Improving Fault Injection in Automotive Model Based Development using Fault Bypass Modeling

Rakesh Rana, Miroslaw Staron, Christian Berger, Jörgen Hansson
Computer Science & Engineering
Chalmers/ University of Gothenburg
Gothenburg, Sweden
rakesh.rana@gu.se

Martin Nilsson, Fredrik Törner
Volvo Car Corporation

Abstract: Fault injection is widely used for validating dependability of computer systems. These techniques have been traditionally used for testing dependability of the both hardware and software systems. With widespread use of model based development in automotive software development more sophisticated needs arise for using fault injection techniques at the model level, which can yield significant benefits in combination with model-based testing or model mutation. In this paper, we address challenges with injecting faults into behavioral models in terms of analysis of results and propose a framework for distinguishing between correct and incorrect simulation results. The focus is laid on an important challenge encountered when injecting faults in continuous models – managing system-environment interdependencies. We analyze the problem in details and outline an effective approach to deal with this problem.

1 Introduction

Dependability is a very important feature of any computer system [BOE87]. A crash or system failure causes discomfort for consumers who then associate the product with low quality. For certain safety critical applications, such as within aerospace and automotive, these failures may result in life threatening situations for the occupants. Therefore, dependability validation is strictly monitored and regulated in development of safety critical applications – for example by following standards like ISO 26262 [ISO11] in automotive and DOC-178C [BRO11] for safety-critical software development within the aerospace industry.

Fault injection is an important and widely used technique; it has been extensively used as technique for experimental dependability evaluation of computer systems and for evaluation of fault-tolerance mechanisms. Fault injection has traditionally been used for emulating hardware and software faults on the prototypes. The prototype-based fault injection has its advantages as given in [BOE87]:

- Identify dependability bottlenecks,

- Study system behavior in the presence of faults,
- Determine the coverage of error detection and recovery mechanisms, and
- Evaluate the effectiveness of fault tolerance mechanisms (such as reconfiguration schemes) and performance loss.

Alongside the evolution of requirements for dependability, software product development methodologies have significantly changed over last decade. Model Based Development (MBD) has been embraced widely as preferred tool for developing software systems. Many sectors such as aerospace and automotive domain today widely use MBD for new software/function development [MST12]. As a consequence of this widespread use of MBD, code generation has also gained momentum in these industries and Model Based Testing (MBT) alongside with it. Traditional software development within automotive and many other domains can be approximated to the established V-model. In this model of software/product development, testing is heavily focused on the right side, which leads to most defects being discovered late in the development process; however MBD has the potential of shifting some of the effort spent on Verification & Validation (V&V) to the left arm of V-model, i.e. by allowing testing of models at early development phases some of the defects can be detected much sooner, this shift could save significant amount of resources/cost for the projects [BOE87] and also reduce development time.

Megen and Meyerhoff in their study [MM95] showed that about 50% of defects detected during testing are found in test preparation, an activity which does not depend on the executable code. Still previous case studies within the automotive domain, peak in open defects is observed during the late stages of software development just before the release dates [MST12]. These observations suggest that by using MBD and through testing models early in the development phase, a possibility exists for shifting the open defect peak from the right-hand side (close to release) towards earlier phases of the project. This shifting of defect detection to the left (early detection) can provide better opportunity for the teams to react early upon problems and thus avoid late defects. MBD offers an important advantage with respect to early testability – the executable models can be simulated to verify and validate the functionality at a very early stage of development process. And by using fault injection in these behavioural models, dependability measurements can also be done very early on; robust, high fault-tolerant models thus can be identified and developed. The functional and logical errors can also be identified early in the development process, thereby saving development time and costs significantly.

For a majority of software functions developed within automotive and other domains – real time behaviour is an important factor and often test cases depend on the system behaviour [BK06]. One way to test the complex functional behaviour of such systems in real time is to use closed loop testing of models with continuous signals, but testing systems with continuous signals is poorly supported by existing test methods which are generally data-driven [BK06]. Two major obstacles for testing behavioural/functional models of software systems in a closed loop configuration with continuous signals using fault injection techniques are:

- Realistic environment modelling, and

- Obtaining realistic system response in presence of injected faults to correctly differentiate between normal system behaviour from system failure.

With regard to the first obstacle, most industrial domains currently using MBD on large extent are either using or are developing virtual environmental models, which are capable of simulating full system and environment conditions that are very close to real ones. These models are intended not only to aid and accelerate new conceptual/functional development, but also to test developed functional models in MIL (Model in Loop) testing for possible logical errors [VBRE07]. Development of these extensive environmental models addresses the first challenge.

In this paper we take a closer look at the second obstacle (b) and propose a framework to overcome it. The rest of this paper is structured as follows. In the next part (section 2) we provide an overview of fault injection techniques, which is followed by a discussion on related work in section 3. In section 4, we describe and analyse the problem with injecting faults on behavioural models using an example. Section 5 introduces the proposed solution to the described problem followed by conclusions in section 6.

2 Fault Injection

A system may not always perform as intended or expected. The causes and consequences of system performance deviations from the expected function are referred as factors of dependability. Dependability evaluation involves study of failures and errors. Important aspects of dependability are defined comprehensively in work by Avizienis et al. [ALR01]. Dependability is particularly important when system/software being developed is safety critical, where failure can cause serious hazard or loss of life. Functional safety standards such as IEC-61508 [SVET10] mandate use of fault injection technique for system development, while it is recommended or highly recommended in automotive safety standard ISO 26262 [FAG02].

