Towards a DECOS Fault Injection Platform for Time-Triggered Systems

H. Eriksson, J. Vinter, B. Leiner, and M. Schlager

Abstract—**This paper presents a fault injection platform targeting the communication bus in the DECOS platform, which is using a time-triggered communication protocol such as TTP or FlexRay. Communication errors are injected by a disturbance node, which emulates errors caused by external sources, e.g. from electrical relays in road vehicles or from lightning affecting airplanes. The platform is flexible and the communication protocol can be reconfigured from TTP to FlexRay or vice versa. The experiments are configured by XML scripts and controlled by Lauterbach TRACE32 software and hardware. Raw data from the experiments are stored in SRAM memory without halting the program execution so only minor time intrusiveness is introduced for logging data. After each experiment raw data are downloaded from memory, automatically parsed and converted into integer and e.g. float data types, and finally stored in a MySQL database for analysis. Several analysis functions are developed to evaluate the effectiveness of hardware-implemented and software-implemented error detection and recovery mechanisms.**

I. Introduction

EMBEDDED computer systems are increasingly being used to protect large investments or human lives. Validating the dependability of such systems is an essential part of the design process. *Fault injection* [1], which is a way of accelerating the occurrences of faults in a system, has become an important method for system engineers to experimentally validate the dependability of computer systems. The main purpose of fault injection is to evaluate and debug the error detection and recovery mechanisms.

Fault injection can be used at various abstraction levels depending on the information available about the system and at which stage of the design process it is applied. Fault injection techniques can be divided into *simulation-based* and *physical* techniques depending on whether faults are injected into a model of a system, or into an actual physical system or prototype. The advantage of simulation-based fault injection is that it can be used early in the development process before the actual system is available which facilitates early discovery of design deficiencies. Physical fault injection is important since it allows the actual implementation of the system to be tested.

Many safety standards, e.g. IEC 61508 [2], require that fault injection shall be performed as one validation activity. In IEC 61508, fault injection is even mandatory for all safety integrity levels (SILs) if the claimed diagnostic coverage is larger than 90%. Additionally, within all application domains there are requirements to test the immunity to external interference, e.g. EMI – electromagnetic interference.

The DECOS (Dependable Embedded Components and Systems) technology [3] is targeting development of dependable distributed integrated systems. Temporal and spatial partitioning between applications are ensured by encapsulated execution environments (in nodes) [4] and virtual networks (communication bus) [5]. The foundation for DECOS is a time-triggered core architecture which has to provide services such as predictable message transport and fault-tolerant clock synchronization. To show the portability of the DECOS technology, the aerospace and automotive demonstrators in DECOS use the time-triggered protocols TTP and FlexRay, respectively.

This paper presents a physical fault injection platform for applications built on the DECOS technology. Since fault injection now can be performed on system prototypes before the final hardware is accessible, it can help to meet certification requirements such as the ones mentioned earlier. The platform can be configured to use different data communication protocols such as TTP and FlexRay. A Lauterbach debugging environment is used to control the experiments and several configuration and analysis functions are implemented in, or executed from, a Java program called FIM (fault injection manager).

The remainder of the paper is organized as follows: Section II describes the fault injection platform as well as the configuration and execution workflow. Examples of target systems that can be validated by the platform are presented and discussed in Section III. Finally, the conclusions are given in Section IV.

This work has been supported by the European IST project DECOS under project No. IST-511764.

H. Eriksson is with the SP Technical Research Institute of Sweden, Brinellgatan 4, SE-501 15 Boras Sweden, phone: +46105165000; fax: +46105165635; e-mail: henrik.eriksson@sp.se.

J. Vinter is with the SP Technical Research Institute of Sweden, Brinellgatan 4, SE-501 15 Boras Sweden; e-mail: jonny.vinter@sp.se.

B. Leiner is with TTTech Computertechnik AG, Schoenbrunner Strasse 7, A-1040 Vienna, Austria; e-mail: bernhard.leirner@tttech.com.

