Paper Preprint – To Appear In:
Journal of Systems and Software, 2008

Assessment of High-Integrity Embedded Automotive Control Systems using Hardware in the Loop Simulation

Michael Short and Michael J. Pont
E-mail addresses: mjs61@le.ac.uk, m.pont@le.ac.uk.

Abstract

Sensor-based driver assistance systems often have a safety-related role in modern automotive designs. In this paper we argue that the current generation of "Hardware in the Loop" (HIL) simulators have limitations which restrict the extent to which testing of such systems can be carried out, with the consequence that it is more difficult to make informed decisions regarding the impact of new technologies and control methods on vehicle safety and performance prior to system deployment. In order to begin to address this problem, this paper presents a novel, low-cost and flexible HIL simulator. An overview of the simulator is provided, followed by detailed descriptions of the models that are employed. The effectiveness of the simulator is then illustrated using a case study, in which we examine the performance and safety integrity of eight different designs of a representative distributed embedded control system (a throttle- and brake-by-wire system with adaptive cruise control capability). It is concluded that the proposed HIL simulator provides a highly effective and low-cost test environment for assessing and comparing new automotive control system implementations.

Index Terms: Road vehicle electronics, simulation, distributed control, road vehicle reliability.

1. Introduction

In the modern passenger vehicle, up to 70 electronic control units (ECUs) are connected to a variety of sensors and actuators in order to realize high-level functionality and services (Lean et al. 1999). To reduce cabling costs and improve flexibility, these ECUs are connected to one another via one or more serial communication buses (Lean et al. 1999). With the advent of heavily sensor-based control technologies and driver assistance systems, such as drive-by-wire and automatic collision avoidance, these distributed embedded systems (DESs) will have no mechanical backup and play a crucial role in safety (Iserman et al. 2002).

Prospective designers have guidelines available to assist in the development process; for example the MISRA coding guidelines (MISRA 2004), SAE J2056 (SAE 1993a; SAE 1993b) and general guidelines for good programming practice (Holzmann 2006). Previous research has also concluded that for certain system aspects, partial recommendations can be made; for example the use of time-triggered (TT) communications (Albert 2004) and TT task scheduling (Bate 1998) have both been found to make system behavior more predictable. Even so, many factors are known to influence the reliability of a DES, including the choice of hardware, programming language, communication protocol, and software architecture (MISRA 1994), and there still remains a huge amount of design scope. With an extensive array of options to choose from, there is a pressing need to investigate the implications of different system architectures and their possible impact on safety, reliability and performance.

When developing such complex safety-critical systems, it is often inappropriate, unethical or even impossible to test the system completely within its natural operational environment (Storey 1996; Levenson 1995). In such cases, “hardware-in-the-loop” (HIL) simulation of the system and its environment can allow developers to make an initial assessment of performance without compromising safety. In addition, HIL simulation has been shown to both increase the quality and reduce the time-to-market (and hence development costs) for prototype vehicle systems (e.g. Elims 2000; Li et al. 2002; Kendal & Jones 1999; Hakiwara et al. 2002).

The principle of HIL simulation of an embedded system is illustrated in Figure 1. The embedded system outputs are fed directly to the simulator, where they are sampled and used as input variables. A dynamic simulation model, acting on these input variables, is evaluated (normally in real-time, but this is not always the case). The outputs from the simulation, which are synthesized from the dynamic model(s), are then fed back into the system under test as outputs, thereby closing the control loop. Simulators of this nature are not exclusive to the automotive domain and have been used successfully in a variety of applications, including (for example) verification of new manufacturing machine tool designs (Stoepler 2005), commercial aircraft autopilot system design (Gomez 2001) and numerous military applications (Cole & Jolly 1996).

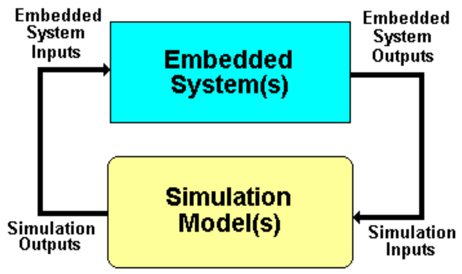


Figure 1: HIL Simulation Principle.

In this paper, we argue that many of the current generation of automotive HIL simulator technologies are unable to test fully the safety and performance requirements of next-generation, safety-critical automotive driver assistance systems. We go on to suggest that there is a need for low-cost, real-time, multi-vehicle simulator systems which can be used to explore the impact of sensor-based control systems, and which also feature the ability to inject faults into the underlying hardware and software under test under free-flowing motorway traffic conditions. We then describe the principle and implementation of such a test facility.

The remainder of this paper is organized as follows. Section 2 provides an overview of the use of HIL simulator and the evaluation of dependability and safety integrity in the automotive domain. Based on this discussion, Section 3 goes on to propose a set of requirements for future simulator environments which (it is argued) would help to make such tools more useful. Section 4 provides an overview of a simulator designed to meet these requirements while Section 5 details the development of models to represent motorway vehicles. Section 6 goes on to develop microscopic driver models for these road users, and presents macroscopic simulation data. Section 7 then discusses an effective mathematical basis for the implementation of fault injection and classification of resulting failure modes. Section 8 then introduces a generic, low-cost HIL simulator framework that has been used for the implementation of the simulation models. In Section 9, we summarize an extended case study, which was performed using the facility to investigate the performance and safety / reliability of eight different versions of a test system, a distributed throttle- and brake-by-wire system with adaptive cruise control. Section 10 presents our conclusions and suggests possible areas of further research.

2. Previous work in this area

In the Introduction, it was argued that many of the current generation of HIL simulators may not allow a full assessment of the next-generation of vehicle control systems. A summary of some of the features of current simulators, along with a discussion of the inherent problems encountered when assessing safety properties of automotive systems, is provided in this section.

2.1 Automotive ECU design

In the automotive sector, HIL testing techniques have been successfully employed for a number of years: Ellims (2000) provides a good summary and general overview of some typical techniques that have been employed. From a starting point in the early 1980s, when low-cost processing power allowed HIL systems to become feasible, many specific companies developed their own closed, proprietary systems tailored to the needs of individual manufacturers and products. With the subsequent emergence of generic, model-based dynamic simulation tools for desktop PCs, such as Matlab® and Simulink®, these have now become the preferred tools for many developers of automotive HIL simulators (Mathworks 2005).

The types of embedded control systems tested using these means typically include ECUs used to run internal combustion engines and to provide anti-lock braking facilities. They typically consist of small-scale simulations of a single vehicle (or part thereof) and its engine. For example, Li et al. (2002) describe the use of a HIL system for the rapid development and validation of a vehicle ABS system using Simulink® and the Real-Time-Workshop® tool. Kendall and Jones (1999) document an almost identical methodology that was used to develop the main body control ECU functionality for a Jaguar vehicle. Hakiwara et al. (2002) demonstrate the use of Matlab®, MATRIXx® and dSpace® in the accelerated design of a vehicle automatic transmission ECU.

As can be observed from these studies, and many others in the literature, these environments and tools allow for much flexibility and have been employed for a host of vehicle systems. However, such simulations are limited to models of a single vehicle, driver, and tire/road interface, and do not include free driving conditions in realistic traffic flows. As such, the impact of sensor-based driver assistance systems cannot be fully explored, as the traffic pattern on roadways is emergent from not only the control systems under investigation but also the behavior of surrounding drivers (Bullock et al. 2004).

For example, the addition of a lead vehicle into a simulation may be used to verify the basic performance of an ACC system. However, an extended 1999 study of a commercial ACC system revealed a potentially dangerous condition that caused distress to the majority of drivers who experienced it (Fancher et al. 1998). It must be noted that this was not a systematic error, as the controller was performing according to its specifications. Since the traffic situation that caused the condition could only arise under free-flowing multi-lane driving conditions, this would never occur in the simplified simulation suggested above; it follows that no amount of testing would uncover this situation.

2.2 Simulation of traffic flow

Although most existing HIL testing facilities do not include such features, the simulation and analysis of motorway and city traffic flows is an in-depth and widely covered subject, with many differing approaches described in the literature (e.g. Yang 1997; Wu et al. 2000; Gipps 1986).

A survey all such methods is beyond the scope of this paper. However, it can be observed that the vast majority of the software tools that do exist to model these phenomenon, such as the Paramics® package (Paramics 1999), are off-line (and hence non real-time) graphical environments, employed in most cases by engineers primarily concerned with traffic signaling and transport planning.

More recently, there has been much research interest in adaptive real-time traffic signaling. However it has been recognized within the community that many of the popular simulation packages (such as Paramics®) do not have the capacity to directly integrate these newer signaling algorithms into their simulation environments; there is also a need to investigate the effects of the actual implementation hardware, which may be distributed over physical distances of several kilometers (Bullock et al. 2004; Courage & Lee 2005). To overcome these problems, researchers have suggested (and implemented) real-time microscopic traffic flow models in dedicated HIL facilities (Bullock et al. 2004; Courage & Lee 2005).

Such solutions have many attractive properties from the perspective of investigating in-vehicle sensor-based control technologies; however two main limitations still remain. The first is that these systems, by definition, concentrate on fixed signaling installations (such as ramp metering for motorway intersections) and the time duration that a single vehicle is actually present within this fixed area is very short; although traffic flow is simulated, it does not concentrate on a single vehicle traveling under free-flowing motorway conditions. The second problem is the time resolution of the simulations; the simulations that have been reported in the literature have update intervals ranging from hundreds of milliseconds to tens of seconds. The required timing resolution for investigating the type of control systems that this paper is concerned with is in the millisecond area.

There are, however, a limited number of large-scale real-time driving simulators with HIL testing capabilities. For example, the AutoSim® driving environment can be modified to allow external hardware to interact with a full-scale simulation of motorway or city driving (Autosim 2005). Although primarily intended for human factor research (it has a vehicle chassis mounted on a Stewart platform) it can - at least in principle - be used for HIL, although how the required on-vehicle sensors would be integrated into the system is not clear.

The Vehicle Hardware-In-the-Loop (VEHIL) system is a natural extension of such a large-scale simulator specifically designed for testing the type of driver assistance systems we are concerned with in this paper (Verburg et al. 2002). Although only partially complete, when finished its large implementation scale requires a test facility the size of a warehouse; a fully instrumented test vehicle is placed on roller bench, and other traffic participants (vehicle bodies mounted on movable bases) are directly controlled by a central computing facility implementing a traffic simulation. The host vehicle is equipped with real sensors, and they thus generate realistic signals for the control hardware.

Despite many advantages, such systems (by their very nature) are extremely costly, and their sheer physical size and complexity mean that dedicated personnel are needed to maintain and operate them. Such systems are beyond the reach of most researchers and engineers.

2.3 Assessment of system performance

From the perspective of functional performance assessment of distributed / embedded systems, a variety of successful techniques have been suggested in the literature. With application to control systems, these include (but are not limited to) quality measures based upon output error (e.g. Franklin et al. 1994; Torngren 1998) and performance monitoring (e.g. Li et al 2003), and more generally quantifications of jitter, delays and latency plus measures of processor and memory utilisation in the system have all been suggested as useful (and sometimes necessary) benchmarks to evaluate (Bate 1998; Pont 2001; Storey 1996; Torngren 1998). All of these techniques are particularly well suited to HIL simulation, as they can be directly integrated into the hardware and software infrastructure.

Performing rigorous assessments of safety and reliability properties of complex, reactive automotive control systems, however, is a wide and ongoing area of research. Many automotive systems are safety-critical in nature, and hence special measures must be taken at all stages of the design process to ensure that the required Safety Integrity Level (SIL) has been achieved. SILs are a safety measurement standard developed by the IEC (Standard 61508): a discussion of how these measurement standards apply to the automotive domain may be found in (MISRA 2001). The SIL of a system depends on the consequences of system failures, which can be determined using risk assessment; a required dangerous failure rate λ_d is then assigned for a system based on this risk. Demonstrating that the dangerous failure rate for a system is at a specific level requires many factors to be taken into consideration; a major element in this process is the determination of reliability measures for each sub-system and component.

The traditional method of validating reliability is through life testing: however for software-based systems designed to have a failure rate less than 10^{-5}, such as those considered in this paper, this form of testing is impractical (on any reasonable timescale) and alternate means must be considered (Butler & Finelli 1993).

Aside from the simplest of automotive systems, most such designs will employ some form of redundancy for tolerance to faults and failures in subsystems to achieve the required levels of safety integrity (Storey 1996). This redundancy can take many forms; static and dynamic redundancy of electronic components and processing elements, communication systems and sensor/actuator subsystems in conjunction with software-based redundancy management algorithms (Iserman 2002; Hammett 2002). In such cases, fault injection is the preferred means for extracting dependability information (e.g. see Arlat et al. 1990). Fault injection enables the estimation of fault coverage parameters for analytical system models; however great care must be taken in selecting appropriate test sequences and determining representative rates of fault occurrence for such experiments.

