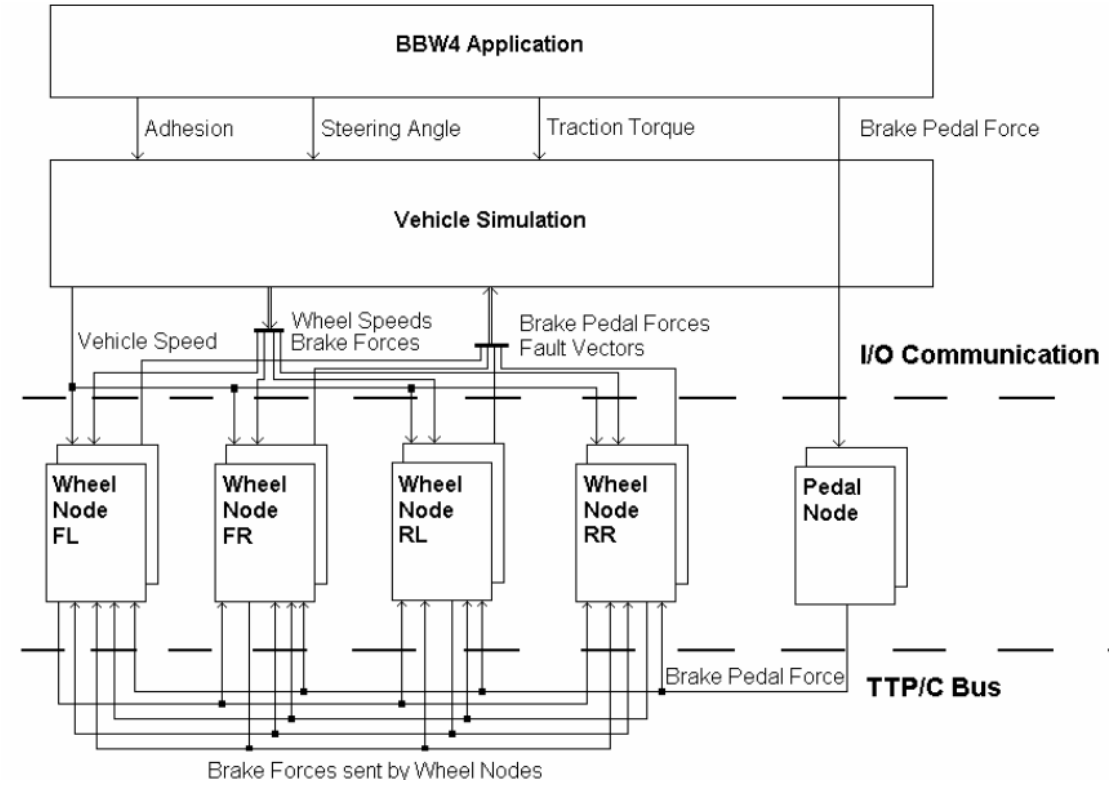

Figure 2: BBW4 TTP/C Cluster Structure and Data Flow

application the brake pedal value change will be slow if we know that the ABS controller period is only 4 milliseconds long, so there is no reason to refrain from sending of I-frames by the pedals nodes.

Fault Detection Mechanisms

Fault detection is the main task of safety critical applications when faults are introduced into the system. The application has to be fault-tolerant or at least be able to minimize the influence of the fault. Our application is the second case, because when we use only two nodes for each wheel we cannot develop a fault-tolerant application for each wheel, but we can detect a fault by comparing outputs from wheel nodes, and correct the fault consequences by changing the application parameters using the distribution unit.

The fault detection mechanism in our application is used twice.

- An application task uses the data sent on TTP/C bus. The *Fault_Vector* is used in the distribution unit to minimize the influence of a fault.

- In the vehicle simulation thread that simulates actuators on each wheel that receive the value of *Brake_Force* from nodes on the wheel sent by I/O communication from the nodes. This unit needs to have two identical values on its input; otherwise it cannot decide which value to use and performs no action.

