Paper Preprint – To Appear In:
Control Engineering Practice, 2008

Assessment of performance and dependability in embedded control systems: methodology and case study

Michael Short[1], Michael J. Pont and Jianzhong Fang

Embedded Systems Laboratory, University of Leicester,
University Road, Leicester LE1 7RH, UK

Abstract

Distributed and embedded control systems play an increasing role in modern safety-critical systems, and there is a pressing need to investigate the impact of different design decisions on system performance and safety integrity. In this paper, a methodology for the measurement and estimation of such attributes is presented. The methodology integrates statistical fault-injection testing with the application of on-line, model-based performance monitoring of the embedded control system under test. The methodology is particularly suited to late-phase system testing in which "hardware-in-the-loop" (HIL) simulation techniques are employed. The methodology is illustrated in an extended case study, in which the performance and dependability of eight possible designs for an automotive control system are compared. It is concluded that the methodology is a useful adjunct to the available testing and analysis techniques for such systems.

Keywords: Safety Systems, Performance Monitoring, Distributed Embedded Systems, Fault Injection, Testing Techniques, Automotive Control.

1. Introduction

Distributed and embedded systems play an increasing role in the development of safety-critical systems. A distributed embedded system consists of a number of electronic control units (ECUs) connected to one another via one or more serial communication buses. For example, in the modern passenger vehicle, up to 70 such ECUs are connected to a variety of sensors and actuators in order to realize high-level functionality and services (Lean et al. 1999): Isermann (2008) surveys the use

[1] Corresponding author:
Tel: +44 (0)116 252 5052
Email: mjs61@le.ac.uk (M Short)

of such technologies in modern passenger vehicles. With the advent of systems such as Drive-by-Wire, these distributed embedded systems will have no mechanical backup, and will play a crucial role in safety (Isermann et al. 2003; Isermann 2008).

Since many such systems are safety-critical in nature, special measures must be taken at all stages of the design process to ensure that the required Safety Integrity Level (SIL) has been achieved. 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, safety, security and availability measures for each sub-system and component as part of a safety case.

Many different design decisions have to be considered in the creation of such systems – such as the choice of hardware / software architecture, programming language and communications network - and many of these decisions are known (or thought) to influence both the performance and dependability of the resulting system (e.g. MISRA 1994; MISRA 2004; SAE 1993; Holzmann 2006). For example, an embedded system may employ one or more design paradigms: Event Triggered (ET), Time Triggered (TT), Preemptive (P) or Cooperative (C) (e.g. see Buttazo 1997). Although many - sometimes opposing - opinions have been voiced regarding the properties of these various paradigms in recent years (e.g. Kopetz 1991; Bate 1999; Pont 2001; Xu & Parnas 2003; Scheler & Schroeder-Preikschat 2006), little empirical evidence has been presented which provides direct comparisons of their functional performance and safety[2].

This lack of evidence may be attributed, in part, to the problems inherent in determining the reliability and dependability of such systems. 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). Additionally, assuming that effective testing can take place in the limited time available, it may also be extremely difficult to produce an 'oracle' that can verify the correctness – and safety properties - of each test output (Butler & Finelli 1993). As such, achieving accurate estimations of safety and reliability properties of complex, reactive embedded control systems is a wide and ongoing area of research.

---

[2] It should be noted that some small-scale studies have been conducted in this area, for example by Lonn & Axelsson (1999); Claesson et al. (2003), Albert (2004) and Short et al. (2008a).

In this paper, an alternate means for such estimates for critical embedded systems is presented. The methodology is designed to complement existing strategies employed in the design process for embedded systems, and it is best suited to late-phase system verification and validation exercises[3] in which HIL simulation is employed, as shown in Figure 1. The methodology utilises statistical fault-injection in conjunction with on-line performance monitoring techniques to generate data that is then used for the estimation of key system attributes. This approach was taken in order to both maximise the usefulness of the available testing time, whilst providing an automated means to check the validity of the system outputs against a specification in real-time.

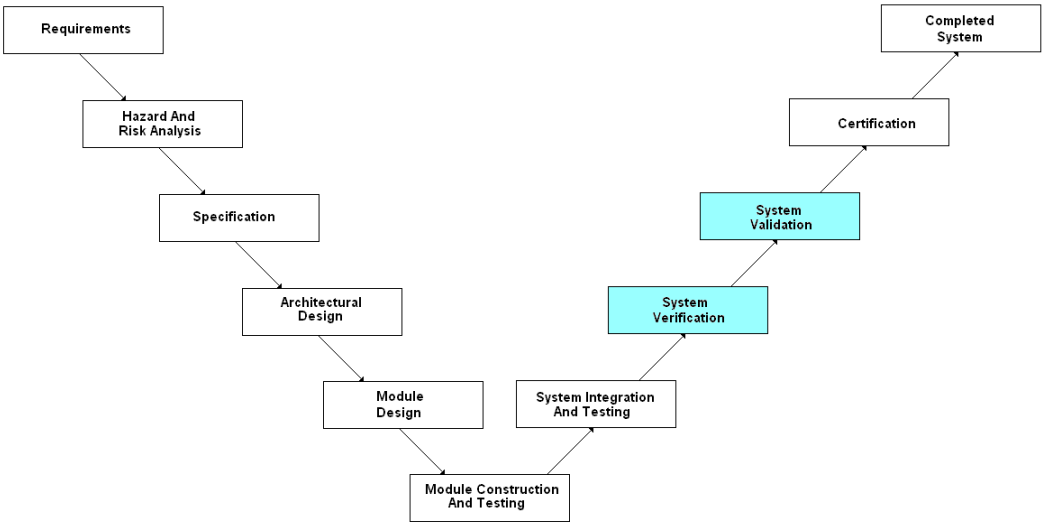

Fig.1. Relation of the proposed methodology to the traditional design process

The remainder of the paper is organised as follows. In Section 2, the concept of HIL and fault injection testing is reviewed, and the proposed methodology is outlined. In Section 3, an effective mathematical basis for the implementation of fault injection is discussed. Section 4 then describes the implementation of an on-line performance monitor which may be used to classify the system behaviour. Section 5 describes an existing HIL test facility, and explains how the proposed techniques were integrated within this framework. Section 6 describes an extended case study, which compares the performance and dependability of eight representative systems designed using a combination of ET, TT, P and C techniques. Section 7 presents the results of this case study. In Section 8, the paper is concluded.

[3] For larger systems, the methodology may also be applied to sub-system testing and integration.

2. Proposed methodology

In this section, the proposed methodology is outlined. The section begins by describing and reviewing the use of HIL simulation and fault injection techniques, and some potential problems that can arise when testing non-trivial systems.

2.1 HIL Simulation

The principle of HIL simulation of an embedded control system is illustrated in Figure 2. The embedded system outputs are fed directly to the simulation, 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, thus closing the control loop. Simulations of this nature have been used successfully in a variety of applications, including verification of new machine tool designs (Stoepler et al. 2005), aircraft autopilot system design (Gomez 2001), numerous military applications (Cole & Jolly 1996) and testing of automotive ECUs (Ellims 2000).

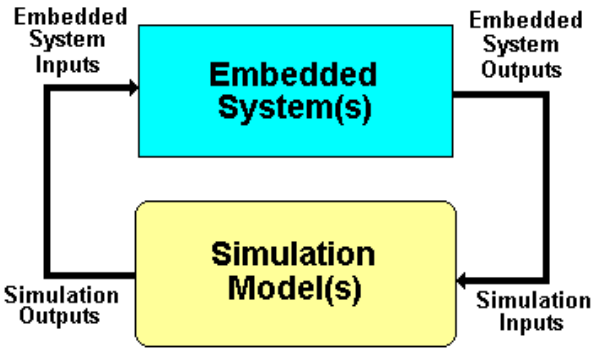

Fig.2. HIL simulation principle

HIL simulation has been shown to both increase the quality and reduce the time-to-market (and hence development costs) of prototype embedded control systems. In addition, when developing safety-critical systems it is often inappropriate, unethical or even impossible to fully test the system within its natural operational environment (Storey 1996; Levenson 1995). In such cases, HIL simulation of the system's environment can allow developers to make assessments of performance without compromising safety.