M. Schlager is with TTTech Computertechnik AG, Schoenbrunner Strasse 7, A-1040 Vienna, Austria; e-mail: martin.schlager@tttech.com.

II. Fault Injection Platform

This section describes the hardware and software infrastructure of the fault injection platform as well as the workflow used during configuration and execution of fault injection campaigns.

A. Hardware and software infrastructure

The physical fault injection platform consists of: a number of DECOS prototype nodes, a TTX-Disturbance Node [6], a Lauterbach TriCore Debugger, and a desktop PC executing the Lauterbach debug software and the FIM, see Fig. 1. An oscilloscope can be connected to the disturbance node to observe the data traffic on one or both of the individual bus lines.

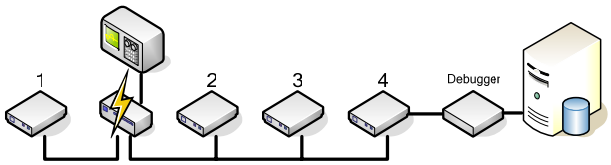

Fig. 1. Fault injection platform.

The disturbance node used is the TTX-Disturbance Node from TTAutomotive which is targeting the FlexRay protocol but the protocol-independent disturbances can be used for TTP as well. The disturbance node has a primary and a secondary bus connector. This makes it possible to fractionate the bus into two separate parts. The disturbance node supplies protocol-independent disturbances such as mismatched bus terminations, cross-talk, as well as short circuit to the supply voltage or to ground. A useful protocol-independent disturbance is the provision of a pulse of specific (or random) duration at a specific (or random) time after an input trigger has been detected, e.g. the start of a new cluster cycle. If necessary, this pulse can be repeated at a specific (or random) frequency to create intermittent or permanent faults. This injected pulse can be permanent high (stuck-at-1), permanent low (stuck-at-0), or white noise.

An example of fault-free bus communication displayed on an oscilloscope screen is shown in Fig. 2. In this example the cluster cycle is 5 ms and there are 2 rounds. The cluster has four nodes and consequently each time slot is 625 µs. As can be seen in the figure, the bus is idle most of the time. This is due to the fact that it is a small application and the nodes do not send much data.

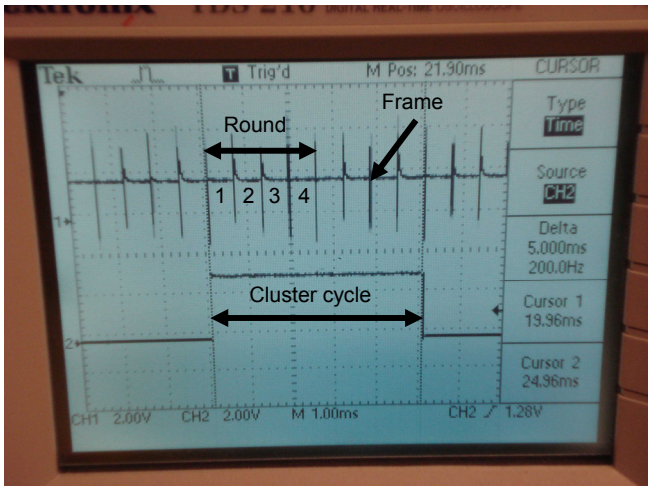

Fig. 2. Example on fault-free bus communication.

Fault injection scenarios are defined in XML files. In these files the user defines the fault models, i.e. the fault types, fault locations, the frequency of faults and e.g. the number of faults. That is:

- if the disturbance shall occur at a specific time after a new cluster cycle or at a random point in time and its duration,
- if the disturbance shall be injected only on channel A, only on channel B, or both, and
- if the disturbance shall be high, low, or white noise.