One interesting area of research has involved the rare events technique (RET). The RET originated in the field of Operations Research as a methodology for speeding up simulations in which certain events of interest occur with extremely low probability: Heidelberger (1995) provides a useful survey of this technique. The primary aim of the RET is to determine the effects of rare events using simulation models such as Markov chains. From the perspective of real-time HIL simulation, the RET seems particularly suited for use in automatically generating representative test sequences (in a relatively short space of time) for the system under investigation. This is because the events of interest (such as sensor failures) happen very rarely.

Despite suggestions (e.g. Hecht & Hecht 2000; Tang et al. 1997) that the RET is particularly suited for integration into the high-integrity software reliability evaluation process, to the authors' knowledge this possibility has yet to be explored in full. In addition, the use of the technique to generate the test sequences that the evaluation is to based on has yet to be considered in the literature, despite its seeming suitability.

3. Proposed system requirements

Section 2 highlighted some of the limitations of existing HIL simulation frameworks. As an alternative to existing designs, we propose the following list of requirements for a next-generation comprehensive, low-cost and open-source HIL simulator for automotive control systems:

- The platform must integrate a high-fidelity simulation of a vehicle under test along with its interactions with other road users in free-flowing traffic conditions, having a variable and temporal resolution suitable for the application under investigation.
- It must be suitable for both the rapid prototyping and safety testing of driver assistance systems implemented using realistic ECU hardware. This will allow the both the operational and safety impacts of new technologies and control methods to be investigated simultaneously.
- The software architecture must be flexible enough to allow (for example) the use of

vehicle code generated from existing tools (such as Simulink®), and the integration of models and code developed from other areas of transportation research. This will allow the simulation to be upgraded as more realistic models are developed and help ensure that existing models are not rendered unusable.

- It must feature an ability to repeat tests under tightly controlled conditions using a form of scripting. This will allow complicated testing programs to be managed and documented accordingly.
- For dependability evaluation purposes, it must allow the controlled injection of a range of representative faults into the underlying hardware and software, based on a failure model of the system under test; this procedure must be as automated as possible, and designed to maximize the statistical coverage of the possible test cases.
- It must be based on a flexible and low-cost hardware platform that can be assembled and maintained by a single person. This will enable small, independent groups of researchers to maintain such a facility.

In the remainder of this paper, we describe the design, implementation and assessment of a prototype HIL simulator which was designed to meet these requirements.

4. Overview of the traffic simulation

In this section we provide an overview of the real-time traffic simulation which was employed in our prototype HIL system.

4.1 Simulation scope

The simulation described in this paper provides a real-time microscopic representation of road users traveling down an infinitely long three-lane motorway (plus hard shoulder) in the UK. Road users are represented over a 2 KM (1.24 mile) section of road, at the centre of which lies the host vehicle (which is controlled by the embedded system under test). The co-ordinates of the simulated road section are implemented as a moving frame of reference that is attached (longitudinally) to the host vehicle (Figure 2). A consequence of this is that the relative velocity between the host and each other vehicle dictates their position in the simulation, as time advances. The hard shoulder lies to the leftmost (slowest) lane in the Figure and is labeled Lane 0 - it is only used for emergency purposes.

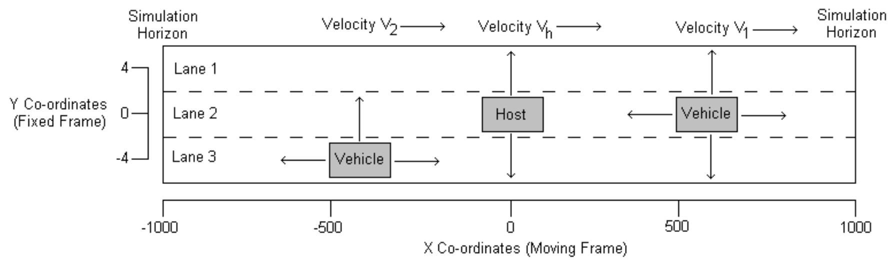


Figure 2: The motorway co-ordinate system.

The simulation has an initialization procedure, where each simulated vehicle is assigned a desired speed and following headway, and given an initial position in the simulation plane. As the simulation evolves, faster vehicles exit the frame of reference to the host vehicle's front, while slower vehicles exit to the rear. When this occurs, new vehicles will be generated and assigned desired speeds and headways, entering either the front or rear of the simulation depending on the current velocity of the host.

The version of the simulation described in this paper is concerned with control of the host vehicle longitudinal velocity only: this is reflected in the granularity of the host vehicle dynamic model. The additional road users consist of heavy goods vehicles (HGVs) and passenger vehicles, while the environmental variables included in the simulation consist of the road gradient, traffic density, weather conditions and prevailing wind/turbulence level.

4.2 Statistical data

In the system described in this paper, the simulation represents is a section of the M1 motorway, to the south of Leicester (UK). The statistical data that is to be used to obtain parameter information for the simulation was taken from several sources; a 1994 video study of 500m of this section of motorway (McDonald et al. 1994) and Governmental statistics reports (DFT 2003; DFT 2004). From these sources, it was found that the overall distribution of desired speeds for passenger vehicles possesses a slightly negatively skewed distribution with a mean centered on the speed limit of 112.7 KPH (70 MPH). For HGVs, many of which are fitted with speed restrictors, it has slightly more pronounced distribution centered at approximately 85.3 KPH (53 MPH). In both cases, since the amount of skew is relatively small, for simplicity the distribution of drivers' desired speeds was taken to be normal. The average percentage of HGVs was taken to be approximately 20%.

The average traffic flow for the M1 in this area was found to be 4,167 vehicles/hour, with an absolute maximum flow of 6,750 recorded. The 1/3 and 2/3 lane occupancy crossover points[1], with 1 being the slowest lane and 3 the fastest, occurred at flows of approximately 2700 and 3350 veh/hr respectively. Sufficient data was not available regarding the 1/2 crossover point, however it could be estimated to occur at a point between 1500 and 2000 veh/hr by extrapolating the existing data. The lane occupancy was approx 30% at the 1/3 crossover point, and 37% at the 2/3 point. The lane change rate was found to vary quite wildly, even for identical traffic flows occurring at different times, with an approximate mean located around 1100 events/Km/hr for flows greater than 2000 veh/hr.

5. Modelling the vehicle dynamics

This section describes the dynamic models that were created to represent the operation of both the host vehicle and the additional road users.

5.1 Host vehicle

The version of the simulation described in this paper was concerned with longitudinal vehicle dynamics only. For this reason, a two-wheel traction model was chosen to represent the host vehicle (Gillespie 1992). A schematic of this model is shown in Figure 3.

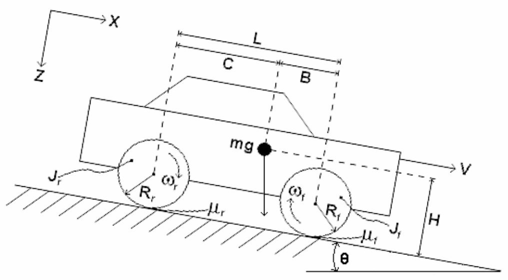


Figure 3: The two-wheel traction model.

The resulting dynamic equations that were created to implement this model were non-linear and of high order. A block diagram outlining the main elements of the vehicle model, and the system variable dependencies between them is shown in Figure 4. A summary of the development of the equations of motion, powertrain model and selection of suitable parameters may be found in (Short et al. 2004a). The so-called Pacejka magic model was

[1] A lane occupancy crossover point is the value of traffic flow at which the observed occupancy in two traffic lanes becomes identical. Hence, the 2/3 lane crossover point is the observed traffic flow, in vehicles/hour, at which the middle lane occupancy equals the fast lane occupancy.

used to capture the dynamics of the tire/road interface (Bakker et al. 1989). In the simulation, the dynamics of the host vehicle were updated every 1 ms.

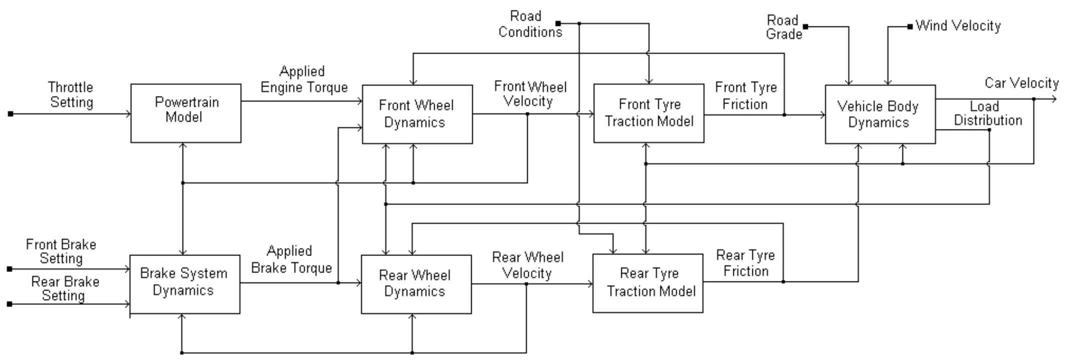


Figure 4: Host vehicle model block diagram.

5.2 Additional road users

Additional road users are represented by a simplified dynamic model, which is given by Equation (1), where x_2 is the vehicle velocity and x_1 the vehicle position.

$$\dot{x}_1 = v_h - x_2$$
$$\dot{x}_2 = \min(\alpha_a, \alpha_{\max})$$
(1)

The host vehicle velocity is v_h, and the vehicle driver provides the acceleration signal α_a. The saturation level α_{max} takes into account the vehicle speed, gradient and road condition to provide realism, as described by Yang (1997). The dynamics of additional road users were updated every 10 ms.

6. Modelling the driver

In order to provide the acceleration signals for the additional road users, and to decide when to change lanes, we implemented microscopic driver models for both longitudinal and lateral vehicle control. Although soft computing techniques (such as fuzzy logic; Wu et al. 2000) have been found to be useful for modelling driver behaviour, we have opted to use a simpler approach in the present simulation, in order to reduce the computational load.

6.1 Longitudinal

In this simulation two different types of longitudinal motion are to be modeled:

- Free Driving – the driver endeavors to achieve and maintain the desired speed.
- Car Following – the driver attempts to maintain a safe distance to a leading vehicle in the same lane.

In both cases, the driver behavior was implemented as a feedback control system acting on the simulation kinetics. The free driving model was implemented as a proportional controller producing an acceleration signal α_{fd}, and the car following model as a proportional-velocity controller producing an acceleration α_{cf}. The actual applied acceleration signal α_a was taken to be the minimum of the two, to provide smooth switching between controllers. A block diagram of the controller is shown in Figure 5. The controllers are updated every 10 ms in the simulation.

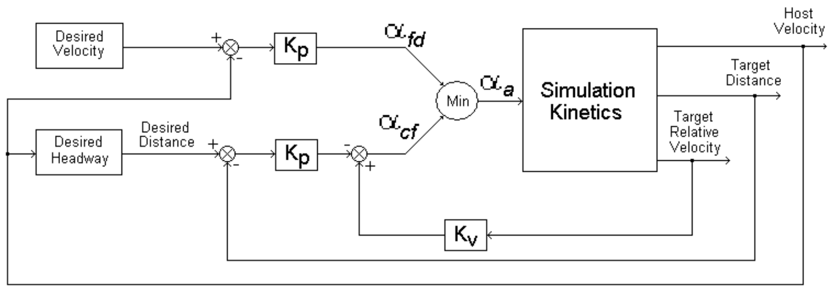


Figure 5: Longitudinal acceleration controller.

6.2 Lateral

Lane changing is a notoriously difficult and complex behavioral process to model. It is best considered in terms of a number of perception thresholds, which consider the risk involved in accepting a gap in neighboring lanes, compared to a benefit factor of some kind when performing the maneuver (McDonald et al. 1994). This simulation is concerned with discretionary lane changes only, since the motorway is considered infinitely long. Three "rules" were considered:

1. Driver's side change (overtaking) – the driver perceives a certain speed benefit in moving to a faster lane.
2. Passenger side change (yield) – the driver perceives a faster car approaching from the rear.
3. Passenger side change (KLEWO) – the driver moves over to the left after overtaking.