Given the above features, use of HIL simulation may be felt to be particularly appropriate during the verification and validation stages of the development lifecycle of safety-critical systems (Storey 1996; Levenson 1995). The complexity involved in such large-scale testing of distributed

embedded systems and software dictates that structured, well-defined testing procedures and benchmarks are preferred over more 'traditional', ad-hoc methods for assessing system performance (Broekmann & Notenboom 2002). During these testing procedures, it is normal to measure various outputs from the system whilst performing stress and fault-injection testing, in order to determine how the system will react to abnormally high loads and random failures of components and subsystems.

The type of embedded system of concern in this paper will usually employ some form of redundancy in order 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 (Isermann 2002; Hammett 2002). Where redundancy is employed, fault injection is the preferred means for extracting dependability information (e.g. see Arlat et al. 1993). 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. Care must also be taken when interpreting the resulting data, as analysing system performance in situations where multiple (possibly adaptive) control loops are employed and the set points / disturbances may be changing rapidly is a non-trivial task.

In the following section two techniques which may be used to great effect to complement basic HIL simulation techniques are described, in order to improve the efficiency of the testing process and help to ameliorate the problems outlined above.

2.2 Complementary techniques

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.

Also of relevance in this context is the concept of Control Performance Monitoring (CPM). Much research within the control community has concentrated on this concept (e.g. Qin 1998; Huang & Shah 1999; Jelali 2006). CPM provides on-line, automated procedures which give information to operators and plant engineers: this may be used to determine whether specified performance targets are being met for the controlled processes. Many different approaches have been taken in this area. For example minimum variance benchmarking, linear quadratic regulator benchmarking and various model-based approaches: Jelali (2006) provides a recent summary and comparison of work in this area. Most of these techniques provide performance indices in the range [0, 1] that indicate good or bad control over a specified time history.

From the point of view of safety-critical control systems, measures of system functional safety can be obtained, in part, by applying the principles of CPM and determining how close the controlled variables are to perceived critical levels (such as reactor temperature, vehicle separation distance etc) by using an idealised system specification as a benchmark.

It would seem that with an appropriate implementation framework, both the RET and CPM may be applicable to automated fault-injection testing for performance and dependability evaluation in complex safety-critical control systems.

The following section proposes a general framework for such an implementation.

2.2 A framework for performance and dependability evaluation

The framework that is proposed consists of three basic elements; an HIL simulator, complemented by a performance monitor and a fault injector. The structure of the framework is shown in Figure 3.

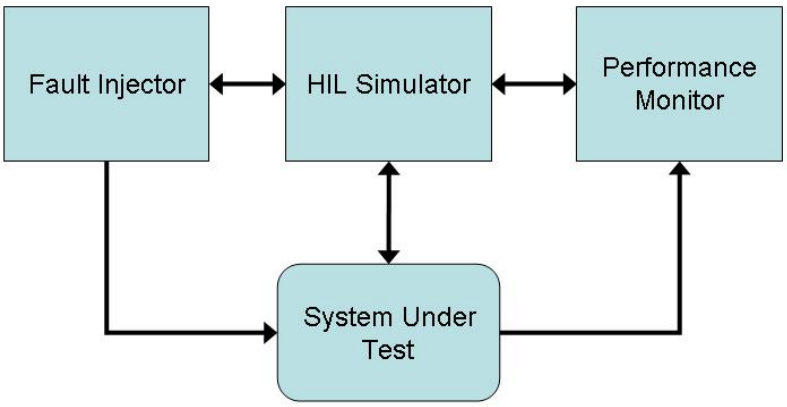

Fig.3. General structure of the performance assessment framework

The framework operates as follows. A test sequence is started, and the HIL simulator is used to generate the real-time inputs to the system under test and read its outputs. The fault injector implements a statistical model of the system under test, adapted using the RET. At "random" times, faults and abnormal operating conditions are introduced into the system, either using direct fault injection or through the HIL simulator. The performance monitor continuously assesses the quality of control of the system under test in real-time, and provides a log of the recorded performance. This log, in conjunction with the log of faults injected, may then be used to calibrate suitable reliability models of the system.

The fault injector and performance monitor are described in depth in the following two sections.

3. Fault injection framework

The RET, as applied in the evaluation of high-integrity software, 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[4]. It follows that

[4] Indeed, where formal techniques have been employed in the software design process, it may be possible to prove mathematically 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.

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 4, showing the operational profile of a typical system and highlighting the point P_{rare} where normal operation ends and abnormal operation begins.

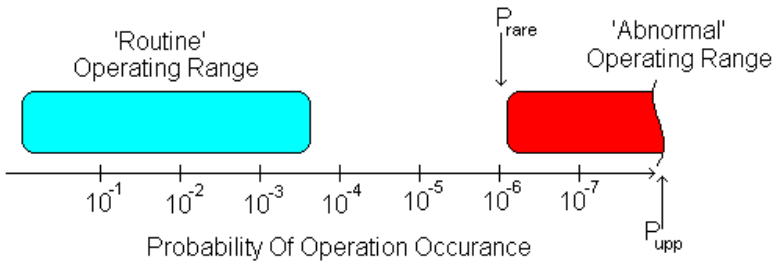

Fig.4. Normal and abnormal operational profiles

Also indicated in Figure 4 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:

$$P_{rare} > P_c > P_{upp} \tag{1}$$

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 normal system operations or_1, or_2 … or_n or [2] the critical set containing abnormal 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 (2):

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

(2)

where $P_r >> P_c$ in order to maintain consistency with the assumptions above (and shown in Figure 4).

In order to exploit the limited test time available for the reliability assessment, two factors must be considered. The first is that the probabilities of occurrence of the operations in the critical test set are adjusted by a likelihood ratio α (see (3)) during the period of testing T, giving pc_1', pc_2' … pc_n':

$$pc_i' = pc_i.\alpha$$

(3)

In which the likelihood ratio α is chosen as some suitable value for a tractable test, e.g. $1/P_{rare}$. The second is that the test sequences must be selected to cyclically exercise as many of the operations in both the regular and (adjusted) critical sets in the least possible time, to obtain maximum possible test coverage. 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 estimated as shown in (4).

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

(4)

However, it is assumed 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).

In the approach proposed in this paper, statistical information regarding these measured failures may then 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 the system failure rates.

Assuming a sensible choice is made in the selection of α, a suitable statistical model may be calibrated and used to select test patterns for the HIL system which simulate abnormal environment conditions. Assuming that the fault activation time for the system is much less than the total available test time T (as will be true in most – but not all - cases), and observing that the average 'mission time' for an embedded control system is comparatively short (and that a safe state can be entered upon detection and tolerance of a sub-system failure in a relatively short time in comparison to T), then it is suggested that sufficient 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.

In order to implement the above procedure, 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 highly application dependant, and will more than likely contain interfaces by which hardware, software, sensor, actuator and communications errors can be introduced into the system. Such mechanisms have been employed in this paper with the test facility outlined in Section 5.

4. On-Line Performance Monitor

Despite being a relatively new field of engineering, performance monitoring of control systems has expanded rapidly over the last decade and many innovative algorithms and methodologies have been proposed. From recent survey material, an analysis of the existing methodologies reveals that from the point of view of HIL simulation, where dynamic models of the process under control are readily available, a model-based approach would seem to be the most suitable. The methodology presented in this paper is an adapted version of the relative performance monitor presented by Li et al. (2003).

The monitor is shown in Figure 5 and consists of three separate parts; the actual control system implementation and existing HIL simulation, a reference model that specifies the required closed loop dynamic behaviour, and the metric calculator. From an implementation perspective, integration of the monitor to an existing HIL simulator should be relatively straightforward, as the monitor can utilise the same timing and computation resources as the simulator.