The interesting parts of an XML scenario file for a 3 µs high pulse disturbance are presented below and the resulting disturbance in Fig. 3. In this figure the TTP frame has been magnified to be able to see the disturbance pulse.

```
<scenario name="SINGLE_HIGH_PULSE">
        <wait_for_trigger_in edge="RISING"
name="TRIGGER_1">
  <offset unit="us" value="1312" />
  </wait_for_trigger_in>
  <run_scenario name="Disturbance" loop="1" />
</scenario>

<scenario name="Disturbance">
   <disturbance>
      <channel CH_name="CH_A">
       <driver mode="STRONG" />
       <signal_high />
          <duration value="3" unit="us" />
     </channel>
  </disturbance>
</scenario>
```

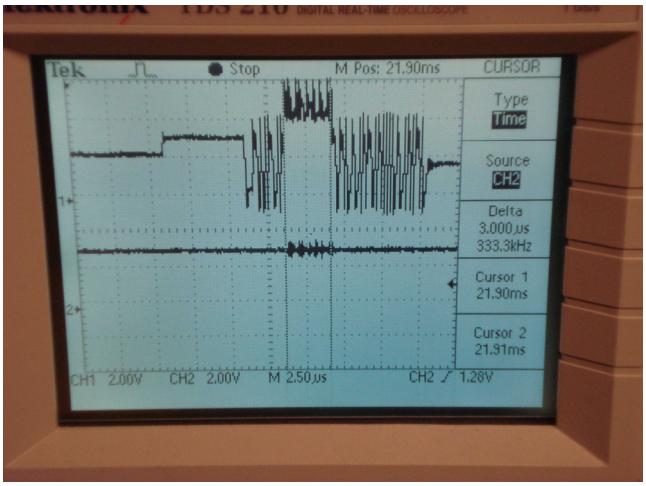

Fig. 3. A TTP frame with 3 µs high disturbance.

The scenario file has to be loaded into the disturbance node, which is accomplished by a serial connection from the PC. A connection is configured in the HyperTerminal program in Windows and then the XML file is downloaded.

One option for logging data is to use the serial port of the node and continuously send data to the PC. However, the bandwidth of a serial connection is limited and instead logged data are stored in internal memory of the node, and after the experiments are finished the logged data are downloaded to the PC via the Lauterbach debugger. An example of a monitor task which logs some information in the external memory is shown in Fig. 4. The first part of the code defines addresses to status areas as well as message replicas. In the status areas there is information on if the received frame of the present slot was valid or not. Also, it can be checked if a specific message was received on both channels or only on one. The addresses to status areas and message replicas are application-dependent and node-dependent and can implicitly be found in an XML file containing the C-MEDL – message description list. This XML file is created for the corresponding node by the tool TTP-Build [7]. The FIM automatically extracts these addresses and inserts them into the monitor task shown in Fig. 4.

To keep track of the memory usage, a memory counter is used. The memory counter is incremented by the appropriate number of bytes as soon as data have been written to the memory. The logging continues until the memory is full or the fault injection scenario is finished.

Obviously, all information on data types is lost during the dumping exercise; the debugger treats the data as plain hex values. Therefore, a parser is necessary. The parser takes the memory dump and a configuration file as inputs and produces a text file (containing e.g. converted integer values) suitable to load into a database such as MySQL.

```
#define address_int (unsigned int *) 0x82000000 //Address to log memory
#define Memory_MAX 1048576 //Size of log memory (1 MB)

#define s1r1cha (unsigned int *) 0xA3004810 //Frame status address, slot 1, round 1, channel A
#define s1r1chb (unsigned int *) 0xA3004828 //Frame status address, slot 1, round 1, channel B

//Addresses to message box statuses (slots 1, 2, and 3)
#define s1r1_Average_float (unsigned int *) 0xA3004b14
#define s2r1_Average_float (unsigned int *) 0xA3004b1c
#define s3r1_Average_float (unsigned int *) 0xA3004b0c

//Addresses to message replicas
#define s1r1_Average_float_r1 (unsigned int *) 0xa3004b10
#define s2r1_Average_float_r2 (unsigned int *) 0xa3004b20
#define s3r1_Average_float_r3 (unsigned int *) 0xa3004b18

//Memory pointer of log memory
unsigned int *base_address = address_int;

//Memory counter
int mem_cnt = 0;

void monitor_task (void){

 ST_msg_RD_Average_float = tt_Message_Status(msg_RD_Average_float);

 if (mem_cnt <= Memory_MAX - 40){
 *  base_address = * s1r1cha;
 base_address++;
 *  base_address = * s1r1chb;
 base_address++;
 * base_address = ST_msg_RD_Average_float;
 base_address++;
 memcpy(base_address,&msg_RD_Average_float,sizeof(msg_RD_Average_float));
 base_address++;
 * base_address = * s1r1_Average_float_r1;
 base_address++;
 * base_address = * s2r1_Average_float_r2;
 base_address++;
 * base_address = * s3r1_Average_float_r3;
 base_address++;
 * base_address = * s1r1_Average_float;
 base_address++;
 * base_address = * s2r1_Average_float;
 base_address++;
 * base_address = * s3r1_Average_float;
 base_address++;
 mem_cnt += 40;
  }
}
```