KLEWO (Keep Left Except When Overtaking) refers to the driving convention in the UK, but it is clear that this is not always adhered to (McDonald et al. 1994). The lane change model developed for this simulation consists of a driver side change model that implements Rule 1, and a passenger side change model that implements Rule 2 and Rule 3. The models consist of a simple 'gap acceptance' principle, where the gap is defined as the time headway to nearest vehicle in the destination lane both to the front and rear of the considered vehicle. A maneuver is performed when both the minimum gap distances (as given by Gipps 1986) are cleared, and the stimulus is sufficiently strong for the available gaps. A non-linear 'urgency' model was created to relate the threshold for maneuver initiation to the current road conditions. A block diagram of the lane change models is shown in Figure 6. The non-linearity was implemented as a scaled exponential function, with a total of seven parameters defining the lane change model. A full discussion of the nature of the models may be found in (Short et al. 2004b).

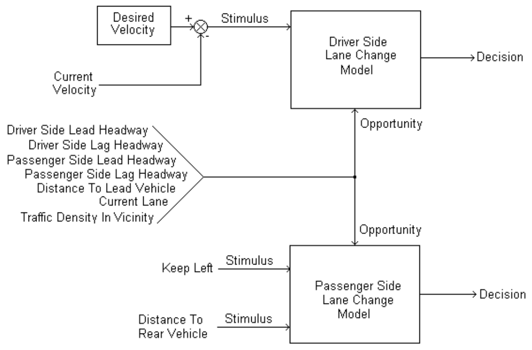


Figure 6: Lane change models.

The actual lane change itself is implemented as a sinusoidal trajectory, described by Equation (2):

$$y_t = y_s + \frac{y_d}{2} + \sin\left(-\pi + 2\pi \frac{t - t_0}{t_1}\right) \times \frac{y_d}{2}$$

(2)

In this equation, y_s is the initial y co-ordinate (at time $t=t_0$), and y_d is the final y co-ordinate (at time $t=t_0+t_1$). The trajectory accommodates a changing velocity v, and the change time t_1 was made inversely proportional to the move urgency, centered on a mean maneuver time of 6.2 seconds (Olsen et al. 2002). The lane change models are updated in the simulation every

10 ms.

6.3 Parameter selection

The longitudinal and lateral driver models both possess calibration parameters that determine how aggressively they will react to the road conditions around them. In the case of the longitudinal model, a study by Fang et al. (2001) have shown that a proportional gain Kp between 0.02 and 0.24 (with a half-normal distribution) generates realistic behaviors. Once a value of *Kp* has been generated for a particular driver, the velocity gain K_v may be obtained from Equation (3).

$$K_v = 0.1839 + 2.5719K_p - 5.5182K_p{}^2$$

(3)

A simple optimization procedure was used to modify the parameters of the lane change model to produce macroscopic flow/lane occupancy behavior close to that observed in practice (McDonald et al. 1994). The host vehicle was replaced during this procedure by the simplified vehicle model given in Section 3.2. A number of simulations were automatically run in a “super loop” with an increased time-step of 1 second, for a duration of 60 simulation minutes. An additional 10 minutes of initial data were discarded to allow any transients to subside. Each test was duplicated five times with different initial seeds in the random number generator, and the averaged results taken. The overall traffic flow and lane occupancy were calculated at each time instant, and averaged over the test duration.

The algorithm that was used simply made small changes to the value of each parameter, from an initial “guesstimate”, in order to reduce the error between the simulation result and the empirical data, at the lane crossover points. The target error was set at 5% of lane occupancy. The resulting parameter set that was selected for use in the simulation produced an overall traffic flow/lane occupancy relationship as shown in Figure 7. The corresponding lane change rate was found have a mean located at 1250 events/Km/hr for flows greater than 2000 veh/hr.

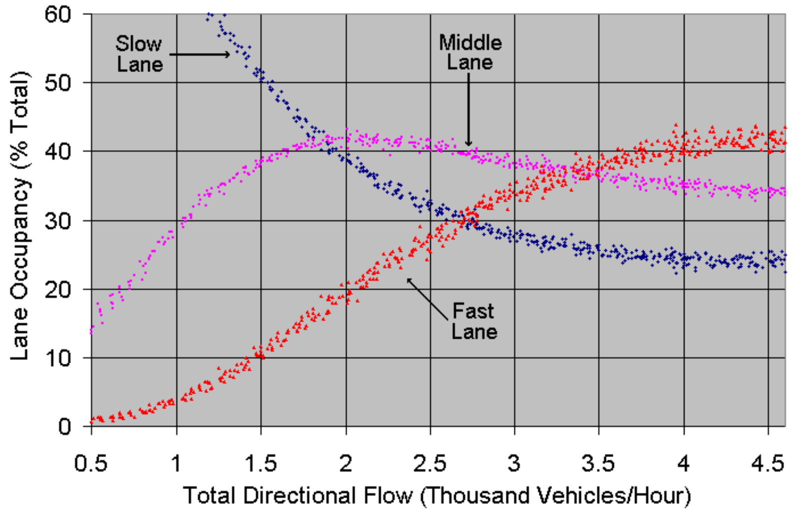


Figure 7: Simulation traffic flow vs. lane occupancy.

Finally, we note that large changes in the observed flow/lane occupancy can be observed by altering the parameters in these driver models; the models may therefore be calibrated to a wide variety of road types, and empirically observed motorway flow regimes.

7. Assessment of safety integrity

In this section we describe a means, based on fault injection and the rare events techniques (RETs: reviewed briefly in Section 2.3), for performing assessments of safety integrity. We begin by describing the mathematical basis for accelerating the occurrence of rare events in the simulation, and how this capability can be added into the simulator.

7.1 Accelerated fault injection

The RET, as applied in the context of high integrity software evaluation, starts from a single basic assumption:

“Well designed software does *not* fail in routine operating conditions”

This assumption is validated by numerous data from sources such as NASA (Lutz & Mikulski 2003; Tang et al. 1997). In addition, when software has been designed using the rigorous techniques that are mandated for high-integrity systems (e.g. see MISRA 2004; Holzmann 2006), and put through an initial test/debug phase, it can be assumed that all high-intensity defects have either been designed out of the system or have been removed during initial testing[2]. It follows that all subsequent failures are caused by non-routine conditions such as abnormal / unexpected input sequences, erroneous computer states and hardware failures (Hecht & Hecht 2000; Lutz & Mikulski 2003; Tang et al. 1997).

The rare events technique allows a qualitative assessment of the system under test by exploiting the discontinuity between 'routine' systems events, and abnormal or 'rare' system events. Although determining the operational profile for sets of system inputs is (in general) extremely difficult, since most rare events (such as abnormal inputs) are generally caused by well-understood physical phenomenon, there is normally a single point which can be selected to distinguish between "normal" and "abnormal" operation (Hecht & Hecht 2000; Lutz & Mikulski 2003). This is highlighted in Figure 8, showing the operational profile of a typical system and highlighting the point P_{rare} where normal operation ends and abnormal operation begins.

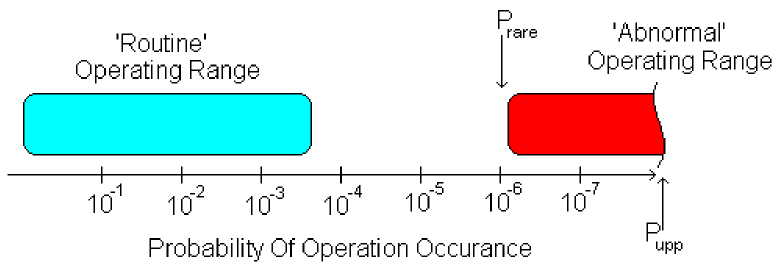


Figure 8: Normal and abnormal operational profiles.

Also indicated in Figure 8 is an upper limit P_{upp}, which is an estimated upper bound for all the rare / critical operations of the system; that is the probability of a critical or rare event Pc is limited as follows:

$$P_{rare} < P_c < P_{upp}$$

(4)

[2] Indeed, where formal techniques have been employed in the software design process, it may be mathematically provable that *all* such defects have been removed. However, even in such cases there is a need to validate that the design specification itself is correct, especially when the specification contains functional techniques for the management and tolerance of unexpected system conditions such as hardware failures.

In particular, to make the assessment of failure rate, the entire operational profile of the system is placed into one of two sets [1] the regular set containing operations or_1, or_2 … or_n or [2] the critical set containing operations oc_1, oc_2 … oc_n. The associated probabilities of occurrence for these operations are then pr_1, pr_2 … pr_n and pc_1, pc_2 … pc_n, satisfying the conditions shown in Equation (5).

$$P_r = \sum_{i=1}^{n} pr_i, \quad P_c = \sum_{i=1}^{n} pc_i, \quad P_r + P_c = 1$$

(5)

where $P_r << P_c$ in order to maintain consistency with the assumptions above (and shown in Figure 8).

In order to exploit the limited test time available for the reliability assessment, the probabilities of occurrence of the operations in the critical test set are adjusted during the period of testing T, giving pc_1', pc_2' … pc_n', by a likelihood ratio α (see Equation 6)

$$pc_i' = pc_i.\alpha$$

(6)

where the likelihood ratio α is chosen as some suitable value for a tractable test, e.g. $1/P_{rare}$.

Suppose that during the testing time T, both routine operations and critical operations are randomly selected (with adjusted critical operation probability). If m failures are observed due to routine operations, and n failures are observed due to critical operations, the failure rate λ may then be calculated as shown in Equation 7.

$$\lambda = \frac{m}{T} + \frac{n}{\alpha T}$$

(7)

However, we assume that the system should not fail for routine inputs: m should therefore be zero (if this is not the case then further, standard testing should be performed to remove these high-intensity bugs). Alternatively, as is the approach suggested in this paper, statistical information regarding these measured failures may also be used to estimate the fault coverage probability for each rare event that has been considered; this may be then further utilised in reliability models to reason about system failure rates. Assuming a sensible choice is made in the selection of α, a suitable statistical model may be calibrated to select test patterns for the HIL system to simulate abnormal environment conditions. Assuming that the fault activation time for the system is much less than the total available test time T, and observing that the average 'mission time' for an automobile is comparatively short (and that a safe state can be entered upon detection and tolerance of a sub-system failure in a very rapid time), then we would suggest that confidence may be gained in the assessment of failure rate by testing for a duration of approximately 10 times the MTTF of the target failure rate, adjusted by the inverse of the likelihood ratio. For example, if the target failure rate of a system is 10^{-7} failures / hour, then with $\alpha = 10^6$ the system must be tested for a minimum of 100 hours under statistical fault injection.

However, in order for this to be implemented, accurate testing of both routine and critical (rare) operations must be achieved: this implies that a form of physical fault injection must be used. Such a mechanism for use in these situations is described in detail elsewhere (Short & Pont 2007b) and will be briefly outlined in Section 8. This technique also requires that system failures can be accurately classified and recorded during long periods of automated testing; a solution to this problem is outlined in Section 7.2.

7.2 Automated performance monitoring

The technique for automated monitoring of system performance and failure mode classification that is to be employed in the facility described in the present paper is based around the concept of model-based control system monitoring. A real-time performance monitor is introduced into the simulator environment that compares the behaviour of the real, hardware/software based system with the desired system behaviour (see Figure 9). The performance monitor that is to be employed is an adapted version of the design presented by Li et al. (Li et al 2003). The monitor is used to detect persistent deviations from the specification that are indicative of system failures (e.g. sluggish / oscillatory performance, out of range or 'stuck at' errors), and also catastrophic failures such as rear-end collisions, or the vehicle becoming stranded or otherwise out of control on the motorway.

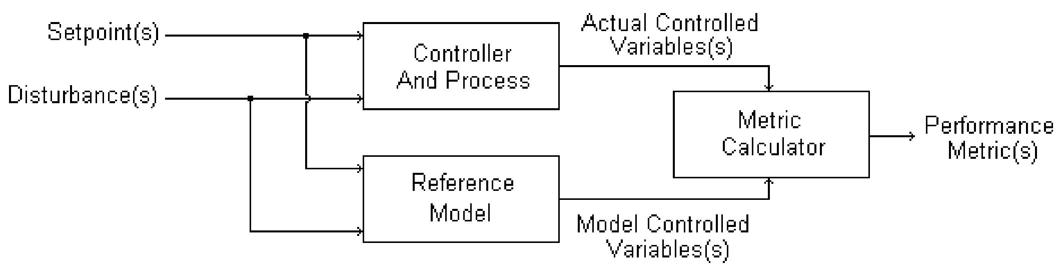


Figure 9: Real-time performance monitoring.

8. Implementing the simulator

In this section we describe the hardware and software framework that we developed in order to implement the simulation models described in the previous sections (and to meet the requirements proposed in Section 3).

8.1 General configuration

The system is based around three standard desktop PCs, without the need for expensive or proprietary interface equipment. Figure 10 shows the overall simulator architecture.