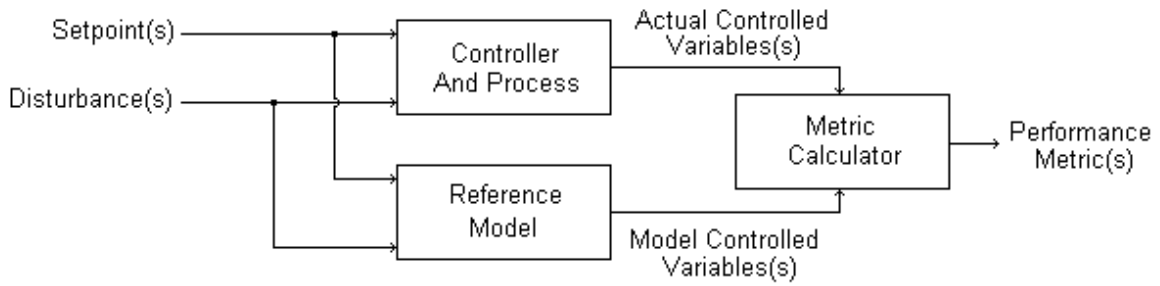

Fig.5. Performance monitor

Selection of a suitable closed loop reference model will depend highly on both the controller and process dynamics. For many plants and controllers, the required closed-loop reference model can be as simple as a low-order transfer function. For certain classes of non-linear processes and controllers, the reference model may be obtained from the required performance specifications for the control system. For the most complex of controllers, the required closed loop system behaviour may even be generated by a secondary, idealised simulation of both the process and controller.

Once this required reference model has been determined, the actual set points and disturbances that are fed to the real controller and process are also fed into the reference model, thus dynamically generating idealised target outputs for the controlled variables. These reference model outputs are then passed, along with the actual controlled outputs, to the metric generator for further comparison and processing, to generate the performance assessment. Many different methodologies may be used to generate these metrics; the approach taken by Li et al. (2003) to generate the performance metric is to use the ratio of reference and actual weighted moving average of squared errors, as shown in (5):

$$RPI = \frac{M(e_r)}{M(e_a)} \tag{5}$$

where $M(e_r)$ and $M(e_a)$ are the weighted averages of the reference model and actual errors respectively, calculated recursively at each iteration k by (6):

$$M_e(k) = \lambda \cdot M_e(k-1) + (\lambda - 1) \cdot e^2 \tag{6}$$

where e^2 is the squared error in the reference or actual control systems, and λ is a constant between 0 and 1, indicating the weighting of present and previous data.

It should be noted that this performance metric is intended to provide a measure of the improvement potential, in terms of potential error reduction in the actual system, and is not generally of interest in this application. In addition, unless a very accurate noise model is included in the reference, the performance in the-steady state will tend towards 0 as the model error approaches 0: indeed it may be of more use to employ a metric that also indicates the level of noise that is present in the actual system, allowing the impact of partial system failures on noise rejection to be categorised. In Section 5, some possible alternate metrics will be proposed for use with the case study.

5. Test Facility Description

In the remainder of this paper, the techniques described in Sections 2, 3 and 4 will be applied to an existing HIL test facility (previously developed in the Embedded Systems Laboratory at the University of Leicester). The test facility is described briefly in this section.

5.1 Real-Time Simulation

A real-time HIL simulator has been developed in order to assess and compare different embedded software and hardware architectures for use in automotive control applications. This HIL simulation is described in detail elsewhere (Short & Pont 2005; Short & Pont 2008). Briefly, the simulation consists of a real-time representation of a motor vehicle travelling down a three-lane motorway, under realistic traffic conditions. It enables different implementation architectures to be assessed and compared in a variety of realistic and repeatable scenarios, and enables multiple faults to be injected into the underlying hardware/software. The facility is implemented on three PCs: an overall schematic is shown in Figure 6.

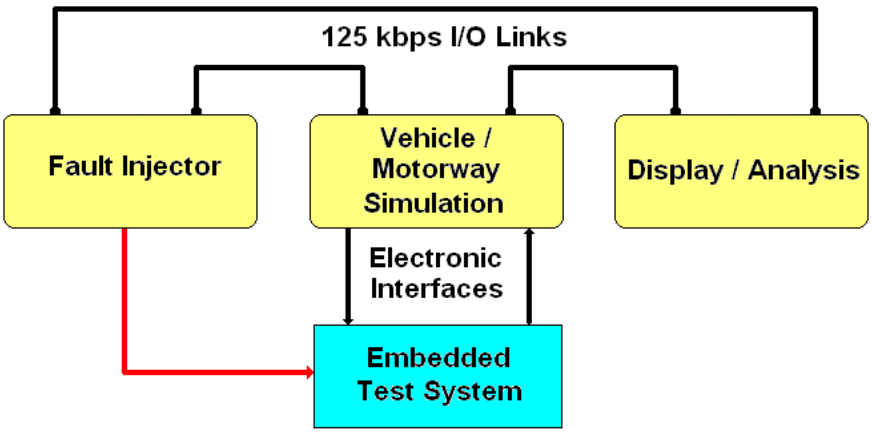

Fig.6. Test facility general arrangement

The first PC employs the DOS operating system, and uses a re-programmed hardware timer to iterate the HIL simulation every 1ms (Pont et al. 2003); this PC was also used to implement the performance monitor. The second PC, running the Windows operating system, is connected to the first via a duplex high-speed serial link, and provides an interactive, graphical interface for user feedback and simulation control. A screenshot of the user interface is shown in Figure 7. The third PC, again connected to the first by means of a high-speed duplex serial link, is able to inject the faults into the system and is used in the study described in this paper to implement the statistical fault models.

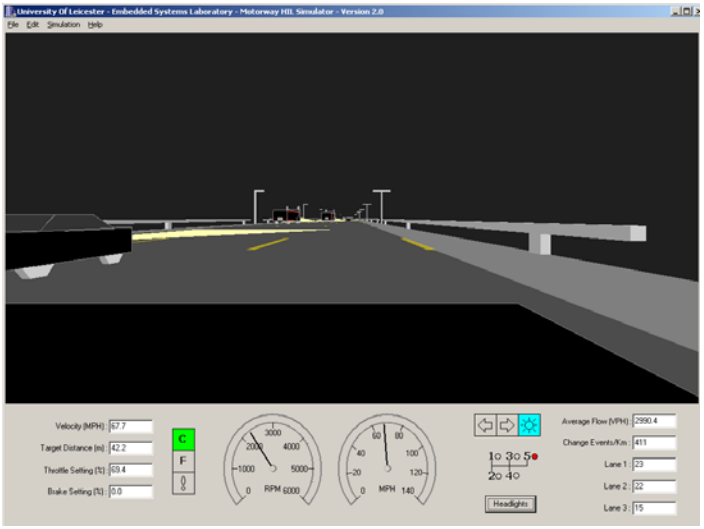

Fig.7. Test facility user interface

5.2 ACC Principle

The case studies to be described in the following section apply the methodology outlined in the previous sections to various implementations of a throttle- and brake-by-wire system, with adaptive cruise control (ACC) capability.

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 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. It also sends status information to the driver. Figure 8 shows the overall principle of ACC. The outer acceleration control law used in

this study was given by (7), where v_{set} is the desired set speed, h_{set} is the desired headway and d is the vehicle separation distance. An integrator was employed to convert the desired acceleration signal α_d into a velocity demand for an inner loop PID (speed) controller, represented by equation (8).

$$\alpha_d(t) = \min\left(\left[v_{set}(t) - v_{acc}(t)\right], \left[-0.3 \cdot \left(\left(v_{acc}(t) \cdot t_{set}(t)\right) - d(t)\right) + 0.4 \cdot \left(v_t(t) - v_{acc}(t)\right)\right]\right) \tag{7}$$

$$u(t) = 0.198\, e(t) + 0.050 \int_0^t e(t)dt + 0.076 \frac{de(t)}{dt} \tag{8}$$

where *e(t)* was the error between desired and actual vehicle speed, and *u(t)* was the commanded actuator signal. If *u(t)* was positive, this was taken as a throttle signal; otherwise it was used as the brake actuation signal. A 0.5 % hysteresis band was employed to prevent frequent switching between throttle and brakes.

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). Since the deceleration is limited to this value, the system additionally implements a predictive collision warning system. This system predicts, based on the kinematics of the current road situation, when a collision is likely to occur due to ACC system braking saturation and informs the driver (via a sounder) that manual intervention is required. In addition, the system also provides an anti-lock brake (ABS) controller for each wheel of the vehicle, and traction control stability (TCS) algorithm. Further details of the nature of the controllers employed may be found elsewhere (Short et al. 2004a).

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, the required Safety Integrity Level (SIL) for the overall system was classified at SIL 3, with a minimum failure rate of 10^{-7} dangerous failures / hour. This decision was based on the MISRA guidelines (MISRA 2001).