Based on injection of faults to the actual hardware/prototype or on its model, fault injection techniques can be classified as physical or simulation-based. While based on the implementation of fault injection mechanisms, the techniques can be classified as hardware-implemented fault injection (HIFI) or software-implemented fault injection (SWIFI) techniques. A brief overview of fault injection techniques is provided here whereas a more detailed description on different fault injection techniques, their advantages, drawbacks and important tools is presented in [HTI97], [ZAV04].

In hardware-based fault injection, faults are injected at the physical level by controlling the environment parameters. Faults may be injected by injecting voltage sags, disturb the power supply, heavy ion radiation, electromagnetic interference etc., while software-based fault injection refers to techniques that inject faults by implementing it in the software. Different types of faults can be injected with software based fault injection for example register and memory faults, error conditions and flags, irregular timings, missing messages, replays, corrupted memory etc. For using fault injection techniques where target is a software application, fault injector may be inserted within the application or it can also be inserted between the target and operating system, while in case where the target is the operating system, fault injector used have to be embedded within it itself [Boe87]. Software-based fault injection techniques can be classified into compile-time faults or run-time faults based on when the faults are injected. Software implemented fault injection methods can be adapted to inject faults on various trigger mechanisms such as exception, traps, time-out, code-modification etc.

TABLE 1: ISO 26262 RECOMMENDATION FOR USING FAULT INJECTION TECHNIQUES

ISO 26262 Chapter		**Reference to recommendation**
4	Hardware-software integration and testing	•Table 5 — Correct implementation of technical safety requirements at the hardware-software level. •Table 8 — Effectiveness of a safety mechanism's diagnostic coverage at the hardware-software level.
	System integration and testing	•Table 10a — Correct implementation of functional safety and technical safety requirements at the system level •Table 13b — Effectiveness of a safety mechanism's failure coverage at the system level
	Vehicle integration and testing	•Table 15 — Correct implementation of the functional safety requirements at the vehicle level •Table 18 — Effectiveness of a safety mechanism's failure coverage at the vehicle level
5	Hardware integration and testing	•Table 11 — Hardware integration tests to verify the completeness and correctness of the safety mechanisms implementation with respect to the hardware safety requirements
6	Software unit testing	•Table 10 — Methods for software unit testing
	Software integration and testing	•Table 13 — Methods for software integration testing

On the other hand, simulation-based fault injection [ZAV04] involves constructing a simulation model of given hardware using hardware description languages such as VHDL and faults are injected into these models during simulation. Two important approaches to inject faults within simulation-based techniques are:

- Those that need modification to VHDL code, and
- Those that use modified simulation tools using built-in commands of VHDL simulators.

Simulation-based fault injection techniques have been quite powerful and widely used for hardware models, though their application on behavioural models for software artifacts has been limited.

3 Related Work

While MBD has been widely adopted as the development methodology in automotive domain, quality assurance of MBD is still not well supported. Bringmann and Kramer [BK06] suggest that in practice only a few automotive domain-specific model based testing procedures are available, the applied test methods and tools are often proprietary, ineffective and require significant resource input in form of effort and money. They highlight the main requirements for successful automotive model based testing some of which are

- Reactive testing/ closed loop testing
- Real-time issues and continuous signals
- Testing with continuous signals

The authors also introduced TPT (Time Partition Testing) as a model based testing approach, its test cases can be used on different test platforms and can be executed in real time. Lamberg et al. [LBE04] also describes a systematic way of testing embedded software for automotive electronics, the process referred to as MTest allows model-based testing in early function and software development, but it does not use or suggest using fault injection for dependability evaluation. In this paper we introduce the fault bypass modelling (FBM) framework that addresses same requirements to allow closed loop testing with continuous signals but at model level under fault injection conditions.

Techniques for injecting faults into system models have been developed and evaluated in ESACS [ESACS] and ISAAC [ABB06] projects using SCADE (Safety-Critical Application Development Environment) modelling language to simulate hardware failure scenarios, these techniques were applied to identify fault combinations that lead to safety case violations. In [VBRE07] a model-implemented fault injection plug-in to SCADE called FISCADE is introduced which can replace original operators by fault injection nodes. During execution FISCADE controls the SCADE simulator to execute the model, inject the fault and log the results. Model based software implemented fault injection techniques have also been used for dependability evaluation of automotive functions such as in [JH05]; our work complements these earlier works by presenting a framework to manage system-environment interdependencies under fault injection conditions.

A further attempt to use fault injection techniques for software functional model evaluation is done with development of MODIFI tool [SVET10]. MODIFI (or MODel-Implemented Fault Injection tool) extends the fault injection methodology to behaviour models in Simulink. The tool allows for introducing single or multiple point faults on behavioural models, the fault tolerant system properties are studied by analysing faults leading to failure. But injecting faults in behavioural models is not same as injecting faults in physical hardware prototypes or their models; the interdependencies between the system and their environment may lead to inconsistent simulation results under fault injection modes. FBM offers a solution in form of fault bypass principle to the single factor causing incorrect system behaviour under fault injection mode for behavioural models.