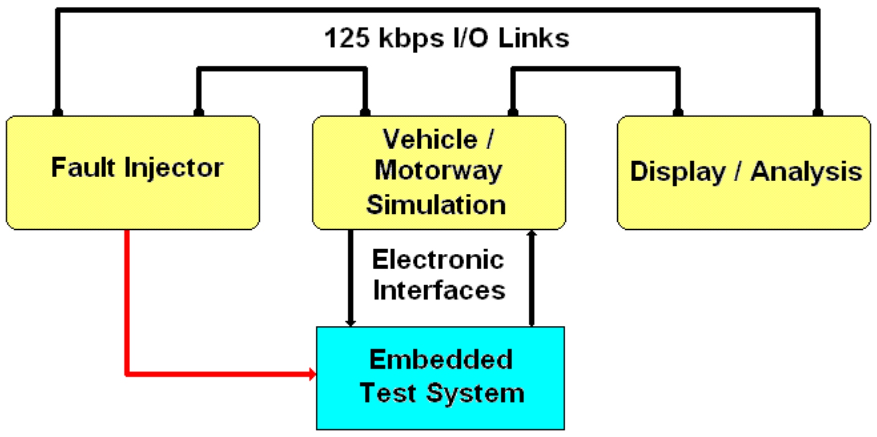
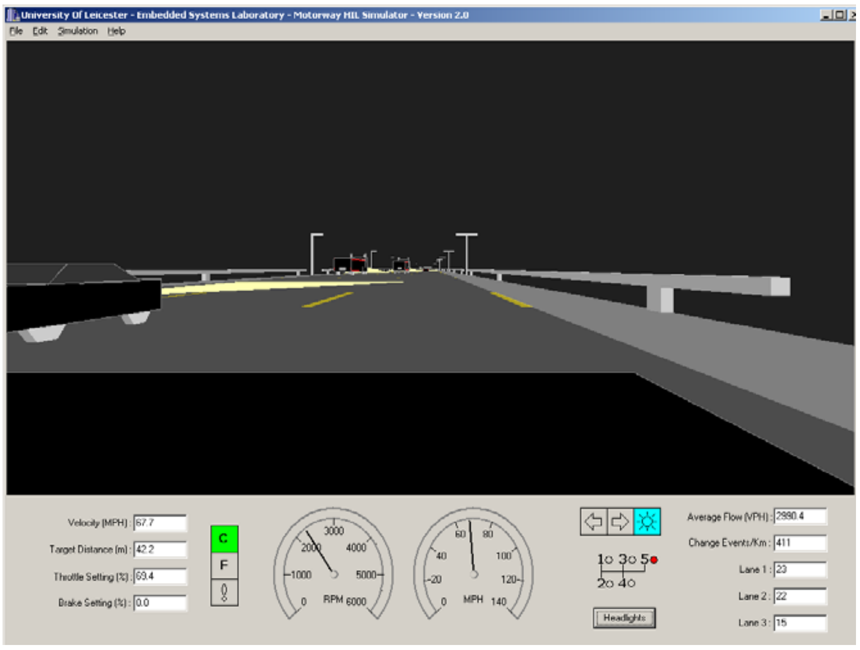


Figure 10: Simulator framework. In the Figure the hard real-time applications run on DOS-based PCs while the soft real-time applications run on a Windows® PC.

A primary PC, running the DOS operating system (or equivalent), is used as a host for the main real-time elements of the software: this is shown as the vehicle / motorway simulation PC in the Figure. Software libraries have been created to perform real-time task execution via a scheduler and provide interfaces to both the parallel and serial ports for digital, analogue and serial I/O. Libraries were also created to implement the performance monitor as discussed in section 7.2. An additional library was created to log simulation variables, system performance and failure modes and to transmit these to a second PC, for display and user interface purposes.

The second (optional) PC runs the Windows® operating system, and is labeled the display / analysis PC in the Figure. If used, this employs a (custom) software library to decode the simulation information transmitted from the primary PC. A further library was also created for this second PC (using OpenGL): this allows 3-D visualization of the simulation information. This interface can be used to shows the simulation evolving in real-time from the perspective of the driver of the host vehicle. A sample screenshot of the user interface is shown in Figure 11. Software was also created to upload and analyze data files generated by the main PC. In both cases, the libraries and analysis tools consist of code and components for the Borland C++ Builder® development IDE (Hollingwood et al. 2001).


University Of Leicester - Embedded Systems Laboratory - Motorway HIL Simulator - Version 2.0
File
Edit
Simulation
Help
C
F
3000
2000
4000
1000
5000
0
RPM
6000
60
80
40
100
20
120
0
MPH
140
1 3 5
2 4
Headlights

Velocity (MPH) : 67.7
Target Distance (m) : 42.2
Throttle Setting (%) : 69.4
Brake Setting (%) : 0.0

Average Flow (VPH) : 2990.4
Change Events/Km : 411
Lane 1 : 23
Lane 2 : 22
Lane 3 : 15

Figure 11: Simulator user interface.

The third (optional) PC again runs the DOS operating system, and is labeled as the fault injection PC in the Figure. Software libraries again are used to perform real-time task execution via a scheduler and provide interfaces to both the parallel and serial ports for digital, analogue and serial I/O. Libraries were also created to implement and manage the rare events technique as discussed in Section 7.1.

8.2 Timing considerations

To support the execution of tasks in the primary and tertiary simulation PCs, we created a DOS-based time-triggered, co-operative scheduler, driven by the system timer (as described in Pont et al. 2003). This type of scheduler was implemented because it is known to be highly stable, is well suited to periodic execution of dynamic models, and possesses low memory and CPU overheads when compared to other forms of scheduler (Pont et al. 2001).

Although the application running on the secondary PC is designed to be time-driven via the underlying data transfer protocol, a Windows®-based scheduler was also created to allow other tasks to be executed in a timely fashion. The timer for this scheduler was provided by a high-priority Deferred Procedure Call (DPC) from a multimedia timer (Microsoft 1998).

8.3 Interface considerations

To keep costs to a minimum, the input/output interface is implemented entirely through the use of standard PC serial and parallel ports. Digital TTL inputs and outputs can be used with ease (a method for creating ‘virtual ports’ is detailed in Pont et al. 2003).

Analogue signals can be interfaced to the simulation via simple interfaces using low-cost, low-power ADC/DAC chips connected to the parallel port (Analogue Devices 1998). Drivers have been created to allow both reading and writing of the digital and analogue signals. Please note that, for an effective simulator, it is usually argued that analogue signals should be sampled and generated with a greater resolution than the system under test, and that the update interval of the dynamic model(s) should be in the region of 5-10 times smaller (Gomez 2001).

For serial digital I/O, drivers have been created for synchronous and asynchronous RS232 communication. The hardware and software fault-injection mechanisms can be implemented via these interfaces.

8.4 Simulator control

In order to allow the simulator to be used for a wide variety of different scenarios, a simple script file interface allows control of system variables. In this case, the files include such variables as traffic density, road gradient, weather conditions (for creating specific scenarios), an option that allows variables to be set randomly by the simulator, and the total test duration. These script files, which have a simple text format, contain both initialization and run-time information. The script files also contain information pertinent to the rare events technique, such as the structure of the system under test (number of nodes, number of communication buses, number and type of sensors etc.) and the representative failure rates for each individual component. They also contain information regarding any additional rare events to be considered (such as transient interferences), their natural rates of occurrence, and the likelihood ratio to be employed during the test run.

8.5 The system configuration employed in this paper

The HIL simulator described in this paper was implemented using three standard desktop PCs. The primary and tertiary real-time PCs had a clock speed of 400 MHz, and the secondary graphical interface PC has a clock speed of 700 MHz. The primary PC scheduler was configured with a 'tick' interval of 1ms, while the tertiary PC was configured with a 'tick' interval of 5ms. The secondary PC scheduler was not required in this case, as the periodic transmission of simulation data was used as the time reference to update the GUI.

In the implementation described in this paper, a total of four parallel ports were used for I/O purposes on the primary PC. Ten analogue output signals were generated from the primary PC. No analogue inputs were required. A single RS232 channel (in addition to the high-speed links to the secondary/tertiary PCs) was required. In total, 45 digital inputs and 11 digital outputs were used.

These basic I/O channels were extended into the simulation by the use of 'virtual' sensor and actuator interfaces, which represent individual sensor/actuator dynamics, measurement accuracy and noise models. In this case parameters for time constants and the accuracy of each virtual sensor/actuator in the simulation were taken from manufacturers' data sheets and survey data (Turner & Austin 2000). The sensor and actuator models were coded into tasks which were updated every 1 ms in the simulation.

On the tertiary PC, a total of two parallel ports were used for I/O related to fault injection. No analogue inputs or outputs were required. A total of 30 digital outputs were made available for connection to relays, allowing hardware faults to be injected (e.g. to simulate broken signals or loss of power). Twelve RS232 channels were made available for the injection of transient software faults into the microcontrollers under test. A non-intrusive and high performance fault injection protocol was created to implement these faults. The protocol itself operates via the PC serial port and on-chip UART of the microcontroller under test, and is based on the well-understood 'bit-flip' or 'upset' model. Such an approach has been shown to be representative of a wide range of transient faults in embedded systems (Arlat et al. 1990). Briefly, the protocol operates as follows. Three control bytes are sent by the PC, invoking a high priority interrupt in the microcontroller. Fault injection is achieved by the use of pointer indirection to achieve the bit flips; further information is provided elsewhere (Short & Pont 2007b).

Implementation of the rare events technique was achieved as follows. Initially, a statistical fault model was calibrated using information provided in the script files: component failure rates were adjusted using the specified likelihood ratio. This statistical fault model was then evaluated very 5 ms, and a pseudo-random number generator was employed to evaluate the health of each component at every iteration of the model (a form of 'Monte Carlo' style simulation). When the model indicated that a particular component failure or rare event had occurred (either transient, permanent or intermittent), this was physically transferred to the equipment under test using fault injection.

We also measured the total PC CPU utilization on the chosen hardware. This was found to be less than 90% on the primary PC, and less than 30% on the tertiary, both with bounded worst-case task execution time, even when in excess of 120 vehicles and drivers were being simulated. On the secondary PC, the CPU utilisation was measured at 60% when implementing the GUI with the maximum number of vehicles being displayed. We therefore suggest that the model code, rare event code and display code we have developed is both portable and suitable for simulating high traffic flow regimes in real-time, on low-cost non-specialized computer hardware.

Given the fact that we wished to employ the test facility to make judgements regarding the safety integrity of the systems under test, it was required to demonstrate that the facility itself exhibits a desired level of integrity. In light of this, a variety of techniques were thus employed to demonstrate that the facility was indeed fit for purpose. Documentary evidence demonstrating its suitability may be found elsewhere (Short & Pont 2007c).

9. Case Study

In this section, to illustrate the use of the HIL simulator described in this paper, we present the results from a representative case study.

9.1 Overview of the case study

The case study presented in this section involves the use of an Adaptive Cruise Control (ACC) system. Specifically, the study explores the behavioural effects and safety integrity impacts of employing different numbers of processor nodes and communication system configurations in eight different implementations of the ACC system. We begin the description of the case study by summarising the principles of ACC.

9.2 ACC description

ACC is a relatively new technological development in the automotive field, and is said to reduce driver fatigue and the rate of auto accidents, whilst increasing fuel efficiency (Stanton et al. 1997). The main system function of ACC is to control the speed of the host vehicle using information about the distance between the subject vehicle and any forward vehicles (using Doppler radar), the motion of the subject vehicle itself and commands from the driver. Based upon this information, the controller sends commands to the vehicle throttle and brakes to either regulate the vehicle speed to a given set value, or maintain a safe distance to any leading vehicles (see Figure 12). The system also sends status information to the driver.

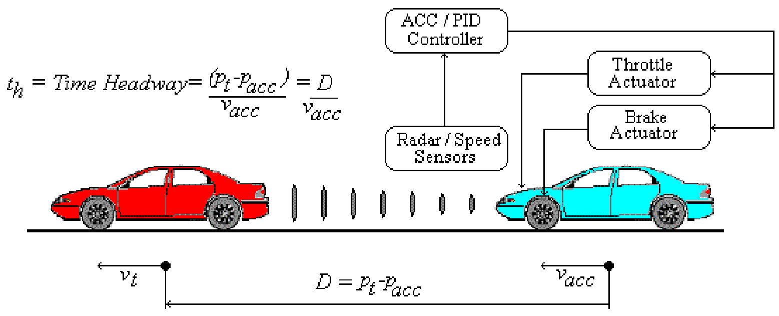


Figure 12: ACC concept.

The system under consideration in this study is a Type 2b ACC system: such a system employs active braking. Vehicle acceleration is limited to 2.0 m/s^2, deceleration to 3.0 m/s^2 in order to comply with ISO standards (ISO 2003). Further details of the nature of the controller itself may be found in (Short et al. 2004c).