Fault Detection in Wheel Node Task.

Three fault detection routines are executed during the wheel node task. The first serves to detect a fault in the actual node or its replica. The wheel node reads messages sent by both nodes from CNI Message Area and compares them. The result of the comparison is *equal* or *not equal*. When the messages are different, the wheel node increases total fault counter together with continuous fault counter. Total fault counter is information about the wheel stability only, but the continuous fault counter is used to limit future failures — when the continuous fault counter reaches 5 the node automatically restarts itself (then a self-test can be performed). These counters can be also incremented in one other case — when the wheel node cannot read valid data from its message stored in CNI.

The second test is used to create node's own overview of the whole cluster situation. The node tries to detect wheels that do not work properly, i.e. both nodes are broken down or the nodes send different data or send invalid data. The obtained information is used for *Fault_Vector* set-up — each node has its own copy of this vector.

The third test is used for *Brake_Torque* (*BT*) signal validation. The wheel node compares the values sent by Pedal nodes and sets a new *BT* value if they are equal or sets the *BT* value to 0 if they are different.

Types of Tests

The tests have to show that it would be possible to stop the vehicle in spite of faults injected into TTP/C nodes or the bus and in the best case the car would not slip from the straight direction. BBW4 minimizes the influence of faults by a brake force distribution mechanism. The use of this mechanism is enabled by the TTP/C cluster services — membership service, node replication, channel (and frame) replication and time synchronization.

A fault injection experiment can be organized in several ways:

- *White-box FI* — the faults are targeted into exact locations and/or injected at exact points in time. Such experiments are used often to verify a particular hypothesis (e.g. clock synchronization).

- *Black-box FI* — the faults are injected at random intervals and hit random targets in the modeled system. This experiment organization is suitable to prove general resistance to FI. A fault model that is close to reality is essential.

In our case a variant of black-box FI has been applied primarily — pseudo-random stochastic FI.

Stochastic Fault Injection.

This method utilizes streams of faults (mostly Gaussian or Poisson streams) that are controlled by a pseudo-random number generator. Our basic fault model is a short *burst of single-bit flips* (sequence of several bit-sized faults) that is repeated within a stream with mean period larger than the duration of a single burst. This fault model simulates reoccurring transient problems of the chosen node (e.g. EMI effects) or transient malfunction of node's sensors, depending on the target of FI.

Selected results

A simple fault injection experiment was chosen for demonstration. It was stated that the brake force distribution mechanism should be able to minimize unwanted effects of a brake failure. To work properly the mechanism requires an accurate and up-to-date fault vector. Braking force to be applied to individual wheels is determined for the current communication period using fault vector from previous period. *This means that when the fault vector changes frequently the force distribution mechanism's performance can degrade significantly and the braking distance may be longer than when a permanent brake failure occurs.* This theory can be easily verified by the model when we run experiments with different frequencies of faults. Very high frequency of faults can be used to simulate a

permanent failure (the period must be shorter than the fastest node recovery time, which is 1 TDMA round).

Table 2 bellow summarizes the output received from braking trajectory measurement for different fault injection frequencies (the fault is simultaneous shutdown of both replicas at the rear left wheel). Observations of the car behavior in the presence of faults have revealed that the distribution mechanism over-compensates the failed brake in time, so the vehicle starts drifting in the opposite direction (then the effect is reversed, so the car moves from left to right and back). This makes measurement of current Y-Axis position almost useless, so the table below lists only the maximum deviation in the Y-axis (max. lateral movement).

The intensity of fault injection is given as a ratio of fault-injected TDMA rounds to total number of TDMA rounds. For example "1:10" ratio means 1 fault in 10 TDMA rounds. The special value "0:–" means that no FI was performed and the ratio 1:1 means that a fault is injected in every TDMA round (this is a simple model of permanent failure).