4 Problem Analysis

Fault injection techniques are very useful to test the robustness and dependability of systems. The methodology is straightforward; however in practice for behavioural models, systems are not passively related to their environment. The relationship between system and its environment is active, which means that there are feedback loop(s) between the system and the environment. In such cases a fault injected into the system does not necessarily gives the output which is purely the property of that system, for system-environment models, any change in input of the system not only affects the system itself, but can also influence its environment which is again fed back to the system through single or multiple feedback loops. In continuous models it often results in unrealistic triggers/control values from system to its environment and in turn wrong/unnatural values for environment parameters to the system. Thus the outputs in these cases are unreliable making it difficult to distinguish between a correct system behaviour and system failure. We take a closer look on this problem with an example from automotive domain, the ABS.

4.1 CASE: ABS (Anti-Lock Braking System)

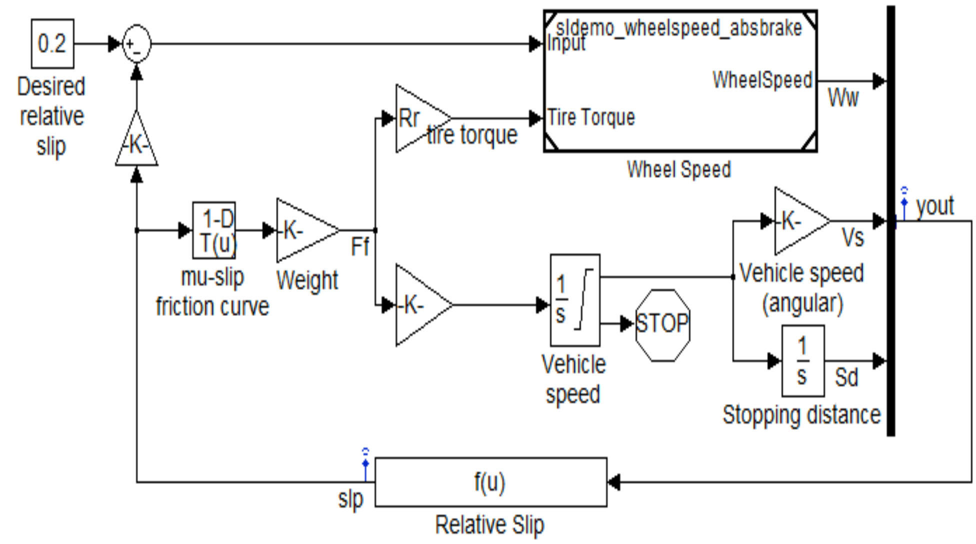

Figure 1: Simple ABS model representation in Simulink based on [17]. In this model wheel speed and vehicle speed is used to calculate the relative slip value, which is compared against the desired set value of relative slip (for maximum traction). The ABS controller (here embedded within the Wheel Speed block) activates/releases brake pressure according to anti-lock principle when relative slip value is different from desired slip value; rest of the blocks in the model are used to simulate the dynamics of a moving vehicle.

ABS system takes a given vehicle speed and the wheel speeds from different sensors on board the vehicle (its environment), the system uses these inputs to calculate relative slip value at each wheel. Based on this relative slip value a control signal is generated which controls the activation/deactivation of brake pressure valve in accordance to Anti-lock braking principle. A simplistic example of single wheel ABS system model in Simulink is presented in figure 1. (For detailed component description and working of model, refer to [Sim12]). The same system can be represented in a simplified form, which separates the elements of ABS system from its environment. The environment is collective term to represent the functions/parameters that are needed to simulate the system.

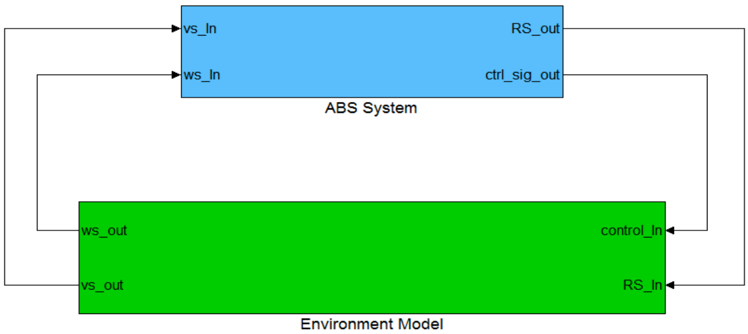

Figure 2: ABS model represented in System-Environment configuration. The ABS system takes input of Vehicle Speed (vs_in) and Wheel Speed (ws_in) as its input, calculates the relative slip value (RS) and using anti-lock braking principle outputs a control signal (ctrl_sig_out), which is used to control the braking force applied. For correctly executing the environment model, the relative slip value is also taken from system (RS_out), which introduces a superficial feedback loop in this case.

Executing the system-environment model to verify ABS functionality gives the following outputs describing vehicle speed, wheel speed which matches to expected output of ABS functionality (refer to figure 3). Solid lines represent Vehicle speed, while dashed lines are used for wheel speed.