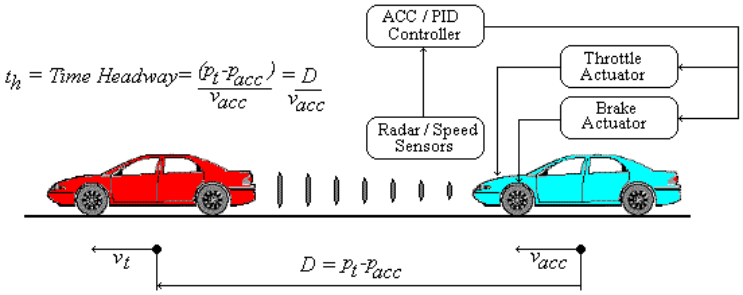

Fig.8. ACC principle

Each system architecture to be described in the case study was designed to a minimum specification. This specification included the following two key elements:

1) The response to a step change in either throttle or brake demand should occur within approx. 100ms (95 % settling time for applied acceleration commands) and should be critically damped.

2) The response to a change in input conditions (either to a driver button press, or to a change in road conditions) should take no longer than 100ms.

5.3 Reference Model

The reference model was implemented as a series of digitised state equations, which included the throttle and braking actuation subsystems and the free driving / following dynamics of the ACC system. The state equations were loosely based on the state-space model proposed by Baraket et al. (2003) for an ACC system operating in a variety of known, healthy conditions. This simplified model has been found to successfully capture the gross vehicle-following dynamics of many working ACC systems (Baraket et al. 2003), given that the actual vehicle models (including those employed in this HIL simulation (Short & Pont 2005; Short & Pont 2008)) are complex and non-linear. The reference model (and desired closed loop response of each controller implementation) was based on the equations given by (9).

$$\dot{u}_{act}(t) = \frac{\alpha_d(t) - u_{act}(t)}{\beta}$$

$$\dot{v}_{acc}(t) = u_{act}(t)$$

$$\dot{d}(t) = v_t(t) - v_{acc}(t) \qquad (9)$$

with u_{act} as the applied acceleration / brake force and β a lag representing the combined response of the inner control loop and actuator subsystems (0.033s in this study). When operating with the ACC system enabled, α_d is as given by (7) and when disabled it is given by the output of the HIL driver model (see Short & Pont 2005 for details). The input parameters v_t, v_{set} and t_{set} are taken directly from the HIL simulation. The primary output parameters of the reference model were v_{acc} and d. The separation distance d was required to be reseeded in the performance monitor when a new physical target was acquired (e.g., after a lane change).

In the following section, the metrics that were recorded using this reference model will be described.

5.4 Recorded Performance Metrics

In the present study, three different performance metrics were measured; the first, described by (10), represents a safety margin of the system, in terms of the ratio of the reference and actual separation distance of the vehicles. The second metric, described by (11) represents the quality of velocity control in the system, by measuring the scaled absolute deviation of the reference and actual vehicle velocities, where a performance of 1 indicates perfect following of the reference dynamics, and a performance of 0 indicates a persistent deviation of more than 10 MPS from the nominal system dynamics. In a similar manner, the third metric, described by (12), represents the quality of service being provided through the by-wire elements of the system, scaled over the span of the signal range:

$$\eta_S(k) = \lambda \cdot \eta_S(k-1) + (1-\lambda) \cdot \frac{D_a}{D_r} \tag{10}$$

$$\eta_p(k) = \lambda \cdot \eta_p(k-1) + (1-\lambda) \cdot \left[1 - \left(\frac{|V_r - V_a|}{10}\right)\right] \tag{11}$$

$$\eta_c(k) = \lambda \cdot \eta_c(k-1) + (1-\lambda) \cdot \left[1 - \left(\frac{|T_r - T_a| + |B_r - B_a|}{100}\right)\right] \tag{12}$$

where D represents the vehicle separation distance and V represents the velocity of the vehicle, T represents the applied throttle setting and B represents the applied brake setting, both expressed as a

percentage of full scale. In all cases, a time constant λ equal to 1s was selected to perform a moving averaging of the data. In this instance, the performance monitor was implemented as a single task within the existing (DOS) scheduler framework, running every millisecond. Over the duration of each experiment, the minimum level of each of the performance indices was recorded to gain some idea of the worst-case deviation of the actual control performance from the ideal.

5.5 Fault Model

In order to implement the RET within the HIL simulation framework, it was necessary to determine representative failure rates for each hardware element of the systems under test, and an appropriate occurrence rate for the rare events under consideration. The test systems to be described in the following section were created using Infineon C167CS microcontrollers (one per node) running at a 20 MHz oscillator frequency (Phytec 2003), connected using 1 Mbit/s twisted-pair Controller Area Network (CAN) links (Bosch 1991). The representative failure rates employed in this study are shown in Table I, with values derived from representative sources (Short et al. 2007d; Short & Pont 2007).

Table I: 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

Implementation of the rare events technique was achieved as follows. A statistical fault model was calibrated using information provided in a script file: 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. 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.

6. Case Study

This section presents the results from an extended case study. This case study was performed in order to test the efficacy of the proposed methodology; and also to apply the methodology to

generate empirical data regarding the comparison of performance and dependability of eight representative systems, designed using a combination of ET, TT, P and C techniques.

6.1 System Designs

As mentioned in the introduction, system designers often have flexibility in decisions about the number and location of processor nodes, the allocation of software tasks to particular nodes and the overall choice of software architecture. As different network topologies and design paradigms 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, the aim was to explore the ways in which the HIL simulator and proposed methodology could be used to support the process of comparing different system designs, and provide a source of comparative data. The particular focus of the study was to explore the links between the chosen microcontroller / network topology and choice of design principle on the resulting control performance and functional safety of the system. To conduct this study, four different hardware topologies for the ACC system were explored. Each of the four different hardware architectures employed various levels or redundancy, and the simulator and proposed methodology was used to assess the failure modes of the architectures under intense fault injection. In addition, each of the four possible hardware architectures was implemented using either a TTC-style, or a ETP-style system, communications and software architecture. The data from each of these eight systems then formed the basis for a qualitative estimation of dangerous failure rate.

6.2 Hardware Architectures

The four different configurations of microcontroller that were considered are shown schematically in Figure 9 to Figure 12. 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 software functionality was distributed in each architecture, with each software function being allocated a number: this will be explained further in Section 6.3.