Fig. 4. Example of a code sequence in a monitor task.

MySQL is a free SQL server that is optimized for speed and in this experimental platform it is managed and used directly from the FIM. The FIM is also in charge of executing the parser. SQL queries which are assumed to be asked only a few times can be written directly in the MySQL Query Browser. However queries that could be used for several fault injection campaigns are defined directly in the FIM to be automatically executed. If intermediate results needs to be stored in variables, they are also defined in the FIM. Database columns are dynamically assigned an appropriate data type and name by using the configuration file. A multitude of SQL queries can be asked to the database to analyze the data. An example is shown in Fig. 5, where all replicas of the message 'MSG_RD_-AVERAGE_FLOAT' and the agreed message are selected.

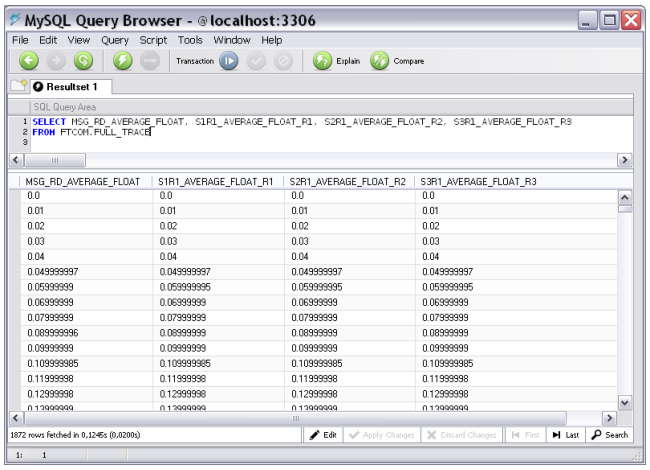

Fig. 5. Query using the MYSQL Query Browser.

B. *Configuration and execution workflow*

The configuration and execution workflow is presented in Fig. 6. At the beginning, the workflow starts by selecting the workload which will be used during the fault injection campaign (a set of experiments). Tasks for logging activities should be scheduled to be run in the slack times of the workload and get a dedicated memory space. In a dependable integrated architecture such as realized within the DECOS project, it is important to take into account the partitioning mechanism, i.e., a log task can only access variables and status areas within its own partition. Furthermore, it is important to remember that logging tasks for e.g. a frame status shall not be run at the same time as the status is being updated, i.e. the receiving instant. As explained earlier, if status information concerning the communication shall be logged, e.g. message replicas, these addresses can easily be extracted from the C-MEDL file.

In parallel with the set-up of the workload and the logging tasks, the fault injection campaign files can be defined and implemented. After the compilation, linking and all the loading are completed; the scenarios are run until there are no scenarios left. For each scenario the log memory is limited and therefore one has to select what to do when the memory is full. Either the logging is stopped when the memory is full (as shown in the figure) or the memory pointer is reset and the old data is overwritten, i.e. a circular buffer. For each scenario the memory is dumped to a text file on the PC. When the campaign is finished, the files are parsed and loaded into the database for analysis.