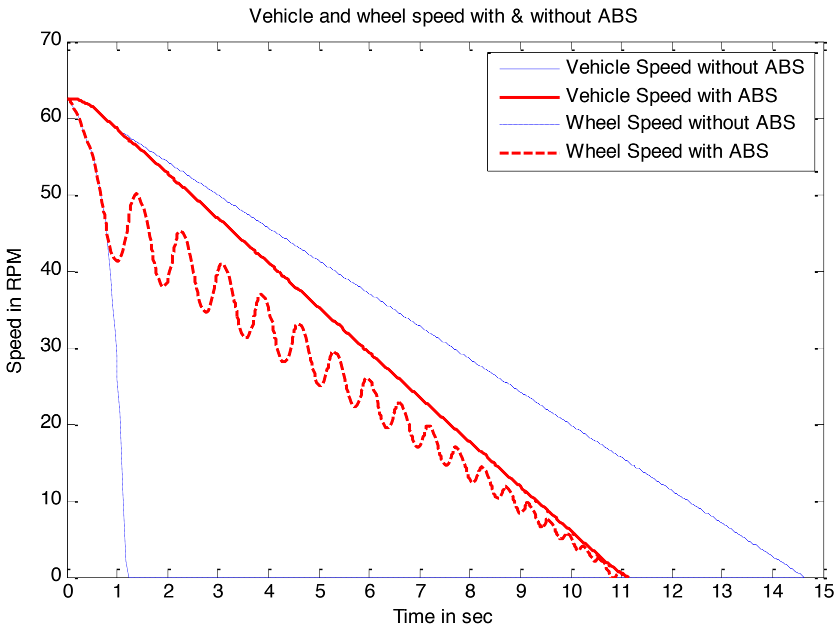

Figure 3: Vehicle and wheel speed with and without ABS system (solid lines represent vehicle speed while dotted lines, the wheel speeds). Note: The stopping times here are for illustrative and comparative purpose only (the given model simulates vehicle dynamics but all input parameters to this simulation do not correspond to any real vehicle model).

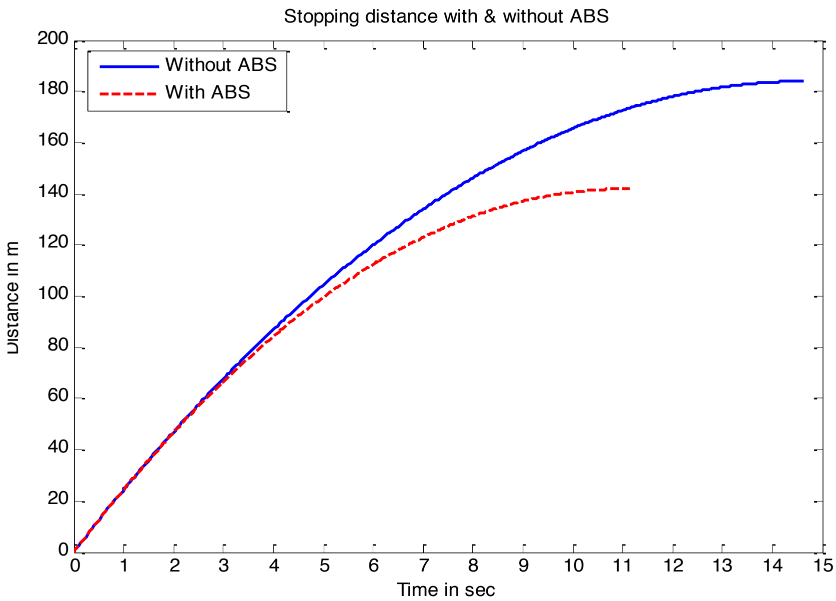

Figure 4: Stopping distance with and without ABS system shows decrease in stopping distance by about 22 meters.

Vehicle and wheel speed without an ABS system shows wheel locking at about t=1.2 sec and vehicle taking about 14.5sec to come to a standstill from the point of brakes application, while with the use of ABS system the wheel locking is prevented due to ABS system response and vehicle is stopped approximately at t=11sec. Also one can verify the reduction in stopping distance obtained by suing the ABS system compared to one without it, indicated here in figure 4.

4.2 Naïve fault injector

In Figure 5 we show an additional block - the fault injector - capable of generating different faults and injecting them into the input or the output signals of our ABS system. The fault injector can be used to inject faults both at the input signal and the output signals of system to evaluate fault tolerant capacity of given system implementation and also to study the system's behaviour/characteristics under these conditions. To simulate a situation where a wheel speed sensor is faulty, we inject a fault at wheel speed signal. For example injecting a fault by adding a high pulse starting at t=6sec, this high constant pulse injection simulates a real condition approximation where the sensor relays permanently high value as its output signal due to a fault/sensor failure. The resulting vehicle and wheel speed of our system-environment model simulation under given fault condition is presented in Figure 6. Under the given state/scenario, model simulation shows vehicle speed increasing exponentially starting at t=6sec (the instant of fault injection) which is unrealistic. It is understandable that in a real system even if the wheel sensor would provide a faulty signal to ABS system due to malfunctioning, the vehicle would stop under normal braking conditions. But due to feedback loop between the system and environment we get wrong results - making it difficult to evaluate/study the correct system behaviour in presence of faults, at least in an automated manner.