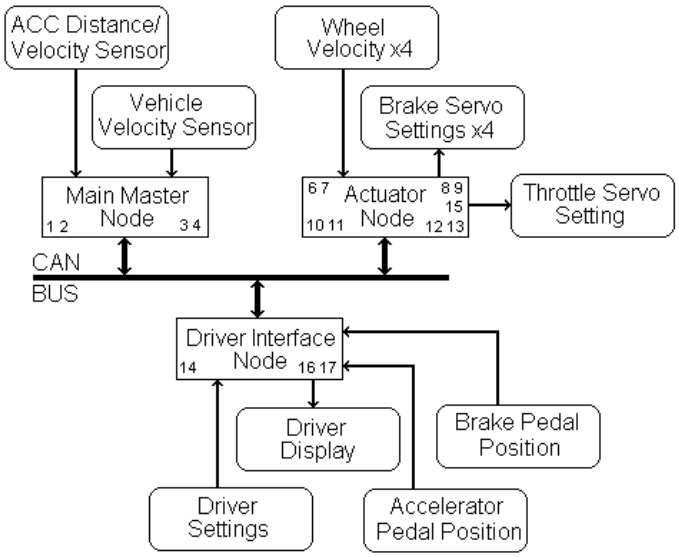

Fig.9. Three-node architecture

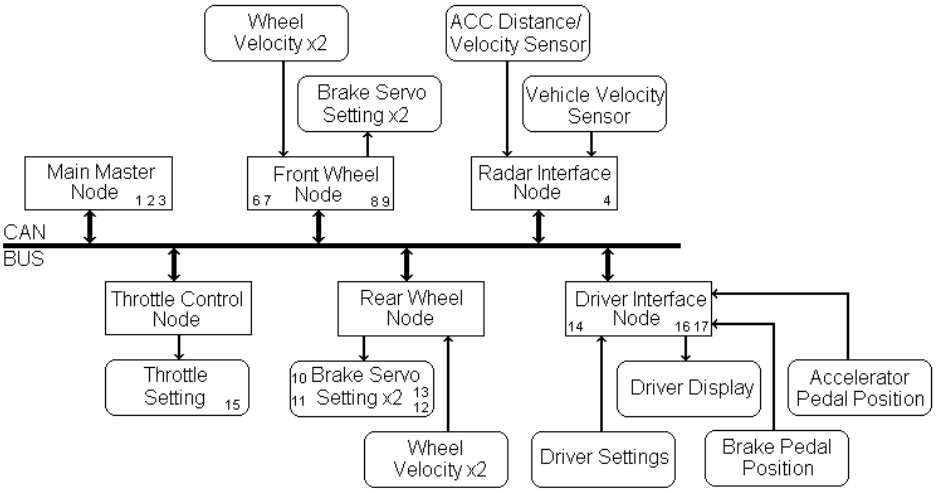

Fig.10. Six-node architecture

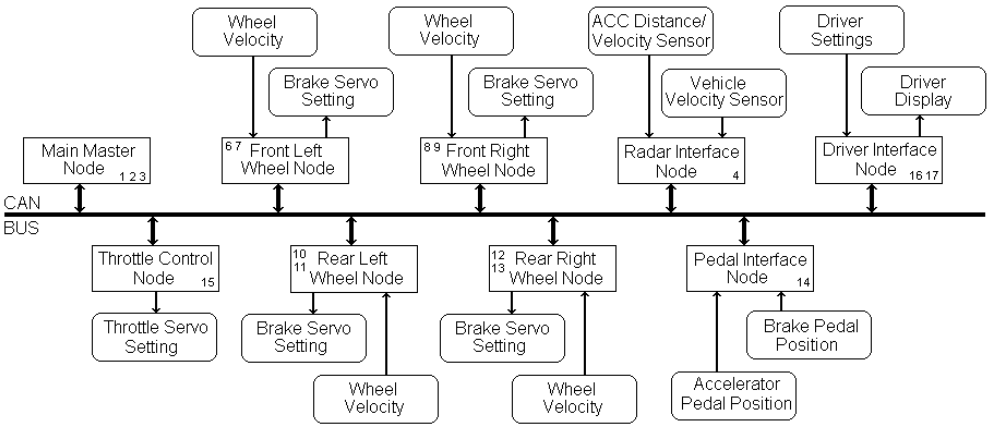

Fig.11. Nine-node architecture

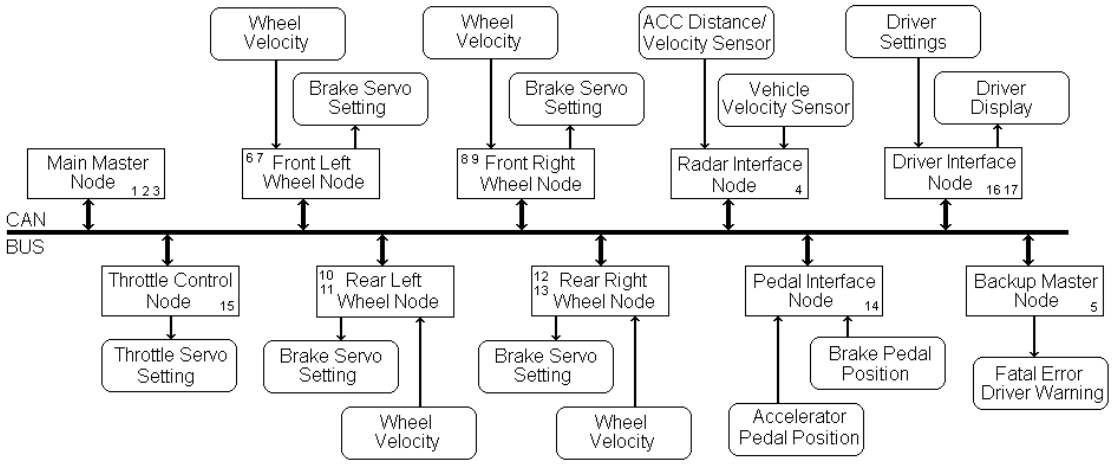

Fig.12. Ten-node architecture

In the most centralised architecture (Figure 9), 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. Each architecture then progresses through varying degrees of distribution until the last architecture, shown in Figure 12. From this figure it can be seen that the system is partitioned due to physical distribution around the vehicle. In Figure 12, each wheel 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.

It can also be seen that the increase in the number of nodes has the potential to increase the 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, a failure of a single brake processor will have a reduced impact on braking capacity and no impact on throttle-actuation ability. In addition, in the systems described in this paper, the backup Master employs dynamic redundancy with fault detection, i.e. it self-activates when the loss of the main Master is detected. The presence of the backup Master in the ten-node system (for example) ensures that when the main Master fails, the system may still be able to operate.

6.3 Software and Communication Architecture

It was noted in the introduction that - while it is generally accepted that TT behaviour can provide a good platform for safety-critical systems - arguments have also been advanced in favour of alternative architectures. For example, as discussed in the introduction, the use of ET-style communication and software architectures has been proposed, and there has also been debate regarding the use of P / C software architectures on individual nodes.

In this study, two architectures were considered, each using a standard Controller Area Network (CAN) protocol as the communication medium (Bosch 1991).

The first architecture was a fully TTC design based around a shared-clock scheduler (Pont 2001). Although CAN is often viewed as an ET architecture, it has been shown that fully TT behaviour can still be achieved using this protocol (Pont 2001; Ayavoo et al. 2006; Short & Pont 2007). For each TTC system considered in this case study, the same 'tick' interval (5 ms) was employed.

The second architecture was an ETP design, with 'loose' TT operation. This system design was based on the paradigm for which CAN was originally intended (an event-triggered communications with a Multi-Master architecture: Lean et al. 1999). In the ETP systems described in this paper, individual nodes employ a TTP architecture, but ET response to external events (such as button presses, reception of CAN messages) was allowed. In addition there was no inherent clock synchronisation between nodes. Task priorities were assigned using rate-monotonic priority assignment (Buttazo, 1997). A 'tick' interval of 1 ms was employed. In these respects the system operates much like a standard RTOS. Further details of the scheduler design have been documented by Fang (2006).

Table II shows a summary of the software functionality that was required to implement the ACC system.

Table II: Software task summary

Task Number	Task Functionality
1	ACC algorithm actions
2	Speed sensor fusion and filtering
3	Centralised processing of vehicle status
4	ACC sensor data acquisition and processing
5	'Limp home' functionality
6	Front left wheel speed sensor processing
7	Front left wheel brake and ABS control actions
8	Front right wheel speed sensor processing
9	Front right wheel brake and ABS control actions
10	Rear left wheel speed sensor processing
11	Rear left wheel brake and ABS control actions
12	Rear right wheel speed sensor processing
13	Rear right wheel brake and ABS control actions
14	Throttle and brake pedal sensor processing
15	Traction and throttle control actions
16	Driver display interface control
17	Driver command processing

In Figure 9 to Figure 12, the distribution of this functionality in each of the four hardware architectures can be seen. In each system, the software was created mainly in C, with a small amount of assembly required for the ETP systems to implement the 'context switch' mechanism.

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[5], sanity checks of control signals, task overrun detection mechanisms and the use of the on-chip exception traps in the C167 processor, as 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 summarized in Figure 13.

[5] The duplex data fields were checked before each use of the variable; any discrepancy forced a hardware reset.

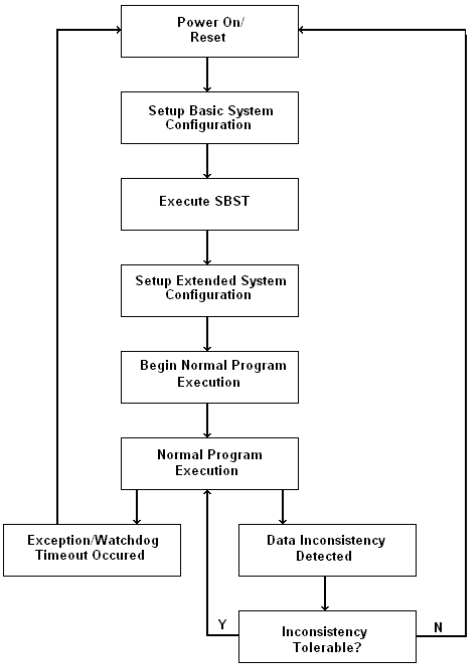

Fig.13. Approach to software fault tolerance

A number of techniques were also employed during the software design process for each system, prior to experimentation. These included the adoption of structured programming techniques, adherence to good programming practice (MISRA 2004; Holzmann 2006), and regular use of the public domain static code analysis tool splint[6] during software development. As a final check (prior to the trials described here), the C source code in each system was formally verified using bounded model-checking techniques (summarized below), and each system was required to run failure-free (and free of failed assertions) for a continuous period of 48 hours using a simulation of a busy motorway.