FI Ratio [faulty : total]	Traveled distance [m]	Max. lateral movement [m]
0 : –	56.67	0.00
1 : 10	60.53	0.41
1 : 5	75.57	0.38
1 : 3	79.65	0.38
1 : 2	128.64	2.45
1 : 1	89.73	1.29

Table 2: Braking results for different FI frequencies

The first row in table 2 contains measurement of the braking process under optimal conditions, i.e. no fault injection was applied. This provides us with reference values, so we can compare the values retrieved with different fault injection settings. The lower rows in the table gradually increase fault intensity and we can see that the measured travelled distance and lateral movement increase accordingly — except for the FI ratio "1:2". In this case the overall braking distance and maximum lateral deviation is significantly larger than for other FI ratios (even for higher fault frequencies). This behaviour is caused by the brake-force distribution mechanism that shows a possibly dangerous instability to certain fault frequencies (the algorithm assumes that current state will be valid for the next communication period and this FI frequency invalidates the assumption). This confirms the previously stated hypothesis (in italics at the beginning of this section) and gives us a rough worst-case braking scenario.

A snapshot of the BBW4 application used to gather the presented results (its visualization) is displayed in figure 3. The application can be easily altered to inject different kind of faults, e.g. to inject into different combination of nodes or to inject only during certain protocol execution phase.

ADVANTAGES AND POSSIBLE DEPLOYMENT

The method of simulation-based verification of safety critical systems is not new and many sources deal with simulation modeling at different architectural levels. What makes our method unique is its wide usability:

- *Functional reference model*: the C-language model of the system's interconnection protocol can be used from the early stages of development as an exact definition of the protocol (e.g. more exact than a written semi-formal specification). Parts of the code can be verified separately. This enables to perform various experiments in early stages of new protocol version preparation. In later stages the exact C-model serves as reference for any HW based implementation.

- *Executable model*: The functional protocol reference model can be embedded within a simulation environment (in our case the C-Sim library) to provide the basis for building executable models of the whole system including applications. An application can be developed and debugged relatively easily using the model even when the modeled device is not yet available. Using a wide set of simulation applications (specially designed for testing, i.e. not necessary real-world applications), it is possible to generalize the obtained results. We can easily implement different fault tolerant mechanisms (active replication, TMR, repeated execution, etc.) and evaluate the effectiveness of these different approaches.

- *Real-world applications*: The ANSI standard of C language, which is used, enables to link the model with any C-written application. This is important because today, most industrial applications (for embedded computers) are written in C language. There are two possible ways to utilize this: We can take an existing application and verify it under arbitrarily severe conditions (unlikely to happen in the normal operating environment). The second utilization is to develop a completely new application using the model and after thorough verification and debugging port this application into the real device. This porting requires usually only minor modifications to the source code. The simplicity of the porting process (resulting from the same used programming language) reduces the probability that a fault in the application will be introduced during the transformation.

- *Portability and performance*: The simulation can run on any system that provides a C language compiler (this means almost every computer