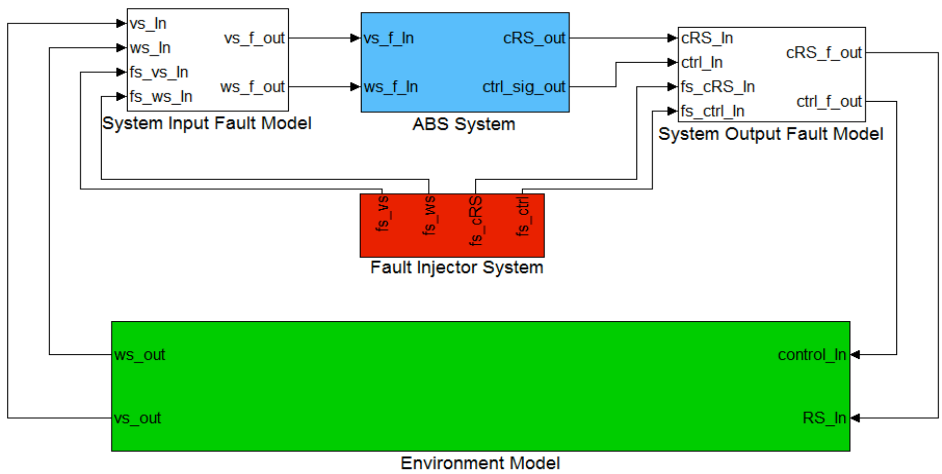

Figure 5: ABS system-environment model representation in Simulink with fault injector setup.

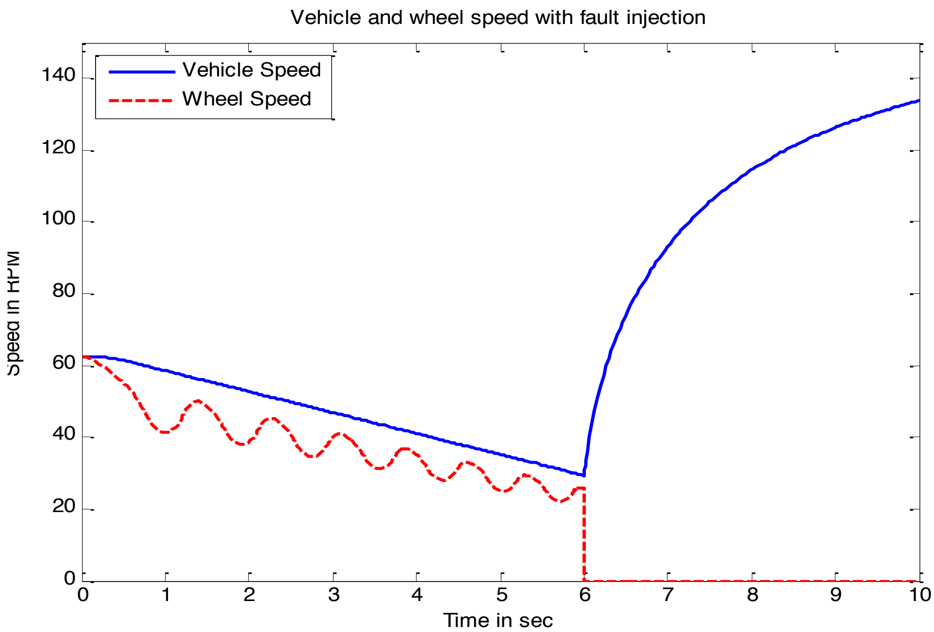

Figure 6: Vehicle and wheel speed with ABS system with fault injection.

One solution under such a situation, which is also widely used in the industry, is to use open loop discrete models instead of closed loop continuous models. The open loop testing is represented in Figure 7.

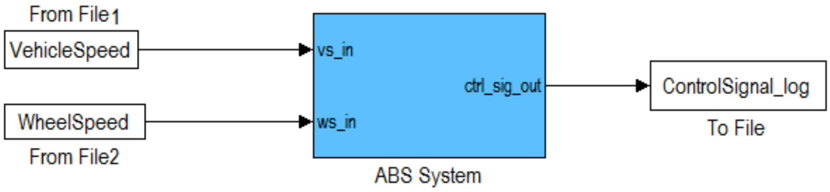

Figure 7: ABS system in open loop discrete model configuration.

Scripts are used to provide recorded data as input, while the output is saved as data file and compared to reference/expected output. The major limitation with such testing is that it's limited by the availability of recorded sensors data as well as need to have the correct output for reference purposes. Thus to test systems under conditions where the input and output data is already not available or if a new functionality is developed or existing system configuration changed, the input/output data may not be available and thus this type of testing unfeasible. Closed loop continuous models do not suffer from these limitations.

5 Fault Bypass Modeling (FBM)

Having described the problem in simple terms we analyze why we observe an unrealistic system response in the ABS system; this will also help us understand the proposed solution framework, which we call **"Fault Bypass Modeling (FBM)"**.