The formal verification process that was employed was as follows. Initially, the high-level safety and functional requirements of the system were refined into node-level requirements for each of the architectures, using techniques similar to those described by Wong & Joyce (1998). These node-level requirements were then translated into code-level verification conditions, in the form of module pre and post-conditions implemented as sanity checks and assertions placed directly into the C source code. Prior to model checking, the microcontroller-specific features of the code were also modeled; for example a read of the ADC data register was represented as a non-deterministic program input of the required bit length. The C source code, along with this information, was then passed to the model checker "CBMC" (described by Clarke et al. 2004). The model checker first compiles the code, whilst unwinding each program loop to its full extent. It then verifies that loop bounds are not exceeded, and also checks the integrity of pointer arithmetic and array bounds/math

[6] Please see www.splint.org for details.

operations. Finally, the code is further processed into equivalent Boolean formulae and passed to a Boolean Satisfiability (SAT) solver. The SAT solver executes every possible trace of the given program with the specified non-deterministic inputs, to verify that none of the assertions or sanity checks can fail during run-time. If an assertion failure is detected, an appropriate counterexample is generated.

It should be noted that a number of low- and medium-intensity bugs were detected and subsequently removed from the software during this verification process.

6.3 Experimental methodology

In order to test the efficacy of the proposed methodology, three separate sets of experiments were conducted. The first set of experiments was performed to assess the efficiency of the proposed CPM methodology and also the reference model that was employed. In these experiments, a simple test scenario was considered, focusing on a situation where the host vehicle, cruising at 112.6 KPH (70 MPH), closes onto a lead vehicle travelling at 96.5 KPH (60 MPH). The lead vehicle subsequently performs a sharp deceleration manoeuvre and settles at 48.3 KPH (30 MPH). The nine-node TTC system was employed in these experiments.

In the first experiment, data for the 'healthy' system were recorded. In the second experiment, a fault was injected into the braking sub-system that simulated a complete power loss in the front left and rear right braking nodes ('braking' fault). In the third experiment, a fault was introduced effectively reducing the sample rate of the ACC system to 5 Hz ('timing' fault).

In the second set of experiments, the facility was used to perform intensive hardware fault injection into each system. The experimental procedure for this was as follows. Each system was tested, in turn, 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 also underwent 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 activated the ACC system. The driver then changed lanes as appropriate, whilst allowing the ACC system to control the vehicle longitudinal motion until the ACC system applied the collision detection warning, as discussed in Section 5.2. At this point the driver resumed manual control and manoeuvred the vehicle out of the current road situation. When

the road subsequently became clear, the driver again attempted to achieve the cruise speed and reactivate the ACC. The ACC system under test also disengaged if a fault was detected, and issued a warning to the driver. At this point the driver would 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 either a vehicle collision (with the safety margin reduced to 0), or the vehicle becoming stranded (or otherwise totally out of control) for a given time period (30 seconds). This was implied by any of the performance metrics described in the previous section reading less than 10% for more than 30 seconds. The likelihood ratio during this phase of testing was set at $\alpha = 10^6$, and the fault models calibrated using the information given in Section 5.5.

In the third set of experiments, the facility was used to perform intensive software fault injection into each system; employing a further 100 hrs of testing per system. 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 likely to experience such 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 $\alpha = 10^7$.

7. Experimental Results

The results obtained from the studies outlined in Section 6 are presented in this section, beginning with the results obtained from the preliminary study employing the performance monitor.

7.1 Healthy System Operation

Figure 14 shows the recorded simulation data for the 'healthy' system scenario. Figure 15 shows the recorded performance metrics during this period.

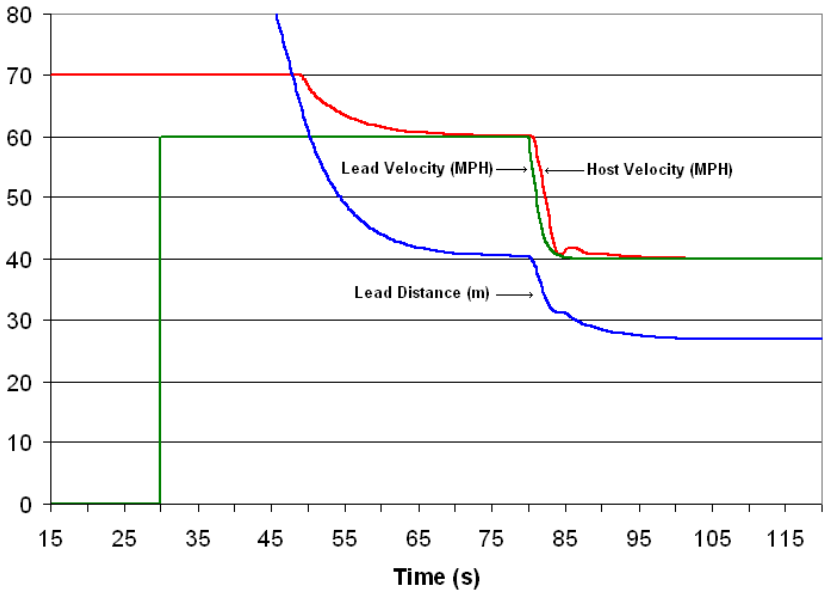

Fig.14. Healthy system scenario

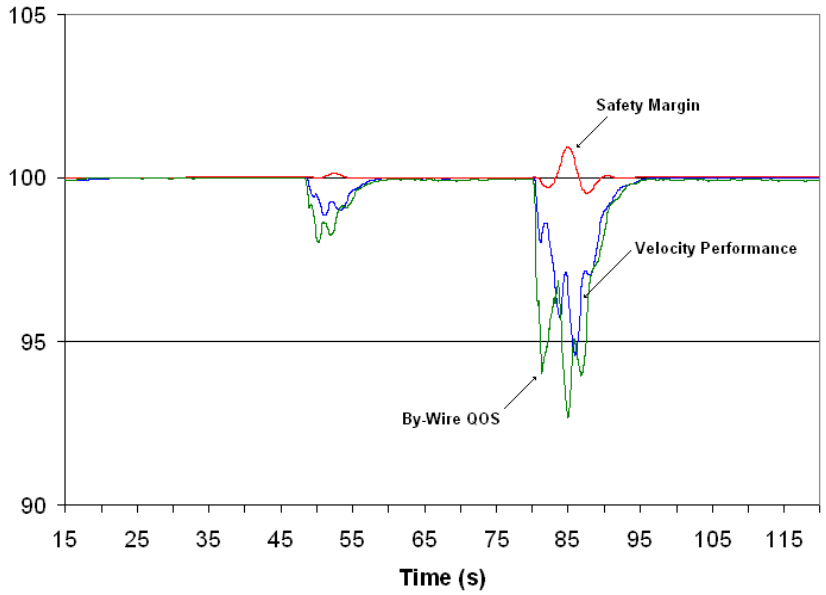

Fig.15. Healthy system performance

7.2 Braking Fault Operation

Figure 16 shows the recorded performance data for the 'braking' fault scenario, and Figure 17 shows the metrics that were recorded during this period.