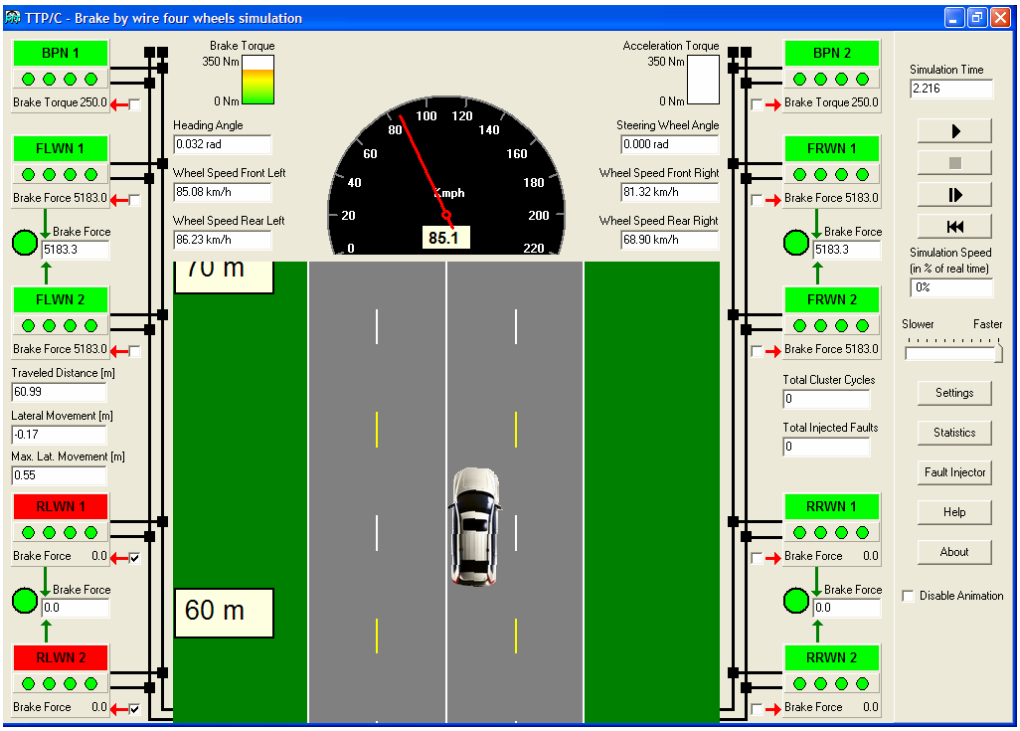

Figure 3: Captured screen from the visualized BBW4 application

system). The experiments were done on several platforms, including: PC with Linux, Windows NT and Windows XP, supercomputer Digital AlphaServer 8400 5/300 with Digital UNIX V4.0E. The performance of the model is determined mainly by the applications design (i.e. number of nodes, computation complexity). For example, on the a 1.4GHz PC the synthetic application sine wave (4 nodes) executes approximately four times faster than real-time and the real-world application BBW4 with 10 nodes executes at half speed of real-time.

- *Analyzability of results:* Because C-Sim based simulation is fully deterministic, we are able to analyze discovered problems at arbitrary level of detail. This enables to pinpoint to source of a problem exactly, i.e. to find whether the problem is caused by a faulty implementation or by a flaw in the system design.

- *Abstraction level* — Limits possible fault injection targets. In our case we are able to perform FI into any memory field of TTP/C that is defined in the official specification, into transmitted messages and into any application defined field. It is impossible to influence internal registers (not covered by the specification) and code.

- *Fault Nature* — only memory based faults are possible, moreover permanent faults are difficult to simulate.

- *Set-up difficulty* — depends on the application and availability of a ready-to-use solution (as the BBW4 made by Volvo). To build a model from scratch is very time-consuming.

- *Reusability* — different for all parts of the model. The C-Sim tool is highly reusable and the TTP/C C reference model can be without any difficulty used for any other application. The real-world application is bound to a particular purpose and cannot be used anywhere else except the real device.

CONCLUSIONS

The presented case study (the BBW4 application) shows many of the benefits that a simulation can offer. The most obvious one is that a HW implementation would not even be possible in advance. Moreover, the simulation tool enables us to add a visual user interface to the model as well as a fault injection capability. Such an interface can be used either for more sophisticated experiments or for demonstration purposes, as it interactively displays the current state of simulation.

It is obvious that evaluation and verification based on simulation can never provide complete assurance of the safety of the modeled system or application. In reality this cannot be guaranteed by any single verification method. The advantages of our approach are clearly stated in section *Advantages and Possible Deployment*.

Usefulness of the described approach and mainly the mentioned case-study (BBW4) was proven within the EU FIT project (Final report of the FIT project 2002).