To test the ABS function/system under MIL, we have to simulate the environment. The environment simulates the vehicle dynamics and gives wheel and vehicle speed as input to the ABS system. The ABS generates the control signal, which controls the brake pressure like in a real vehicle. But since one of the environment parameters (in this case: μ or *coefficient of friction between the wheel and the road*) is dependent on the value of relative slip, real time μ value can only be calculated using corresponding relative slip value calculated within the system and provided to environment. The fact that a natural parameter is calculated based on a property of the system introduces a "superficial" loop between system and its environment. This "superficial" loop is only present in the model, as in the real vehicle the value of μ would be naturally determined based on actual relative slip value at each wheel-road surface. This superficial loop is necessary for the system-environment model to be executable, as realistic real time μ values during the simulations are obtained by its virtue, but under fault injection modes this loop becomes the source of unrealistic feedback mechanisms by providing un-natural values of μ.

Thus in such cases, in order to obtain a realistic/corrected system behavior under fault injection conditions, the FBM principle is described as following:

"If a signal injected with faults or its derivative is used to calculate/control any natural environment parameter(s)[1], the part of signal or its derivative which is used to calculate/control the environment parameter(s) should be made fault free to break the unrealistic feedback loop"

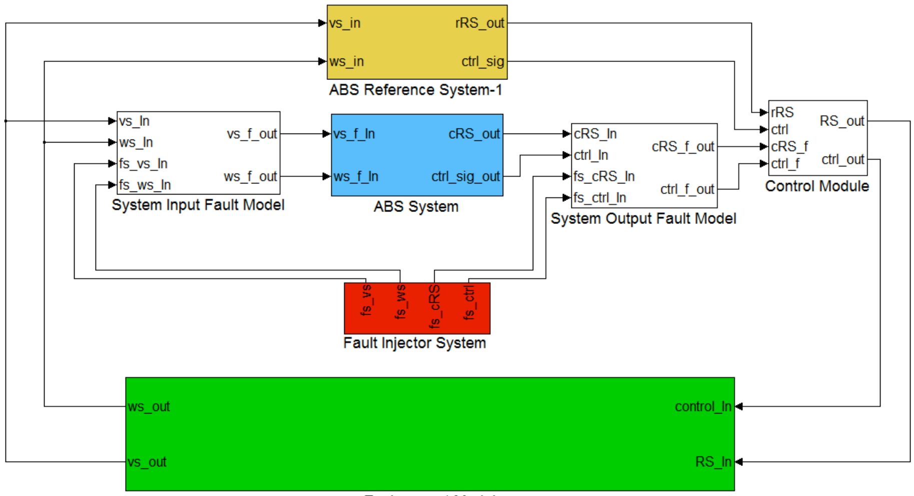

Figure 8: ABS system-environment model representation in Simulink with fault injector setup using FBM.

The FBM setup although very similar to control system setup, the FBM principle is fundamentally different from that of control theory. In control system the controller senses the output of system, compares it to the expected/desired behavior and computes the corrective action based on model of systems response [AM08], whereas FBM principle is used while designing new systems to get the correct system response under fault injection conditions. The desired behavior of system is mostly unknown in case of FBM, while it is a must for using a control system using control theory.

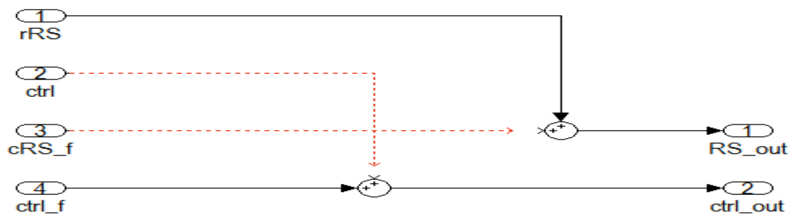

Figure 9: Control module

Use of FBM principle in our case of ABS system is as follows: since the value of μ in on-road conditions is not affected by the calculated value of wheel speed sensors but only by the actual wheel speed, the effect of fault injection on this signal should be bypassed to calculate the real value of relative slip and provided to the environment model for correct value prediction of μ according to actual natural scenario. This is

[1] *Natural Environment Parameter here refers to such a parameter, which is not a property of system but needs correct value from system to define its correct state/value.*

achieved in our model by using a second instance of our system model (*ABS Reference System-1 in* Figure 8), which takes the system inputs that by pass the fault injection. And using the control module (figure 9) real relative slip value are provided to the environment, while the ABS system control value (*ctrl_sig*) that is affected by the injected fault is not bypassed and used to determine the actual system behavior under fault scenario.

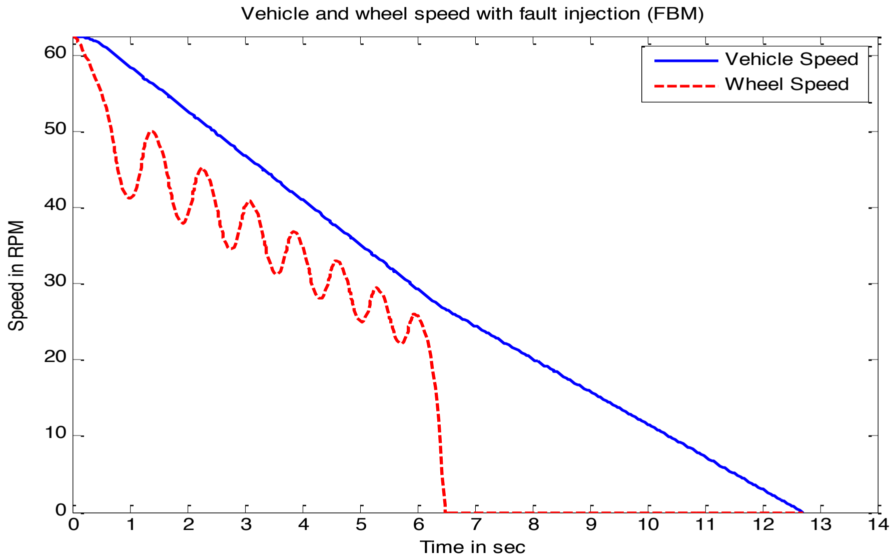

Figure 10: Using FBM shows wheel locking at t=6sec, time of fault injection.