If the workload does not fulfill its requirements on e.g. fault tolerance, a redesign is necessary and when it is completed the workflow starts all over again. It could also be the case that the defined campaign was insufficient and that a new campaign with supplementary scenarios has to be run.

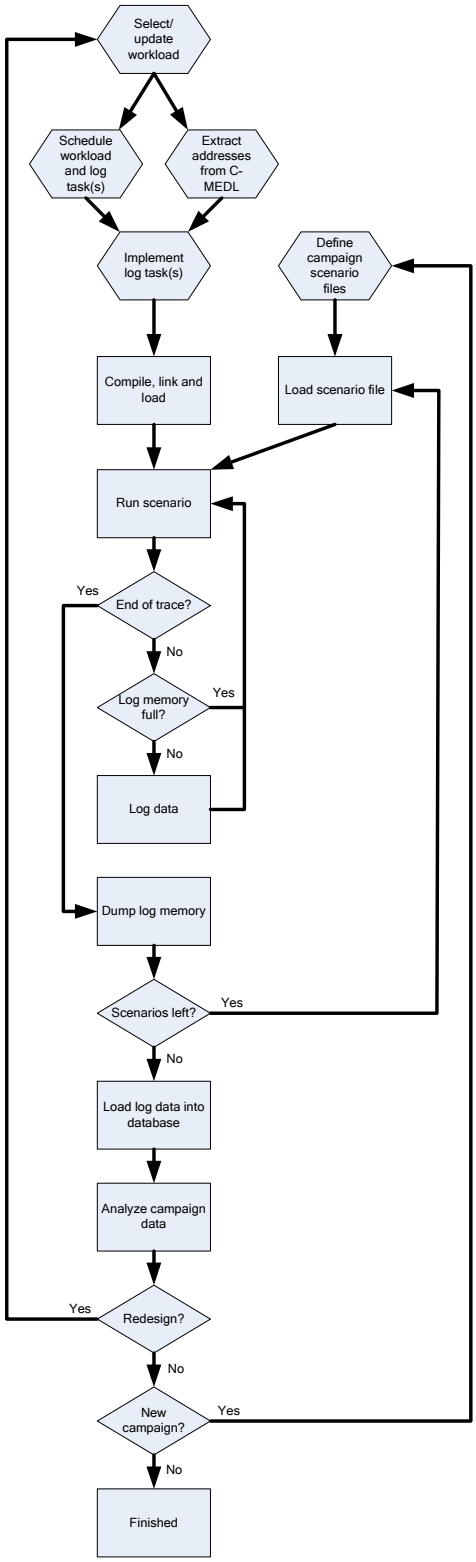

Fig. 6. Fault injection configuration and execution workflow.

III. APPLICATION AREAS

This section provides some suggestions on how the fault injection platform can be used together with automotive and aerospace applications.

A. Automotive

The standard ISO 7637-3 [8] is one of many EMC-related standards for road vehicles. This specific standard defines test pulses and a test set-up for worst-case disturbance coupling to signal lines where the signal consists of pulse trains separated by silence, see Fig. 7. Each pulse train contains 100 pulses and the rise and fall times of each individual pulse are specified in the standard. A source of this kind of disturbance could be a relay controlling e.g. the directional indicators of the car.

When experiments according to this standard are performed, the disturbance signal is capacitively coupled to the communication bus. Hence, there will be a voltage division between this capacitance and the impedance of the bus and as a consequence the resulting disturbance on the bus will very much depend on the bus, nodes, and their configuration.

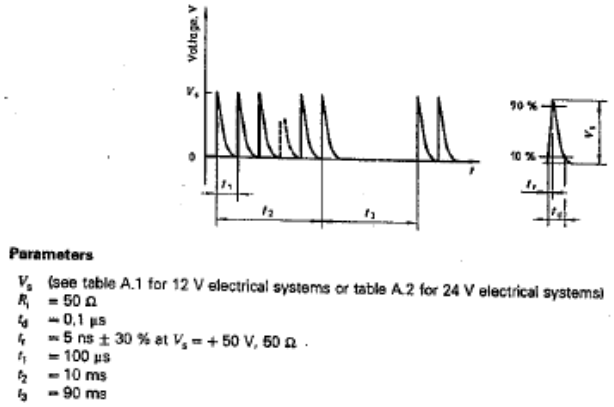

Fig. 7. Test pulses according to ISO 7637-3.