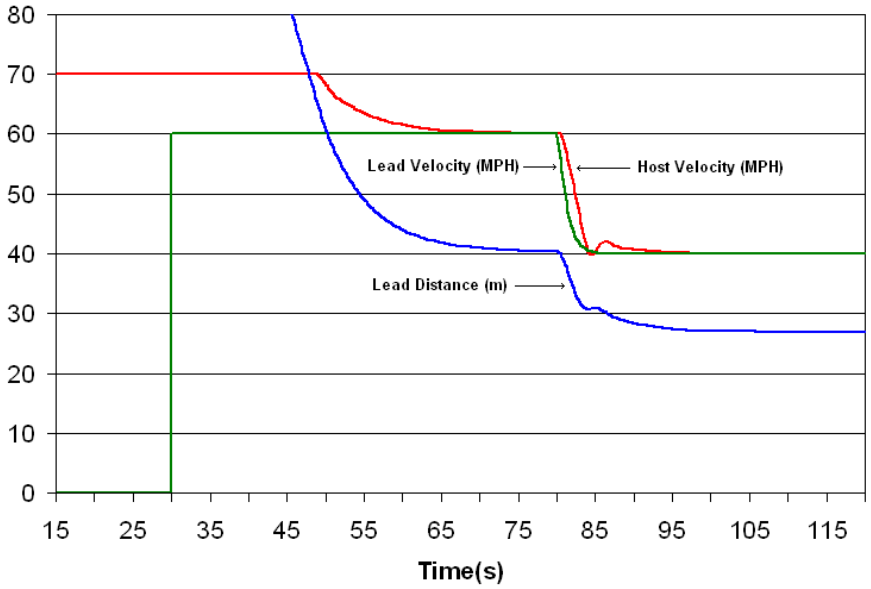

Fig.16. Brake fault scenario

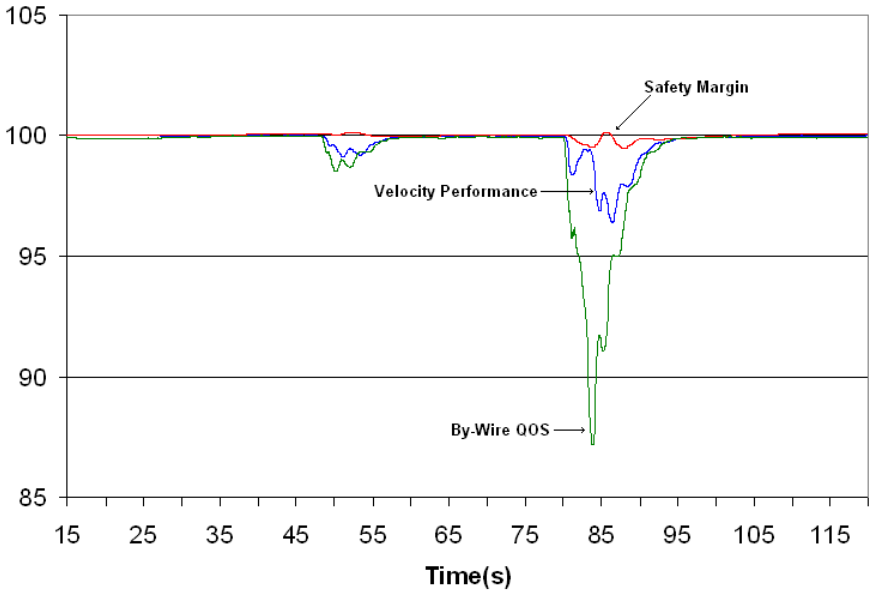

Fig.17. Brake fault performance

7.3 Timing Fault Operation

Figure 18 shows the recorded performance data for the ‘timing’ fault scenario, and Figure 19 shows the metrics that were recorded during this period. These results are discussed in the Section 7.5.

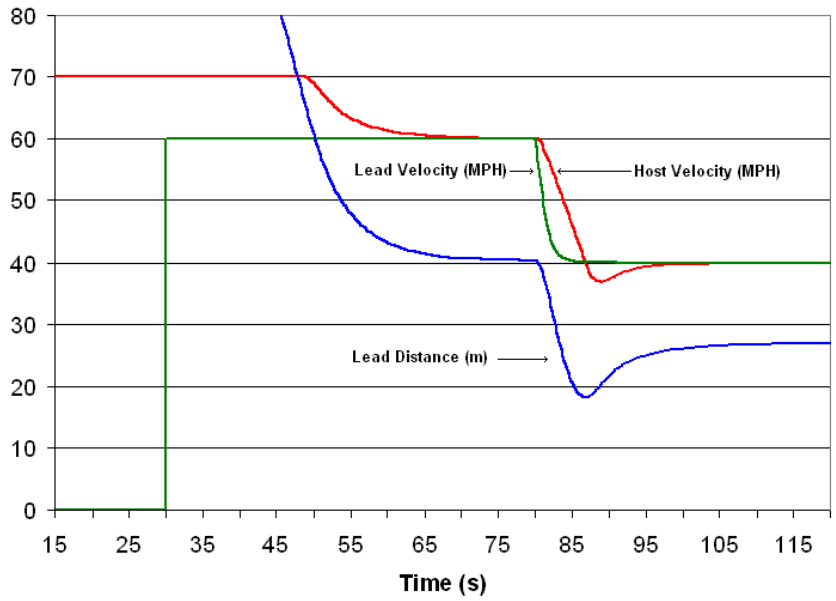

Fig.18. Timing fault scenario

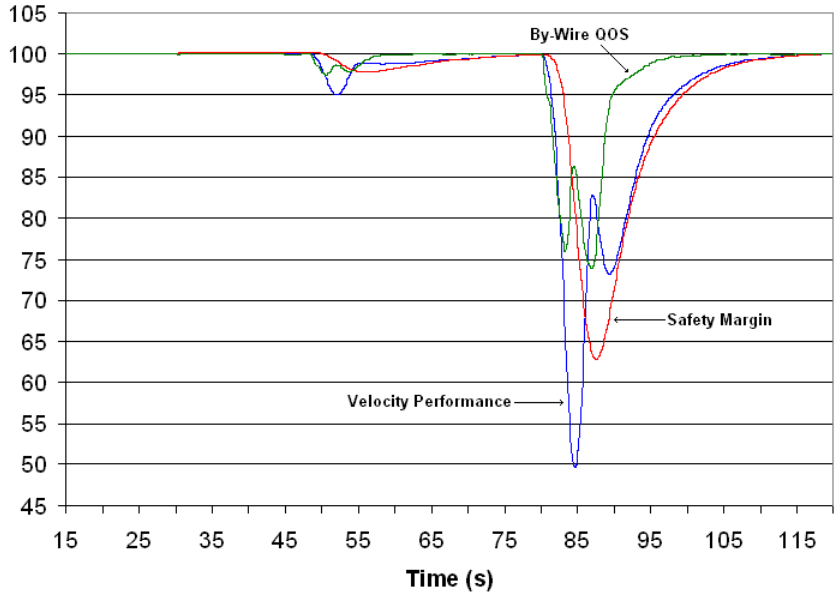

Fig.19. Timing fault performance

7.4 Fault Injection Testing

The results that were recorded during the fault-injection phases of the system testing are summarised in this section. Detailed results of the hardware and software fault injection are given in tabular form in Appendix A. For each system, the total number of test runs are listed, 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 III, where the probability that a hardware fault (P_h) or software transient fault (P_s) will result in a dangerous failure in each system is quoted. 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 I,

are also shown. This then allows the estimation of total dangerous failure rate for each system λ_d, using Equation 13; this failure rate was used to classify the SIL that each system lies in, as shown in Table III.

$$\lambda_d = (\lambda_h \times P_h) + (\lambda_s \times P_s) \tag{13}$$

Table III: Fault injection summary

System	λ_h(x10^{-6})	P_h	λ_s(x10^{-6})	P_s	λ_d(x10^{-6})	SIL
3-node TTC	7.51	0.9825	1.00	0.0247	7.40	SIL1
6-node TTC	14.0	0.3727	1.00	0.0300	5.26	SIL1
9-node TTC	20.5	0.1539	1.00	0.0209	3.18	SIL1
10-node TTC	22.7	0.0402	1.00	0.0480	0.96	SIL2
3-node ETP	7.51	0.9821	1.00	0.0643	7.44	SIL1
6-node ETP	14.0	0.4204	1.00	0.0648	5.96	SIL1
9-node ETP	20.5	0.2465	1.00	0.0629	5.12	SIL1
10-node ETP	22.7	0.1362	1.00	0.1030	3.20	SIL1

7.5 Discussion

Considering first the healthy system operation, it can be seen that during the entire scenario the performance metrics remain in the 90-100% range, indicating good control. Obviously the effects of noise and slight mismatches between the reference and actual models cause the metrics to deviate slightly from the 100% performance level; however this is to be expected. Table IV shows the minimum recorded metrics for all three systems; the healthy system remains well above the 90% level.