When the cruise control system is disengaged, the throttle and brakes are activated via electronic signals from sensors attached to both the brake and throttle pedals. Given the level of risk associated with each element of the system, we classified the required Safety Integrity Level (SIL) for the overall system at SIL 3, with a minimum failure rate of 10^{-7} dangerous failures / hour. The rationale behind this decision was taken directly from MISRA guidelines (MISRA 2001).

9.3 Eight different system implementations

In many application areas, system designers have some flexibility in decisions about the number and location of processor nodes, and in the allocation of software tasks to particular nodes[3]. As different network topologies are considered, it is important that the designer should understand the implications that a particular choice may have on the system's performance and safety. In the study described in this paper, we wished to explore the ways in which the HIL simulator (described in previous sections) could be used to support the process of comparing different system designs.

The particular focus of the study was to explore the links between the chosen microcontroller and communications hardware, and distribution of software tasks in the system, on the resulting control performance and functional safety of the system. To conduct this study, we explored eight different hardware topologies for the ACC system. Since each of the eight different hardware architectures employed various levels or redundancy, we also used the simulator to assess the failure modes of the architectures under intense fault injection. This data then formed the basis for a qualitative estimation of dangerous failure rate.

[3] The reasons for distributing a control system over multiple processors include modularity of design, safety and reduced wiring and harness costs (especially in the automotive sector). However, changes in the number of processing elements - to accommodate various forms of redundancy - generally increases the overall system complexity and associated costs. Of particular concern in this case study is that changes in the number of system nodes is likely to lead - for example - to changes in the time taken to transmit information between system tasks. This can lead to variations in transport delays in the control loop, and thereby degrade the control performance and stability (Franklin et al. 1994).

9.4 Four hardware architectures

In the study presented here we considered systems designs involving four different configurations of microcontroller (shown schematically in Figure 13 to Figure 16). In each case we considered both single- and dual-bus designs: in total, therefore, 8 different designs were assessed and compared in the present study.

In each case, the test system was created using Infineon C167CS microcontrollers (one per node) running at a 20 MHz oscillator frequency (Phytec 2003). The nodes were connected using 500 kbps twisted-pair CAN links. In each Figure, the various inputs and outputs to/from each node and the simulation are shown. From the four Figures, it can be seen that the processor topologies are grouped logically in each of the implementations. The Figures also show how the various software tasks were distributed in each architecture, with each task being allocated a number: this will be explained further in Section 9.5.

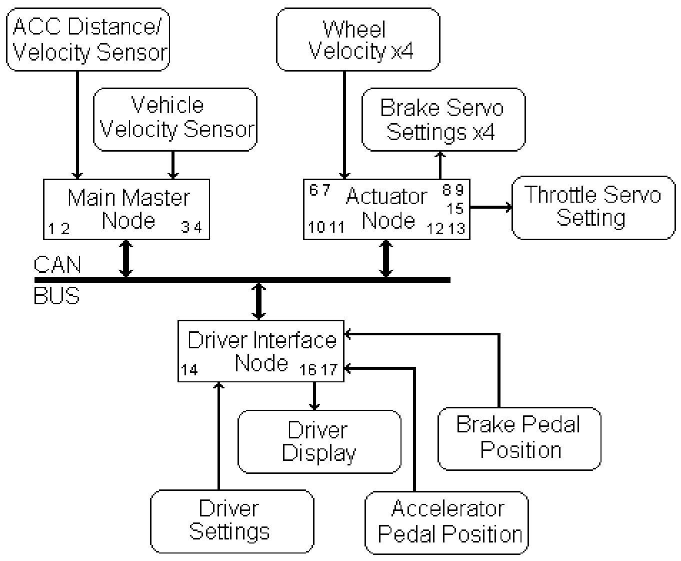


Figure 13: Three node architecture.

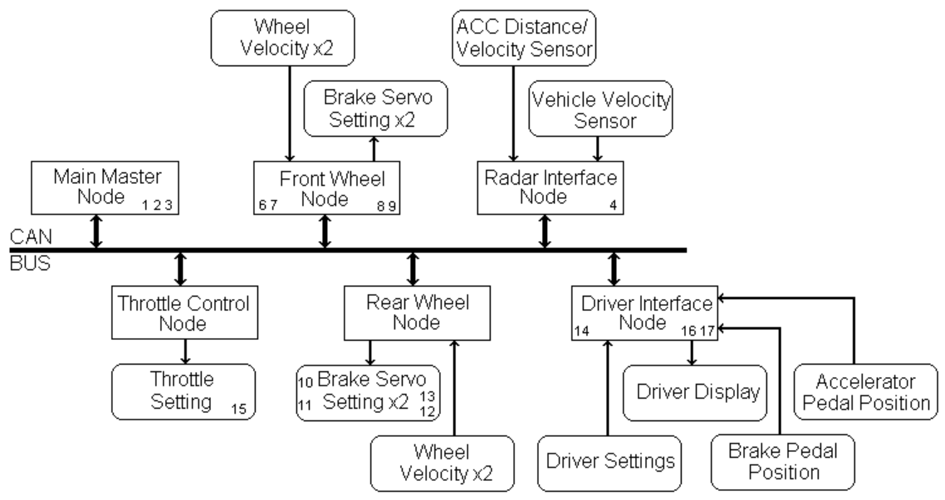


Figure 14: Six node architecture.

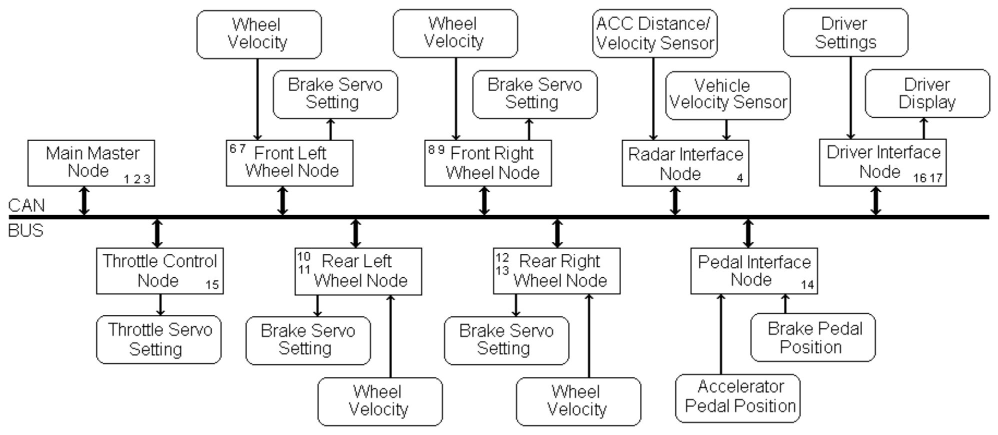


Figure 15: Nine node architecture.

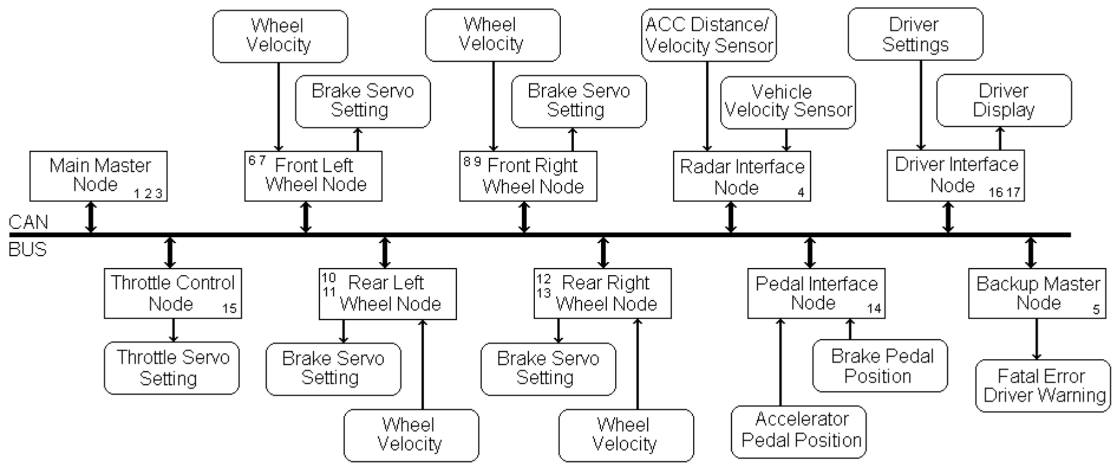


Figure 16: Ten node architecture.

In the most centralised architecture (Figure 13), it can be seen that related control functions are grouped together on the same node: driver interface functions and associated I/O are on one node; the ABS and TCS controllers and associated I/O are on another node; and the ACC system functionality and interfaces are on the Master node. We then progress through varying degrees of distribution until the last architecture, shown in Figure 16. From this Figure it can be seen that we have now partitioned the system due to physical distribution around the vehicle, as opposed to related control functions. Each wheel now has its own local ABS controller; the pedal interface functions have been separated from the remaining driver interface functions located on the instrument cluster, and so on.

We can also see that the increase in the number of nodes has enhanced the inherent fault tolerance of the system. For example, a single processor failure in the braking subsystem of the three node system can have a drastic effect, resulting in complete loss of throttle and brake actuating ability. In all other systems, however, a single brake processor loss will have a reduced impact on braking capacity and no impact on throttle actuation ability. In addition, the presence of the backup Master in the ten node system ensures that in case the main Master fails (silently), the system will not lose its global clock and hence time triggered communication capability (this is explained further in Section 9.5). The backup Master employs dynamic redundancy with fault detection, i.e. it self-activates when the loss of the main Master is detected.

In addition to this, in order to investigate the impact of communication system redundancy on the performance and safety integrity of each design, we also implemented each of these four microcontroller configurations in a dual CAN network configuration. The methodology employed to achieve this was previously described by the authors in (Short & Pont 2007a). The methodology allows the seamless integration of additional CAN channels into a design, and allows for fault-tolerant, time-triggered communications at high levels of clock synchronisation.

9.5 Software and communication architecture

As mentioned in the introduction, it is generally accepted that highly predictable time-triggered behaviour is required for safety-critical systems such as these, for both the network communications (e.g. Albert 2004) and ECU task scheduling (e.g. Bate 1998). For these reasons, in this case study we consider time-triggered designs based around a shared-clock scheduler (Pont 2001), using a Controller Area Network (CAN) as the communication medium (Bosch 1991). Please note that while CAN is often viewed as an event-triggered architecture (Lean et al. 1999), it has been shown that time-triggered behaviour can still be achieved using a "shared-clock" protocol (Pont 2001; Ayavoo et al. 2006). We therefore employed a shared-clock protocol for the present systems, and adapted the protocol to conform to the methodology described in (Short & Pont 2007a) for the dual CAN systems. For each system considered in this case study, we used the same 'tick' interval (5 ms).

Table I shows the software tasks that were created to implement the ACCS. Each task is allocated a number; please note that the same set of (periodic) tasks was executed in each system, with one exception. Specifically, the "Limp_home" task is a safety-related task that is implemented only by a backup Master node: this task is not supported in all four system architectures.

Table I: System task list.

Task Number	Task Name	Period (ms)
1	Vehicle_Control_Update	100
2	Vehicle_Speed_Update	10
3	Vehicle_Status_Update	50
4	ACC_Measurement_Update	10
5	Limp_Home_Update	100
6	FLWheel_Speed_Update	10
7	FLWheel_Brake_Update	10
8	FRWheel_Speed_Update	10
9	FRWheel_Brake_Update	10
10	RLWheel_Speed_Update	10
11	RLWheel_Brake_Update	10
12	RRWheel_Speed_Update	10
13	RRWheel_Brake_Upodate	10
14	Pedal_Detect_Update	10
15	Throttle_Control_Update	10
16	Driver_Display_Update	5
17	Driver_Action_Update	5

In Figure 13 to Figure 16, the distribution of each of these tasks in each of the hardware architectures can be seen. In addition to the high-level software tasks that were created for each implementation, given the critical nature of the system it was required to design techniques for transient fault mitigation into each system. These mechanisms included the use of a 200 ms watchdog timer, duplex duplication of critical data with comparison[4], sanity checks of control signals, and the on-chip exception traps of the C167 processor, listed below:

- Stack overflow;
- Stack underflow;
- Illegal operand;
- Illegal word access;
- Protected instruction fault;
- Illegal bus access.

The unused areas of FLASH memory and RAM in each design were filled with illegal operands to provide added control flow error detection. On activation of any of these traps, a full system reset of the microcontroller was forced. On system boot-up/reset, the microcontroller performed the following software-based self-tests (Soznowski 2006):

- Internal RAM/register/stack validation;
- External RAM validation;
- ROM checksum;
- Peripheral test (e.g. ports, timer).