Test pulses defined in standards are idealized to fit laboratory work but do not cause disturbances which exactly match those experienced in real cars [9]. Obviously, pulses generated by the disturbance node cannot perfectly mimic real disturbances. However, pulse trains consisting of alternating ones and zeros, separated by silence, can easily be injected by the disturbance node, see Fig. 8. Although this is a large simplification, roughly the same number of frames on the bus will be affected by the disturbances and the fault tolerance of the application can thus be evaluated. Since these experiments can be run on prototypes of the system, early indications on the robustness can be achieved.

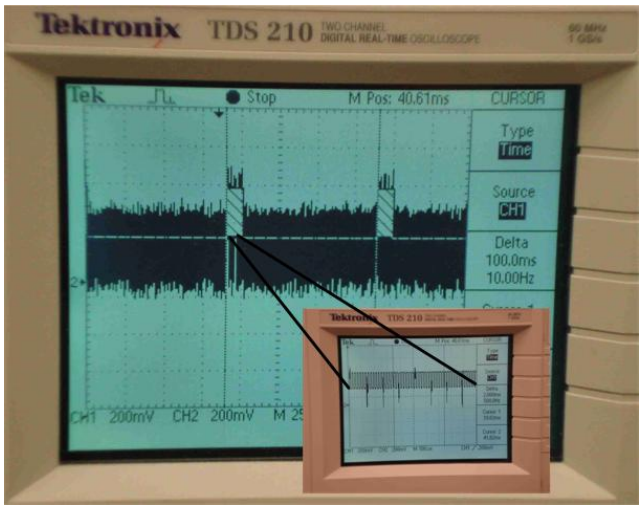

Fig. 8. Disturbances mimicking ISO 7637-3.

B. Aerospace

Similar to the automotive domain there are a lot of standards dealing with EMC issues. In the aerospace domain an important source of disturbances is lightning. Disturbances induced by indirect lightning are described in the standard MIL-STD-464 [10] and the corresponding pulses are shown in Fig. 9. Another standard dealing with the same issue is the FAA Advisory Circular 20-136A [11] which refers to the standard RTCA/DO-160E for the test procedures. Disturbances similar to those in Fig. 9 seem to be used in RTCA/DO-160E [12].

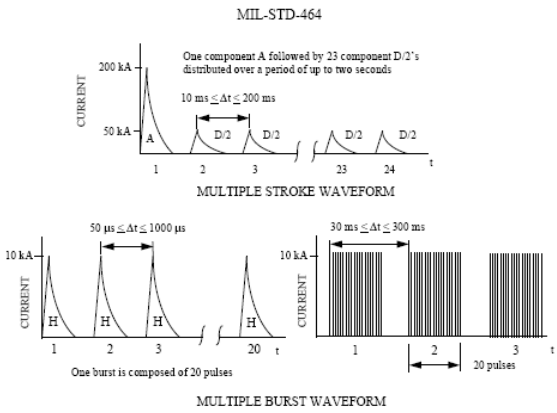

Fig. 9. Description of disturbances from indirect lightning (MIL-STD-464).

The disturbance which can be mimicked by the fault injection platform is the multiple burst waveform which resembles the test pulses of ISO 7637-3. The difference is that the distances between pulses in the pulse train and the duration of silence are specified as ranges and not as exact values. Since the disturbance node offers the possibility of having random delays, the bounds for the random delays in

the XML scenario files are set to the ranges specified in the standard. An example of a pulse train (burst) injected from this kind of scenario file is shown in Fig. 10. The pulses are not equidistant (as can be seen in the figure) and neither are the bursts.