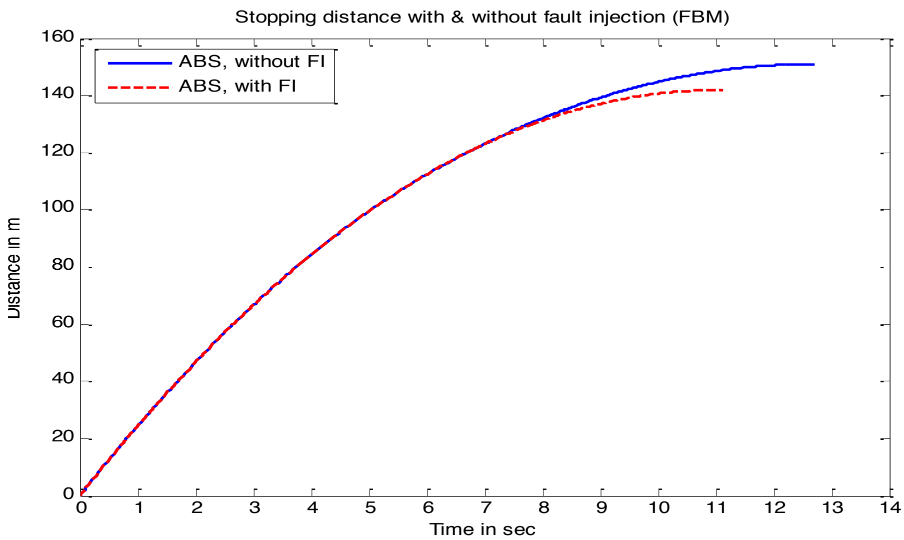

Figure 11: Using FBM shows for this system, the stopping distance would increase by approx. 6m under given fault conditions.

Using the proposed FBM, the system can be simulated with various types of faults to simulate different faulty conditions that can affect the ABS system. Figure 10 shows the system output that matches the tester's expectation on encountering a faulty wheel speed sensor, the wheel gets locked soon after the fault is injected and vehicle stops under normal braking conditions without ABS system functionality. Figure 11 indicates an expected increase in stopping distance by about 6meters and stopping time by about 3sec, if wheel speed senor is failed in the described conditions, such analysis and information availability at design or early development phase is highly valuable to validate as well as to improve the system design and implementation.

The signals that should be by-passed can be easily identified using the FBM principle, i.e. signals, which directly or indirectly influence the value of parameters that in an actual scenario would be naturally determined and not depend on system, should be by-passed when using fault injection. The applicability and generalizability of proposed approach is wide in the context of robust environment modelling. In cases where environment parameters such as coefficient of friction, coefficient of thermal expansion, etc.; where the value of environment parameter depends on the system state, FBM principle can be used to design robust environment models that not only work in normal working conditions, but can also provide realistic outputs under fault injected conditions. FBM principle can be incorporated in the environment model itself or system-environment model can be configured similar to case presented here, to study system behaviour under fault scenarios. The preferential mode of modelling would be incorporate FBM within the environment model itself.

6 Conclusions

Model-based development is already widely adopted in automotive and other sectors, software components in these domains are no longer hand written but usually modeled with MATLAB/Simulink, Statemate or similar tools. These behavioral models offer significant opportunity to verify and validate intended functionality and assess their dependability at an early development stages. Fault injection techniques can be used for dependability evaluation at model levels as they have been used for hardware artifacts. However interdependencies between the system and its environment at model level may cause unrealistic system behavior under fault injection conditions. In this paper we observed this with the help of ABS system example in Simulink. The problem limits the use of fault injection techniques at early stages of development which implies difficulty in testing behavioral models for real life scenarios such as sensor failures, disrupted signals, impact of noise etc.

A framework referred to as Fault Bypass Modeling (FBM) is introduced and evaluated using a case example from automotive domain. Using FBM framework, helps obtain realistic system behavior under fault injection modes. This ensures that correct system behavior/response under real life situations can be studied and analyzed very early in the development cycle. A number of test cases can be designed based on known failure modes and simulated in virtual environment to test models allowing the selection of models with best fault tolerant properties for further development.

Allowing system behavior analysis early in the development cycle will help reducing the late defects; improve the quality of function/software under development and also reduce the development time. Further early system behavior analysis for real life scenarios also imply better communication and understanding between the multidisciplinary development teams. These advantages are especially useful while developing safety critical functions where quality and reliability are of prime importance. Today, embedded software functions development in automotive as well as other domains such as aerospace is increasingly designed using model-based methods. MBD has many advantages, which are well documented.