The overall approach to software fault-tolerance is illustrated in Figure 17. A number of techniques were also employed during the software design process for each system, prior to experimentation. These included the adoption of a structured programming technique, adherence to good programming practice (MISRA 2004; Holzmann 2006) and use of a static code analysis tool[5]. As a final check (prior to the trials described here), each system was required to run failure-free (and free of failed assertions) for a continuous period of 48 hrs using a simulation of a busy motorway,.

[4] The duplex data fields were checked before each use of the variable; any discrepancy forced a hardware reset.
[5] Please see www.splint.org for further details of the tool that was used.

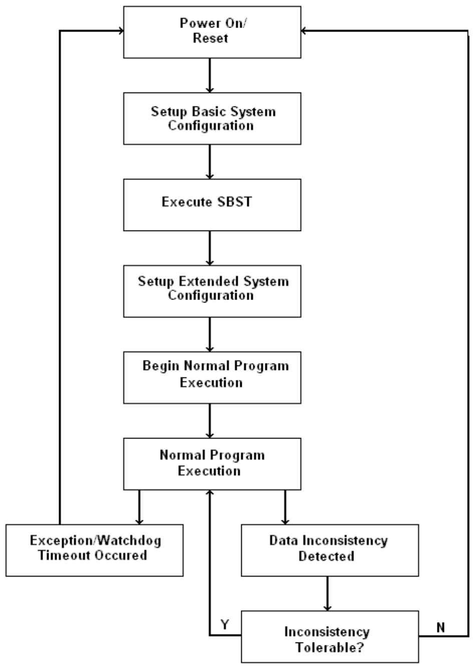

Power On/ Reset
Setup Basic System Configuration
Execute SBST
Setup Extended System Configuration
Begin Normal Program Execution
Normal Program Execution
Exception/Watchdog Timeout Occured
Data Inconsistency Detected
Inconsistency Tolerable?
Y
N

Figure 17: Approach to software fault tolerance.

9.6 Experimental methodology

In the present study, all architectures employed a sample rate of 10 Hz (T_s = 100 ms) for the main ACC control algorithm.

For each system, the time between sampling and control actuation was first measured during fault-free operation. This was achieved as follows. For each iteration of the control loop, a port pin was set high at the start of the sampling task (for a short period of time). Another (initially high) pin was set low at the end of the actuate task, for a short period[6]. The signals from these two pins were then AND-ed (using a 74LS08N), to give a pulse stream as shown in Figure 18. The widths of the resulting pulses was thus representative of the delay between the effective sampling and actuation, and were measured using a National Instruments® data acquisition card 'NI PCI-6035E', used in conjunction with the LabVIEW® 7.1 software package.

The sensor signals that were sampled, in each case, were the vehicle speed, relative distance and relative velocity, with the pin state being set at the start of the first sample being taken.

6 These periods were selected specifically to ensure that the maximum value of delay could be measured.

The control signals that were refreshed during measurement of the actuation tasks were the rear right brake control signal, and the throttle control signal. In each case, the pin state was set upon completion of the relevant task.

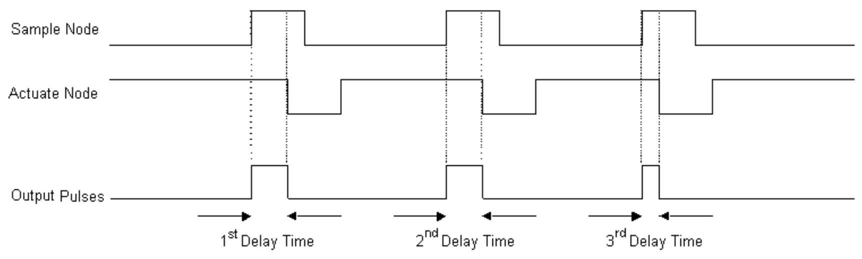


Figure 18: Method used to measure the delay times in each system.

The measures of sample-actuation intervals provide us with a useful indication of the impact that each architecture has on basic system performance. However, we were particularly interested in understanding how these basic timing measures would alter the control performance of the system. To measure the performance of each system, we used the IAE (Integral of Absolute Error) measure (Franklin et al. 1994). The IAE criterion is defined in Equation 8, with *T* equal to the test duration[7].

$$IAE = \int_0^T |e(t)|\, dt$$

(8)

Firstly, we recorded the IAE performance for a 300-second period whilst each system was performing velocity control. During this period, the 'driver' increases and decreases the setpoint from the initial value of 80.45 KPH (50 MPH) several times, via the buttons on the instrument cluster. Secondly, the IAE performance was recorded for each system during a 300-second period when performing distance control. Each distance test was performed whilst following a lead vehicle, initially at 96.54 KPH (60 MPH). The lead vehicle makes a variety of manoeuvres during the test period, and the ACC vehicle attempts to maintain a

[7] Please note that although this measure is dependant on the test length *T*, in this study all systems were tested for an identical duration of time under identical testing conditions; hence no further normalisation of the subsequent results was required.

distance corresponding to the desired time headway of 1.5s. The measures were accurately recorded on-line by the simulation software.

We also used the facility to perform intensive fault injection into each system. The experimental procedure for this was as follows. Each system was first run for a continuous period of 100 hrs, during which time random hardware failures were injected into system, using the rare events technique. Each individual test run during this period was started from random initial conditions, and all previous injected faults were cleared. Each microcontroller was also issued with a full system reset prior to the commencement of a test run.

During each test run, the driver attempted to accelerate the vehicle to a speed of 112.65 KPH (70 MPH). At this point the driver activates the ACC system. Following this, the driver then simply changes lanes and allows the ACC system to control the vehicle longitudinal motion; until such point that the ACC system disengages. The ACC system will automatically disengage if it detects that at the maximum deceleration allowed by ISO specifications, a preceding vehicle will come closer than 15 m. At this point the driver will take over, and again attempt to achieve the cruise speed and activate the ACC when the road becomes clear. The ACC system will also disengage if a fault is detected, and issue a warning to the driver. At this point the driver will attempt to manoeuvre the vehicle into a safe state, as described below.

Each test run could end in one of two ways; either with the vehicle entering a safe state, or a dangerous failure occurring. A safe state could be reached if (and only if) the driver of the host vehicle brought the vehicle to a controlled rest on the hard shoulder of the motorway, in response to the system under test issuing a warning signal to indicate that integrity has been compromised. A dangerous failure was classified as a either a vehicle collision, or the vehicle becoming stranded (or otherwise totally out of control) for a given time period (30 seconds). The likelihood ratio during this phase of testing was set at $\alpha = 10^6$.

Following this, a further 100 hrs of testing was then employed. The overall test methodology during this period was identical to that outlined above, with the exception that each system was this time exposed to intense transient disturbances (as opposed to hardware failures). A transient failure was defined as a period between 5 – 500 milliseconds during which time the system under test was subjected to a series of between 1 – 100 instantaneous bit flips. This was chosen to simulate a variety of disturbances, equivalent to a Single Event Upset (SEU), a nearby lighting strike or a persistent period of electromagnetic upset. Each node was assumed to be equally probable of being subjected to a disturbance during the period of the transient. Bit flips in the C167 internal RAM (IRAM) areas can corrupt the system stack, registers, special function registers (SFRs) and program counter, whilst bit flips in the external RAM (XRAM) areas can corrupt the user stack and also the task data areas. Each fault injected in this study flipped a random bit in a random memory address location from a 4.5 KB area of IRAM or a 4.5 KB area of XRAM; thus implementing a wide variety of data, control flow and CPU / peripheral configuration errors. The likelihood ratio during this phase of testing was set at $\alpha = 10^7$.

The representative failure rates employed during this period of experimentation are shown in Table II, with values derived from representative sources (Short et al. 2007d; Short & Pont 2007a). The statistical nature of the fault injection that was employed ensured that a huge number of different faults were injected with the vehicle in a variety of different representative road conditions.

Table II: Component and event failure rates.

Component / Event	Type	Failure Rate (x 10^{-6})
C167 CPU	Permanent	0.76
C167 RAM	Permanent	0.25
C167 ROM	Permanent	0.16
CAN Link	Permanent	1
CAN Bus Section	Permanent	1
Transient Disturbance	Intermittent	1

9.7 Results

The results to be described in this section are split into two groups; velocity/distance control performance, and sample-actuate timings and safety integrity impacts. We begin by describing the behaviour of each system during the control experiments.

a) Control and system timing

The velocity control profile that was used for each experiment is given in Figure 19. The response shown is that of the single-bus 10-node system. The distance control profile that was used for each experiment is given in Figure 20. Again, the response shown is for the 10-node single-bus system.

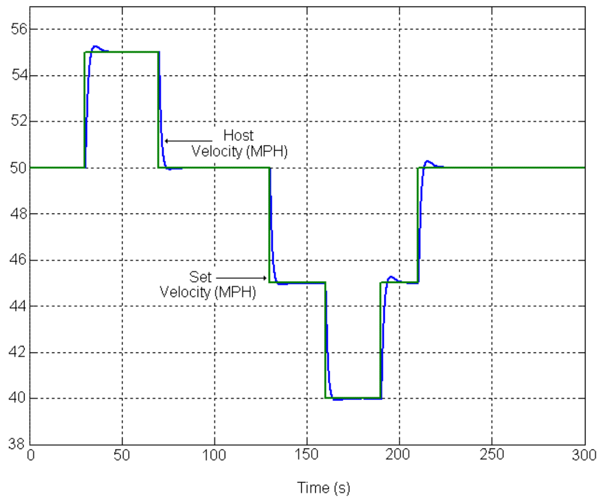


Figure 19: Velocity control profile.

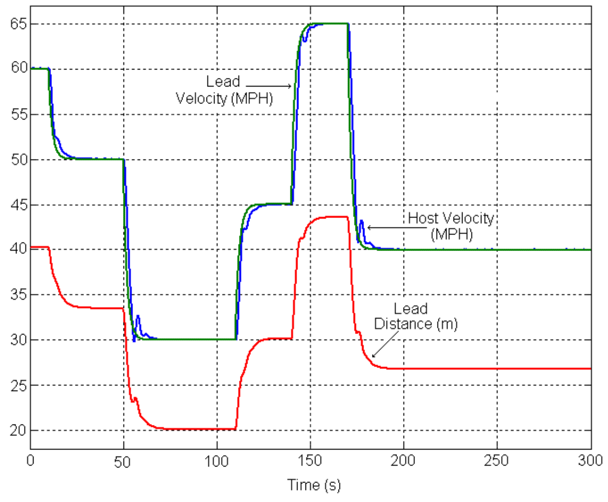


Figure 20: Distance control profile.

The actual values of both velocity (V) and distance (D) IAE recorded, along with the measurements of the effective sample-actuate delays of each system during each experiment, are summarized in Table III. The Table shows the average values that were recorded (each experiment was repeated five times).

Table III: Summary of system performance.

System	V IAE (MPH.s)	D IAE (m.s)	Throttle Delay (ms)	Brake Delay (ms)
3 Node Single	54.06	69.08	10.59	5.96
6 Node Single	55.44	90.23	20.06	25.81
9 Node Single	67.11	105.41	40.43	35.72
10 Node Single	73.65	111.47	60.75	60.50
3 Node Dual	54.11	69.06	10.61	5.98
6 Node Dual	55.50	90.20	20.02	25.84
9 Node Dual	67.03	105.50	40.43	35.74
10 Node Dual	73.66	111.41	60.74	60.54

b) Fault injection

In this section we summarize the results that were recorded during the fault-injection phases of the system testing. Detailed results of the hardware and software fault injection are given in tabular form in Appendix A. For each system, we list the total number of test runs, the total number of faults that were injected, the breakdown of these faults, and the classification of each test run into either a safe state or dangerous failure. Results are summarized in Table IV, were we quote the probability that a hardware fault (P_h) or software transient fault (P_s) will result in a dangerous failure in each system. We also show the equivalent (total) hardware (λ_h) and software transient failure rate (λ_s) for each system, calculated from the representative failure rates that were given in Table II. This then allows the estimation of total dangerous failure rate for each system λ_d, using Equation 9; this failure rate was used to classify the SIL that each system lies in, as shown in Table IV.

$$\lambda_d = (\lambda_h \times P_h) + (\lambda_s \times P_s)$$

(9)

Table IV: Summary of fault injection results.