The model of a newer version of the TTP/C protocol (the C2 chip) is currently developed and in the future we plan to adapt the testing tools to this new model. Also the current BBW4 application should be extended to allow a wider range of experiments.

ACKNOWLEDGMENT

The research was in part supported by a grant of 5th Framework Program Information Societies Technology: IST-1999-10748 Fault Injection for Time Triggered Architecture (FIT). Theoretical part of work was supported by the Ministry of Education of the Czech Republic, project no. MSM-235200005: Information systems and Technologies.

REFERENCES

Ademaj, A.; P. Grillinger; P. Herout; and J. Hlavička. 2002. "Fault Tolerance Evaluation using two Software Implemented Fault Injection Methods". In *IEEE International On-Line Testing Workshop (IOLTW 2002)*, Isle of Bendor, France, July 2002, pp. 21-25.

Final report of the FIT project, IST-1999-10748.

Grillinger, P. and S. Racek. 2002. "Transient faults robustness evaluation of safety critical systems using simulation". In *Baltic Electronic Conference (BEC 2002)*, Tallinn, Estonia, October 2002.

Heiner, G. and T. Thurner. 1998. "Time-triggered architecture for safety-related distributed real-time systems in transportation systems". In *Proceedings of FTCS-28*. Munich, Germany, pp. 402–407.

Hlavička, J.; S. Racek; and P. Herout. 2001. "Modeling a Fault-Tolerant Multiprocessor System". In *IEEE Conference EUROCON 2001*. Bratislava (Slovakia), July 2001, pp. 544-547.

Hlavička, J.; S. Racek; and P. Herout. 2000. "Evaluation of process controller fault tolerance using simulation". In *Simulation Practice and Theory*. Volume 7, Issue 8, 15th March 2000, pp. 769-790.

Herout, P.; S. Racek; and J. Hlavička. 2002. "Model-based dependability evaluation method for TTP/C based systems". In *Proceedings of European Dependable Computing Conference (EDCC-4)*. Toulouse, France, October 2002, pp. 271-282.

Krejzek, T. 2002. *Verification of Application Reliability in TTA*. Master Thesis. Czech Technical University in Prague, June 2002.

Kopetz, H. 1997. *Real-Time Systems, Design Principles for Distributed Embedded Applications.* Kluwer Academic Publishers, 1997

Laprie, J.C. 1992. *Dependability: Basic concepts and terminology.* Springer-Verlag Wien New York, 1992, 265 pp.

Lönn, H. 2001. *Brake by wire status report.* Status report of the EU FIT project, Prague.

Manzone, A. et al. 2001. "Fault tolerant automotive systems: An overview". In *Proceedings of 7th Int'l On-Line Testing Workshop.* Taormina, Italy, July 2001, pp. 117-121.

http://www.c-sim.zcu.cz — Home pages of the C-Sim simulation tool.

http://www.cti.ac.at/fit — Home pages of EU project Fault Injection for TTA (FIT).

AUTHOR BIOGRAPHIES

PETR GRILLINGER was born in Czech Republic and went to the University of West Bohemia in Pilsen where he studied computer engineering and obtained his degree in 2001. He finished his master thesis as a part of the EU project FIT and now continues his PhD study of simulation based fault injection techniques at the same university. His e-mail is: `pgrillin@kiv.zcu.cz` and his web-page can be found at the address `http://www.kiv.zcu.cz/~pgrillin`

PAVEL HEROUT was born in Czech Republic. He graduated in 1985 at the Institute of Technology in Pilsen in specialization Electronic computers. In 1999 he defended his PhD thesis in computer science. He works as a teacher at the University of West Bohemia in Pilsen and delivers lectures and seminars in subjects Object Oriented Programming, Programming in the C Language and Desktop Publishing. His professional interests are programming languages, simulations and fault-tolerant computing. His e-mail address is: `herout@kiv.zcu.cz` and his web-page can be found at `http://www.kiv.zcu.cz/~herout`