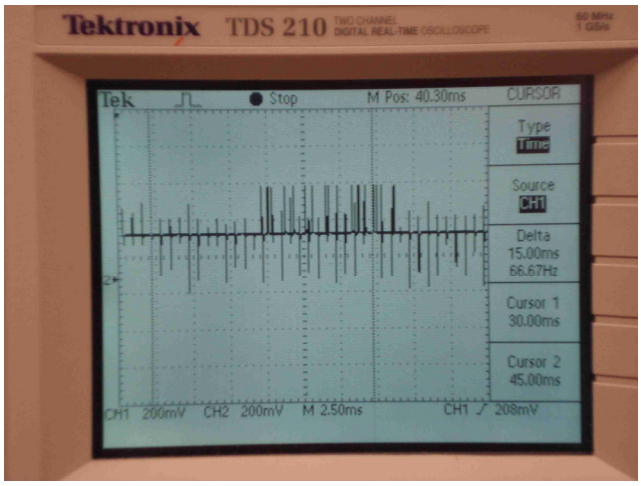

Fig. 10. Disturbances mimicking the effects from indirect lightning.

IV. Conclusions

Fault injection is an important method to experimentally validate the fault tolerance of computer systems. Fault injection is also mandated by important international safety standards such as IEC 61508.

The DECOS technology shall be able to facilitate the development of applications having the highest demand on safety integrity. DECOS is also a *distributed* integrated architecture which means that safety relevant data are sent between nodes on replicated buses and that fault tolerance is increased by having redundant nodes.

With these two facts in mind, it makes sense to provide a fault injection platform which injects communication faults in the DECOS bus system regardless of communication protocol, i.e. TTP or FlexRay.

The areas where integrated architectures will be introduced first are probable automotive and aerospace. Therefore standards from these areas which consider external error sources such as EMC and lightning have been studied and disturbance scenarios based on these standards prepared.

References

[1] R.K. Iyer, “Experimental evaluation”. Special Issue of the *Proceedings of 25th International Symposium on Fault-Tolerant Computing*, 1995. pp. 115-130.

[2] IEC 61508:1-7 Functional Safety of Electrical/-Electronic/Programmable Electronic Safety-Related Systems, IEC, 1998, 2000.

[3] H. Kopetz et al, "From a federated to an integrated architecture for dependable embedded real-time systems", Technical Report 22, Institut für Technische Informatik, Technische Universität Wien, 2004.

[4] M. Schlager, et al., "Encapsulating application subsystems using the DECOS Core OS", in *Proceedings of the 25th International Conference on Computer Safety, Reliability, and Security (SAFECOMP)*, pp. 386–397, Gdansk, Poland, September 27-29, 2006

[5] R. Obermaisser and P. Peti, "Realization of virtual networks in the DECOS integrated architecture", in *Proceedings of the 20th International Parallel and Distributed Processing Symposium*, 2006.

[6] TTX-Disturbance Node User Manual, Edition 1.0.4, TTTech Computertechnik AG, 2006.

[7] TTP-Build - The DECOS Cluster Design Tool for Layered TTP (Prototype), Edition 5.3.69a, TTTech Computertechnik AG, 2006.

[8] ISO 7637-3 “Road vehicles – Electrical disturbance by conduction and coupling – Part 3: Vehicles with nominal 12 V or 24 V supply voltage – Electrical transient transmission by capacitive and inductive coupling via lines other than supply lines”, ISO, 1995.

[9] R. K. Frazier and S. Alles, “Comparison of ISO 7637 Transient Waveforms to Real World Automotive Transient Phenomena“, 2005.

[10] MIL-STD-464 “Electromagnetic environmental effects requirements for systems”, US Department of Defense, 1997.

[11] FAA Advisory Circular 20-136A “Protection of aircraft electrical/electronic systems against the indirect effects of lightning”, FAA, 2006.

[12] E. J. Borgstrom, “EMC requirements for avionics: RTCA/DO160E”, Interference Technology, www.interferencetech-nology.com, 2005.