System	λ_h	P_h	λ_s	P_s	λ_d	SIL
3 Node Single	0.751×10^{-5}	0.9825	1.00×10^{-6}	0.0247	0.740×10^{-5}	SIL1
6 Node Single	0.140×10^{-4}	0.3727	1.00×10^{-6}	0.0300	0.526×10^{-5}	SIL1
9 Node Single	0.205×10^{-4}	0.1539	1.00×10^{-6}	0.0209	0.318×10^{-5}	SIL1
10 Node Single	0.227×10^{-4}	0.0402	1.00×10^{-6}	0.0480	0.960×10^{-6}	SIL2
3 Node Dual	0.115×10^{-4}	0.2870	1.00×10^{-6}	0.0125	0.332×10^{-5}	SIL1
6 Node Dual	0.210×10^{-4}	0.1253	1.00×10^{-6}	0.0179	0.265×10^{-5}	SIL1

9 Node Dual	0.305×10^{-4}	0.0429	1.00×10^{-6}	0.0166	0.133×10^{-5}	SIL1
10 Node Dual	0.337×10^{-4}	0.0000	1.00×10^{-6}	0.0451	0.451×10^{-7}	SIL3

As an example of a dangerous failure, Figure 21 shows the effects of a master-node failure in the nine-node single-bus system whilst following a lead vehicle. Due to the absence of a back-up Master node, it can be seen that a collision occurs with a lead vehicle at 61.3 KPH (38.1 MPH). By contrast, Figure 22 shows an identical situation in which a back-up Master is employed (in the 10-node system). From the Figure, it can be seen that the system recovers with enough time to decelerate the host vehicle and prevent a crash: the driver, alerted to the fault by the warning signalled by the back-up node, then brings the vehicle to a safe state on the hard shoulder: in this case, system integrity is maintained.

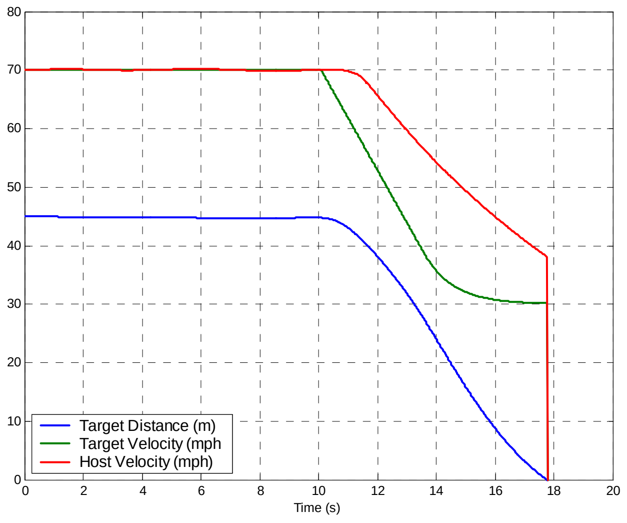


Figure 21: Effect of loss of Master node (without back-up Master).

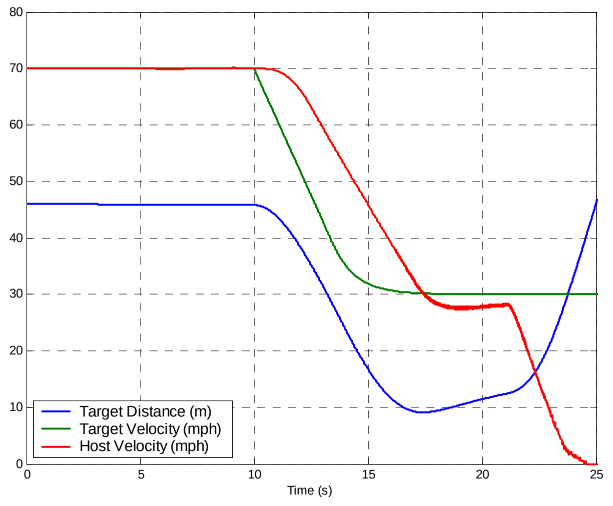


Figure 22: Effect of loss of Master node (with back-up Master).

c) Discussion

It can be noted from Table III that for both the single-bus and dual-bus systems, increasing the number of nodes in the system decreases the overall control performance. It can also be seen from this Table that a major factor in this degradation is the increase in the effective sample-actuate delay for each system (as the number of nodes increases, the effective TDMA cycle time, and hence communication delays, also increases). Despite these performance variations, all the system designs met their real-time specification in this study. (If necessary, the performance of the multi-node system could have been improved using some simple protocol modifications: see Short et al. 2007d; Ayavoo et al. 2007).

From these results, it is also apparent that the addition of the second (redundant) communication channel has had no measurable effect on the system performance. This can be mainly attributed to the fact that the methodology used to implement this redundant channel has an almost negligible impact of the level of clock jitter that may be achieved using the shared clock protocol (Short & Pont 2007a).

Considering next the fault-injection results, it can be seen that adding extra nodes into a system decreases the dangerous failure rate λ_d, by reducing the probability P_h that a hardware failure will result in a dangerous failure. However, the overall probability of a hardware failure λ_h increases proportionally to this increase of processors and may in some cases need to be taken into consideration. In addition, it can be seen that the addition of redundant processors also magnifies the severity of transient disturbances, with a large increase in the number of transient bit-flips; this can be attributed to an increases in the memory usage and computation requirements of the overall system, increasing its susceptibility. In general this has little (if any) effect on the probability P_s that a transient disturbance will lead to a dangerous failure when the additional processors are ‘Slave’ processors. However, in both single and dual bus systems, the addition of a backup Master processor has a detrimental impact on this probability: we suggest that this is because the operation of the Master and backup-Master nodes is tightly synchronised and relatively complex, and therefore more susceptible to short-term disturbances. It is important to stress, however, that this effect is more than compensated for by the increase in hardware fault-tolerance that is gained by the addition of the backup Master node; this is reflected in the reduction in the probability P_h that a hardware failure will result in a dangerous failure in the 10 node systems.

Considering next the impact of the redundant bus arrangement, it can clearly be seen that this a similar beneficial effect on the dangerous failure rate λ_d, again by reducing the probability P_h that a hardware failure will result in a dangerous failure. Again, the overall probability of a hardware failure λ_h increases due to the increased number of hardware components in the system.

To summarise these results, it can be seen that the basic shared-clock mechanism, when complimented with the transient fault detection mechanisms described in Section 9.5, provides a level of reliability that may allow a system to reach the SIL 1 level. With the addition of backup Master and Slave devices, this further increases the effective SIL to 2. Finally, with the addition of a redundant communication bus, a level of reliability commensurate with SIL 3 may be achieved. In this case-study, we conclude that although each system met its functional performance specification, only the 10-node, dual bus system had the required level of reliability and fault tolerance to meet its functional safety requirements when subjected to the faults and failures considered in this study.

10. Conclusions and further work

In this paper, the development of a novel, low-cost and flexible real-time HIL simulator for use in the development and assessment of automotive embedded control systems has been described. We have detailed how this test facility has been configured to represent a road vehicle travelling down a three-lane motorway under free flowing traffic conditions. We have presented macroscopic simulation data of the traffic flows which suggest that the test facility provides a realistic representation of a particular section of motorway in the UK. We have described how fault injection may be incorporated into the simulator, and have described how the resulting failure modes may be automatically classified. Through the use of a case study we have illustrated the use of the simulator, and shown that it is an effective means of examining both the functional behaviour and the functional safety and reliability of different possible designs for automotive control systems.

From the results obtained in the specific case study described here, it can be seen that the use of the HIL simulator has allowed us to rigorously assess the impact of adopting different hardware configurations on the control performance and safety integrity of an ACC system. Since safety arguments in practice are as distinct as the application areas they are employed in, it is difficult to provide generalised results regarding safety concepts. However we believe that this facility described in this paper has allowed us to provide recommendations regarding system reliability and fault tolerance (which are closely related to safety) to prospective designers of automotive systems.

It should be noted that the work described in this paper has been limited to discussions of vehicular longitudinal control. With increasing interest in concepts such as steer-by-wire and automatic lane keeping, there is a need to extend this analysis to consider to more complex situations such as lateral vehicle control and vision-based systems. We suggest that the basic simulator framework presented in this paper may provide a basis for such future work, giving system designers the means to efficiently and rapidly extract a reliable source of empirical evidence related to a wide range of embedded-system applications.

Acknowledgements

The work described in this paper was supported by the Leverhulme Trust (Grant F/00 212/D). An early description of the test facility described in this paper was presented at the 8th International IEEE Conference on Intelligent Transportation Systems in Vienna, September 2005 (Short & Pont 2005). Initial versions of the test results discussed in Section 9.7 were presented at the SAE World Congress 2006 (Short et al. 2007d).

References

Albert, A., 2004. Comparison of event-triggered and time-triggered concepts with regard to distributed control systems, in *Proceedings of Embedded World*, Nurnberg, Germany, 17-19 Feb, pp. 235-252.

Analogue Devices, 1998. AD7394 Data Sheet, Analogue Devices, Norwood, MA, USA.

Arlat, J., Aguera, M., Amat, L., Crouzet, Y., Fabre, J.-C., Laprie, J.-C., Martins, E. and Powell, D., 1990. Fault Injection for Dependability Validation—A Methodology and Some Applications. IEEE Trans. Software Eng., Vol. 16, No. 2, pp. 166-182.

Arlat, J., Costas, A., Crouzet, Y., Laprie, J-C and Powell, D., 1993. Fault Injection and Dependability Evaluation of Fault-Tolerant Systems. IEEE Trans. Computers, Vol. 42, No. 8, pp. 913-923.

AutoSim, 2005. Driving Research Simulators, http://www.autosim.no/prod_s_research.html, [accessed July 2007].

Ayavoo, D., Pont, M.J., Short, M. and Parker, S., 2007. Two novel shared-clock scheduling algorithms for use with CAN-based distributed systems, Microprocessors and Microsystems, Vol. 31, No. 5, pp. 326-334.

Bakker, E., Pacejka H. and Lidner, L., 1989. A new tire model with application in vehicle dynamic studies, SAE Paper No. 890087, pp. 101-113.

Bate, I.J., 1998. Scheduling and timing analysis for safety critical real-time systems, PhD. dissertation, Department of Computer Science, University of York, UK.

Bate, I.J., 2000. Introduction to scheduling and timing analysis, in The Use of Ada in Real-Time Systems, IEE Conference Publication 00/034, 6th April 2000.

Bosch, 1991. CAN Specification Version 2.0, Robert Bosch GmbH.

Bullock, D., Johnson, B., Wells, R.B., Kyte M. and Li, Z., 2004. Hardware-in-the-loop simulation, Transportation Research Part C, Vol. 12, pp. 73-89, 2004.

Butler, R.W. and Finelli, G.B., 1993. The infeasibility of Quantifying the Reliability of Life-Critical Real-Time Software, IEEE Transactions on Software Engineering, Vol. 19, No. 1, pp. 3-12.

Cole Jr, J.S. and Jolly, A.C., 1996. Hardware-in-the-loop simulation at the U.S. Army Missile Command, in Proc. SPIE Vol. 2741: Technologies for Synthetic Environments: Hardware-in-the-Loop Testing, pp. 14-19.

Courage K. and Lee, S., 2005. Hardware in the loop Traffic Simulation: Final Report, University of Florida, Report No. UFTRC-965-1.

DFT, 2003. Road Traffic Statistics: 2003, Transport Statistics Bulletin SB(03)26, The Department For Transport, UK, July 2003.

DFT, 2004. Vehicle Speeds In Great Britain: 2003, Transport Statistics Bulletin SB(04)29, The Department For Transport, May 2004.

dSpace, 2005. dSpace Simulator: Hardware Versions, WWW Page: http://www.dspace.de/ww/en/ltd/home/products/systems/ecutest/hardware_concepts.cfm, [accessed July 2007].

Ellims, M., 2000. Hardware in the Loop Testing, in Proc. IMECHE Symposium IEE Control 2000, Cambridge, UK, September 2000.

Fancher, P., Ervin, R. Sayer, J., Hagan, M., Bogard, S. Bareket, Z., Mefford, M., and Haugen, J., 1998. Intelligent Cruise Control Field Operational Test: Final Report, University of Michigan, Ann Arbor, Transportation Research Institute, UMTRI-98-17, July 1998.

Fang, X., Pham H.A. and Kobayashi, M., 2001. PD controller for car following models based on real data, in 1st Human-Centered Transportation Simulation Conference, Iowa City, Iowa, November 4-7, 2001.

Franklin, G.F., Powell, J.D. and Emani-Naeini, A., 1994. Feedback Control Of Dynamic Systems, Addison-Wesley, Reading, Massachusetts, third edition.

Gillespie, T., 1992. Fundamentals of Vehicle Dynamics, Society of Automotive Engineers (SAE) Press Inc., 1992.

Gipps, P.G., 1986. A model for the structure of lane-changing decisions, Transportation Research, Vol. 20B, No. 5, pp. 403-414.

Gomez, M., 2001. Hardware-in-the-loop simulation, in Embedded Systems Programming, Vol. 14, No. 13, December 2001.