Historically, the use of fault injection techniques for closed loop model testing has been limited due to difficulties in separating the system failure due to injected fault or system failure due to un-natural feedback as result of system-environment interdependencies. Using FBM resolves this specific problem thus making it possible to use fault injection techniques correctly to test behavioral models for dependability evaluation, robustness and correct functionality. Applying FBM means that models can be tested under closed loop configuration with continuous or discrete signals. Close loop testing allows test engineers to build/model number of test cases based on real scenarios – thus quickly expanding the test space without exponential growth of testing effort. Combining fault injection on the model level with continuous testing, nightly testing or similar techniques, provides possibilities of continuous quality assurance of functionality and shorter feedback loops. Storing test cases with injected faults in libraries and adding new test cases over time decreases the probability of defects slipping to the customers which is unacceptable for safety critical software.

By allowing more efficient testing of models, FBM helps in early defect detection, which not only saves significant cost but also reduced the development time. Using fault injection techniques at models level for dependability evaluation implies that dependability evaluation of given system can be done already at the function development level (compared to system development or integration level as it is done today), more robust models can thus be identified and developed, effectiveness of fault-tolerance mechanism can also be evaluated and shortcomings identified/removed early. Preliminary evaluation of FBM applicability in industrial setting was done at Volvo Cars by conducting two semi-structured interviews with designers/developers with minimum experience of 30 years in model development and testing. The feedback obtained was positive, indicating that it could prove to be useful to evaluate correctly what-if scenarios for complex functions very early in the development process. However it was also pointed out in the initial evaluation that FBM if used should be integral part of environment models as the system developers and testers expects the environment models to be free from superficial loops/ system dependencies.

Future work in this area is expected to use and validate the framework with further case studies on industrial scale models and incorporate the initial feedback by making FBM integral part of the environment model, this would mean that system developers and testers do not have to take into account the FBM, but the development/modeling of environment models for the given system, FBM should be implemented to allow correct model testing using fault injection techniques.

Acknowledgements
The work presented here has been funded by Vinnova and Volvo Cars jointly under the FFI programme (VISEE, Project No: DIARIENR: 2011-04438).

References

[ABB06] O Aerlund, P Bieber, E Boede, M Bozzano, M Bretschneider, C Castel, A Cavallo, M Cifaldi, J Gauthier, A Griffault, et al. ISSAC, a framework for integrated safety analysis of functional, geometrical and human aspects. *Proc. ERTS*, 2006:1-11, 2006.

[ALR01] A. Avizienis, J.C. Laprie, and B. Randell. Fundamental concepts of dependability. *Technical Report Series of Newcastle Upon Tyne Computing Society*, 2001.

[AM08] K. J. Astrom and R. M. Murray. Feedback systems: an introduction for scientists and engineers. Princeton university press, 2008.

[BK06] E. Bringmann and A. Kramer. Model-based testing of automotive systems. In *Software Testing, Verification, and Validation, 2008 1st International Conference on*, 2008, pp. 485-493.

[BOE87] Benjamin Brosgol. DO-178c: the next avionics safety standard. In *ACM SIGAda Ada Letters*, volume 31, pages 5-6. ACM, 2011.

[ESACS] ESACS Consortium et al. Enhanced Safety Assessment for Complex Systems (Project Portal). http://cordis.europa.eu/search/index.cfm?fuse action=proj.documentPJ$_R$CN = 4947941.

[FAG02] M. Fagan. Reviews and inspections. *Software Pioneers--Contributions to Software Engineering,* pages 562-573, 2002.

[HTI97] M. C. Hsueh, T. K. Tsai, and R. K. Iyer. Fault injection techniques and tools. *Computer,* 30(4):75-82, 1997.

[ISO11] ISO 26262:2011 Road vehicles - Functional safety standard, 2011.

[JH05] A. Joshi and Mats P. E. Heimdahl. Model-based safety analysis of simulink models using SCADE design verifier, 2005.

[LBE04] K. Lamberg, M. Beine, M. Eschmann, R. Otterbach, M. Conrad, and I. Fey. Model-based testing of embedded automotive software using MTest. In *SAE World Congress*, 2004.

[MM95] R. Megen and D. B. Meyerhoff. Costs and benefits of early defect detection: experiences from developing client server and host applications. *Software Quality Journal,* 4(4):247-256, 1995.

[MST12] N. Mellegård, M. Staron, and F. Törner. A light-weight defect classification scheme for embedded automotive software and its initial evaluation, 2012.

[SIM12] SimulinkDemo. Modeling an Anti-Lock Braking System. *Copyright 2005-2010 The MathWorks, Inc,* 2012.

[SVET12] R. Svenningsson, J. Vinter, H. Eriksson, and M. Törngren. MODIFI: a MODel-implemented fault injection tool. *Computer Safety, Reliability, and Security,* pages 210-222, 2010.

[VBRE07] J. Vinter, L. Bromander, P. Raistrick, and H. Edler. FISCADE - A Fault Injection Tool for SCADE Models. In *Automotive Electronics, 2007 3rd Institution of Engineering and Technology Conference on*, pages. 1-9, 2007.

[ZAV04] H. Ziade, R. A. Ayoubi, R. Velazco, and others. A survey on fault injection techniques. *Int. Arab J. Inf. Technol.,*1(2):171-186, 2004.