Hakiwara, K., Terayama, S., Takeda, Y., Yoda K. and Suzuki, S., 2002. Development of automatic transmission control system using hardware-in-the-loop simulation system, JSAE Review, Vol. 23, pp. 55-59.

Hammett, R., 2002. Design by Extrapolation: An Evaluation of Fault Tolerant Avionics, IEEE AESS Systems Magazine, April 2002.

Hecht, M. and Hecht, H., 2000. Use of Importance Sampling and Related Techniques to Measure Very High Reliability Software, in Proc. IEEE Aerospace 2000 Conference, Big Sky, MT, March, 2000.

Heidelberger, P., 1995. Fast Simulation of Rare Events in Queuing and Reliability Models. ACM Trans. Modelling and Computer Simulation, Vol. 5, No. 1, pp. 43-85.

Hollingwood, J., Butterfield, D., Swart B. and Allsop, J., 2001. C++ Builder™ 5 developer’s guide, Sams Publishing.

Holzmann, G.J., 2006. The Power of Ten: Rules for Developing Safety Critical Code, IEEE Computer, Vol. 39, No. 6, pp. 93-95, June 2006.

Iserman, R., Schwarz R. and S. Stoltz, 2002. Fault-tolerant drive-by-wire Systems, IEEE Control Systems Magazine, Vol. 22, No. 5, pp. 64-81.

ISO, 2003. Adaptive Cruise Control systems – performance requirements and test procedures, International Standards Organization, Standard number 15622, Geneva, Switzerland.

Kendall I.R. and Jones, R.P., 1999. An investigation into the use of hardware-in-the-loop simulation testing for automotive electronic control systems, Control Engineering Practice, Vol. 7, pp. 1343-1356.

Lean, G. Heffernan D. and Dunne, A., 1999. Digital networks in the automotive vehicle, IEE Computing and Control Engineering, Vol. 10, No. 6, pp. 257-266.

Levenson, N.G., 1995. Safeware: System Safety and Computers, Reading, M.A., Addison-Wesley.

Li, J., Yu, F., Zhang, J.W., Feng J.Z. and Zhao, H.P., 2002. The rapid development of a vehicle electronic control system and its application to an antilock braking system based on hardware-in-the-loop simulation, Proc. of the Inst. Mech. Engrs. Part D: Automobile Systems, Vol. 216, pp. 95-105.

Li, Q., Whitely, J.R. and Rhinehart, R.R., 2003. A relative performance monitor for process controllers, Int. Journal of Adaptive Control and Signal Processing, Vol. 17, pp. 685-708.

Lutz, R.R. and Mikulski, I.C., 2003. Operational anomalies as a cause of safety-critical requirements evolution, Journal of Systems and Software, Vol. 65, pp. 155-161.

Mathworks, 2005. Automotive Embedded System Design, WWW Page: http://www.mathworks.com/industries/auto/eng_tasks/embedsys.html, [accessed July 2007].

McDonald, M., Brackstone M. and Jeffery, D., 1994. Simulation of lane usage characteristics on 3 lane motorways, in Proceedings of the 27th ISATA Conference, Aachen, Germany, pp. 365-372, November 1994.

Microsoft Corporation, 1998. About multimedia timers, Microsoft MSDN Library, Platform SDK.

MISRA, 2001. Report 2 - Integrity, Motor Industry Software Reliability Report, Released February 2001.

MISRA, 2004.Guidelines for the use of the C language in vehicle based software, Motor Industry Software Reliability Report, October 2004.

Olsen, E.C.B., Lee S.E. and Wierville, W.W., 2002. Analysis of distribution, frequency, and duration of naturalistic lane changes, in Proceedings of the 46th Human Factors and Ergonomics Society, Baltimore, USA, pp. 1789-1793, Sept 2002.

Paramics, 2005. Paramics: Overview, WWW Page, http://www.paramics-online.com/overview/overview_main.htm, [accessed July 2007].

Phytec, 2003. phyCORE 167CS Hardware Manual, Phytec, April 2003.

Pont, M.J., 2001. Patterns for time-triggered embedded systems: Building reliable applications with the 8051 family of microcontrollers, ACM Press / Addison-Wesley Publishing.

Pont, M.J., Norman, A.J., Mwelwa C. and Edwards, T., 2003. Prototyping time-triggered embedded systems using PC hardware, in Proceedings of EuroPLoP 2003, Germany, June 2003.

SAE, 1993. Class C Application Requirement Considerations, SAE Recommended Practice J2056/1, SAE, June 1993.

SAE, 1994. Survey of Known Protocols, SAE Information Report J2056/2, SAE, April 1993.

Short M. and Pont, M.J., 2005. Hardware in the loop simulation of embedded automotive control systems, in Proceedings of the 8th IEEE International Conference on Intelligent Transportation Systems (ITSC 2005), Vienna, Austria, 13-16 September 2005, pp. 226-231.

Short, M. and Pont, M.J., 2007a. Fault-tolerant time-triggered communication using CAN, IEEE Transactions on Industrial Informatics, Vol. 3, No. 2, pp. 131-142.

Short, M and Pont, M.J., 2007b. Accelerated Fault Injection Testing Of Distributed Embedded Systems, University of Leicester, Technical Report ESL 07-04, March 2007.

Short, M and Pont, M.J., 2007c. Verification And Validation Of The HIL Test Facility, University of Leicester, Technical Report ESL 07-05, March 2007.

Short, M.J., Fang, J., Pont, M.J. and Rajabzadeh, A. (2007d) "Assessing the impact of redundancy on the performance of a brake-by-wire system". *SAE Transactions: Journal of Passenger Cars (Electronic and Electrical Systems),* **115**(7): 331-338.

Short, M., Pont M.J. and Huang, Q., 2004a. Simulation of Vehicle Longitudinal Dynamics, Technical report ESL 04/01, Embedded Systems Laboratory, University of Leicester.

Short, M., Pont M.J. and Huang, Q., 2004b. Simulation of Motorway Traffic Flows, Technical report ESL 04/02, Embedded Systems Laboratory, University of Leicester.

Short, M., Pont M.J. and Huang, Q., 2004c. Control Technologies For Automotive Drive-By-Wire Applications, Technical report ESL 04/04, Embedded Systems Laboratory, University of Leicester.

Sosnowski, J., 2006. Software-based self-testing of microprocessors, Journal of Systems Architecture, Vol. 52, pp. 257-271.

Stanton, N.A., Young M.S. and McCaulder, B., 1997. Drive-by-wire: The case of driver workload and reclaiming control with adaptive cruise control, Safety Science, Vol. 27, pp. 149-159.

Stoepler, G., Menzel T. and Douglas, S., 2005. Hardware-in-the-loop simulation of machine tools and manufacturing systems, IEE Computing and Control Engineering, Vol. 16, pp. 10-15, February 2005.

Storey, N., 1996. Safety Critical Computer Systems, Addison Wesley Publishing.

Tang, D., Hecht, H. and Hecht, M., 1997. Software Dependability Assessment - Myth and Reality, Appears in Issues in NASA Program and Project Management No 12, Washington, DC, June 1997.

Torngren, M., 1998. Fundamentals of Implementing Real-Time Control Applications in Distributed Computing Systems, Real-Time Systems, Vol. 14, pp. 219-250.

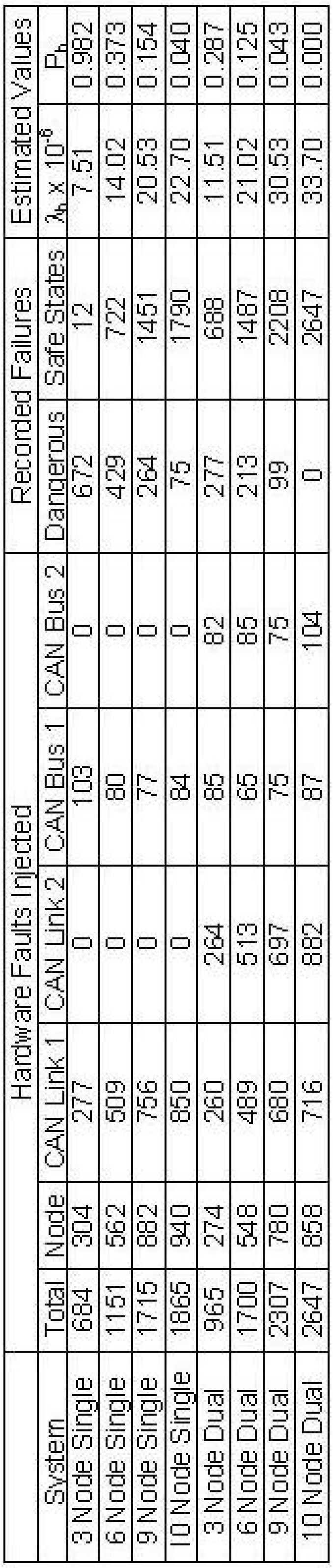
Turner J.D. and Austin, L., 2000. A review of current sensor technologies and applications within automotive and traffic control systems, Proceedings of the Institution of Mechanical Engineers, Vol. 214, Part D, No. D6, pp. 589-615.

Verburg, D.J., van der Knaap, A.C.M. and Ploeg, J., 2002. VEHIL: Developing and Testing Intelligent Vehicles, in Proceedings IEEE Intelligent Vehicle Symposium 2002 (IV2002), Versailles, France, June 2002.

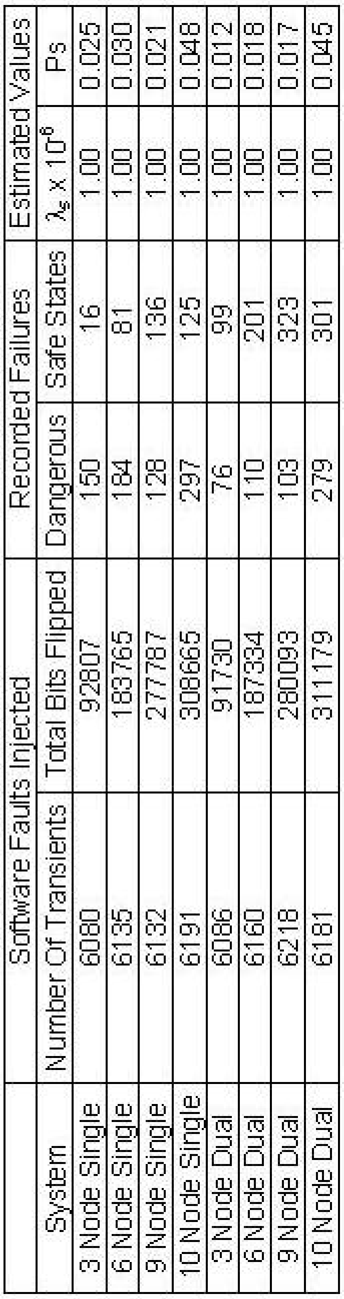
Wu, J., Brackstone M. and McDonald, M., 2000. Fuzzy sets and systems for a motorway microscopic simulation model, Fuzzy Sets and Systems, Vol. 116, No. 1, pp. 65-76.

Yang, Q., 1997. A simulation laboratory for evaluation of dynamic traffic management systems, Ph.D. dissertation, Massachusetts Institute Of Technology, USA.

Appendix A – Summary of hardware and software fault injection results



	Hardware Faults Injected						Recorded Failures		Estimated Values	
System	Total	Node	CAN Link 1	CAN Link 2	CAN Bus 1	CAN Bus 2	Dangerous	Safe States	$\lambda_h \times 10^{-6}$	P_h
3 Node Single	684	304	277	0	103	0	672	12	7.51	0.982
6 Node Single	1151	562	509	0	80	0	429	722	14.02	0.373
9 Node Single	1715	882	756	0	77	0	264	1451	20.53	0.154
10 Node Single	1865	940	850	0	84	0	75	1790	22.70	0.040
3 Node Dual	965	274	260	264	85	82	277	688	11.51	0.287
6 Node Dual	1700	548	489	513	65	85	213	1487	21.02	0.125
9 Node Dual	2307	780	680	697	75	75	99	2208	30.53	0.043
10 Node Dual	2647	858	716	882	87	104	0	2647	33.70	0.000



	Software Faults Injected		Recorded Failures		Estimated Values	
System	Number Of Transients	Total Bits Flipped	Dangerous	Safe States	$\lambda_s \times 10^{-6}$	Ps
3 Node Single	6080	92807	150	16	1.00	0.025
6 Node Single	6135	183765	184	81	1.00	0.030
9 Node Single	6132	277787	128	136	1.00	0.021
10 Node Single	6191	308665	297	125	1.00	0.048
3 Node Dual	6086	91730	76	99	1.00	0.012
6 Node Dual	6160	187334	110	201	1.00	0.018
9 Node Dual	6218	280093	103	323	1.00	0.017
10 Node Dual	6181	311179	279	301	1.00	0.045