Table IV: Minimum Recorded Performance Levels

	Minimum Measured Metric		
System	Velocity	Safety	By-Wire QOS
Healthy System	94.544	99.519	92.647
Braking Fault	96.423	99.455	87.195
Timing Fault	49.638	62.819	73.956

For the braking-fault scenario, it can be seen that - despite the fact that 50% braking capacity has been lost - this can be tolerated by the ACC system (as the maximum deceleration level is bounded). From Table IV, the performance metrics for both the velocity control and safety margin perform no worse than for the healthy system; however, considering the By-Wire QOS metric, it

can be seen that the fault has been diagnosed. This is evident because this metric drops below the 90% margin, to a minimum level of 87.1%, reflecting the fact that the actual applied brake settings were out of the predicted range. Considering this metric as a relative (as opposed to absolute) measure, the minimum recorded level drops to 49.8%, correctly diagnosing a 50% loss of braking capacity.

Considering now the 'timing' fault scenario, it can be seen from the recorded metrics that this is the most severe of the recorded failure modes in the first experimental run; in fact the safety margin has been compromised by almost 38% compared to a healthy system, and the velocity control metrics indicate that the vehicle persistently deviates from the reference by more than 5 MPS during the scenario. These preliminary results highlight the effectiveness of the proposed CPM approach, in that the severity of different failure modes can be efficiently and automatically classified.

Considering now the fault injection testing results that were obtained, a number of trends may be observed. 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 not backup Master processors. However, in both TTC and ETP systems, the addition of a backup Master processor has a detrimental impact on this probability: it is suggested 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 in both systems this effect is somewhat 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.

However, in all cases the ETP systems displayed an increased failure rate over the TTC systems. This failure rate increase is more apparent as the number of nodes increases in this system, since as the complexity of the system increases there is more potential for 'things to go wrong'. It is hypothesised that the differences in failure rate can be attributed to two main reasons:

1) The increased determinism of the TTC system positively increases the effectiveness of the hardware and software fault tolerance mechanisms[7].

2) The increased resource usage and complexity of the preemptive design (context switch, need for locks etc) greatly reduces its resilience to software transients[8].

This difference is large enough to encompass a whole SIL in the 10-node systems; however the differences in the remaining systems are still large enough to advocate the use of the TTC approach in safety-related designs of this nature.

8. Conclusions

In this paper, a methodology for the automated assessment of performance and dependability of embedded control systems has been presented. It has been shown how the dynamic performance of safety-critical control systems can be automatically monitored during testing and verification procedures using CPM, and how large-scale fault-injection testing can be driven by RET. Although the selection of a suitable reference model and particular choice of recorded metrics during the testing procedures is heavily dependant on the given application, it has been shown experimentally that in this particular application the selected metrics give an excellent measure of the system operation, and are capable of diagnosing the severity of the system failure modes.

Although measurement of dependability and functional safety cannot simply rely on a single methodology or technique, these results suggest that - with appropriate adjustments - model-based methodologies such as CPM and RET may prove to be an extremely useful adjunct to existing techniques, and may be readily integrated into existing HIL testing equipment and automated testing programmes. In this particular case, failure modes could be automatically classified by their severity in affecting the controllability of the vehicle; this has allowed us to provide empirical data regarding system reliability and fault tolerance (which are closely related to safety) of ETP and TTC architectures to prospective designers of embedded control systems.

However, it is hoped that the results presented in this case study may stimulate further discussions and research in this area. The basic 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

[7] This was concluded as ‘replica determinism’ is a key requirement for effective management of fault-tolerance (Kopetz 2000).

[8] This is in agreement with results from a preliminary study (Short et al. 2008a).

empirical evidence related to the application of the different design paradigms to a wider range of embedded-system application areas.

Acknowledgements

The project described in this paper was supported by the Leverhulme Trust grant (F/00212/D). Initial versions of the TTC system test results presented in Section 7 were described in a different context in an earlier paper (Short & Pont 2008).

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.

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.

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.

Baraket, Z., Fancher, P.S., Peng, H., Lee, K. and Assaf, C.A. (2003). Methodology for Assessing Adaptive Cruise Control Behavior, *IEEE Transactions on Intelligent Transportation Systems*, Vol. 4, No. 3, pp. 123-131, September 2003.

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

Bosch (1991). *CAN Specification 2.0*, Robert Bosch GmbH, Germany, 1991.

Broekmann, B. and Notenboom, E. (2002). *Testing Embedded Software*, Addison-Wesley, 2002.

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.

Buttazo, G. (1997). Hard real-time computing systems: predictable scheduling algorithms and applications. Kluwer Publishers, Norwell, MA., 1997.

Claesson, V., Ekelin, C., Suri, N. (2003). The event-triggered and time-triggered medium-access methods. In *Proc. IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, 2003.

Clarke, E., Kroening, D. and Lerda, F. (2004). A Tool for Checking ANSI C Programs. In: *Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2004)*, pp. 168-176, 2004.

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

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

Fang, J. (2006) The design of a pre-emptive scheduler for the C167 Microcontroller. Technical Report ESL 06/01, University of Leicester, 2006.

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

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.

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.

Huang, B. and Shah, S.L. (1999). *Performance assessment of control loops*, Berlin: Springer-Verlag, 1999.

Isermann, R. (2008). Mechatronic systems-Innovative products with embedded control. *Control Engineering Practice*, Vol. 16, pp. 14-29, 2008.

Isermann, R., Schwarz, R. and Stoltz, S. (2002). Fault-tolerant drive-By-Wire Systems, *IEEE Control Systems Magazine*, Vol. 22, No. 5, pp. 64-81, 2002.

ISO (2003). *Adaptive Cruise Control Systems – Performance Requirements And Test Procedures*, International Standards Organization, Standard 15622, Geneva, Switzerland, 2003.

Jelali, M. (2006). An overview of control performance assessment technology and industrial applications, *Control Engineering Practice*, Vol. 14, pp. 441-466, 2006.

Kopetz, H. (1991). Event-Triggered Versus Time-Triggered Real-Time Systems. *Lecture Notes in Computer Science*, Vol 563, pp 87-101. Springer-Verlag, Berlin/New York, 1991.

Kopetz, H. (2000). A Comparison of CAN and TTP. *Annual Reviews in Control*, Vol. 24, pp. 177–188, 2000.

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, 1999.

Levenson, N.G. (1995). *Safeware: System Safety and Computers*, Addison Wesley Publishing, 1995.

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, 2003.

Lonn, H., and Axelsson, J. (1999). A comparison of fixed-priority and static cyclic scheduling for distributed automotive control applications. In: *Proc. 11th Euromicro Conf. on Real-Time Systems*, York, UK, 9–11 June 1999, pp. 142–149.

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.

MISRA (1994). Development guidelines for vehicle-based software, *Motor Industry Software Reliability Report*, November 1994.

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, Released October 2004.

Phytec (2003). *phyCORE 167CS Hardware Manual*, Phytec, April 2003.

Pont, M.J. (2001.) *Patterns For Time Triggered Embedded Systems*, Addison Wesley, 2001.

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.

Qin, S.J. (1998). Control performance monitoring – a review and assessment, *Computers & Chemical Engineering*, 23, 173-186, 1998.

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

Scheler, F. and Schröder-Preikschat, W. (2006). Time-Triggered vs. Event-Triggered: A matter of configuration? In: MMB Workshop Proceedings GI/ITG Workshop on Non-Functional Properties of Embedded Systems, Nuremberg, Germany, 27-29 March, pp. 107-112, 2006.

Short, M, Pont, M.J and Fang, J. (2008a). "Exploring the Influence of Preemption on Dependability in Time-Triggered Embedded Systems: a Preliminary Study". Paper Submitted to the *20th Euromicro Conference on Real-time Systems (ECRTS 2008)*, Prague, Czech Republic, July 2008.

Short, M. and Pont, M.J. (2005). Hardware in the Loop Simulation of Embedded Automotive Control Systems, in *Proc. 8th International IEEE Conference on Intelligent Transportation Systems (ITSC2005)*, September 13-16, Vienna, Austria, 2005.

Short, M. and Pont, M.J. (2007). 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. (2008). Assessment of High-Integrity Embedded Automotive Control Systems using Hardware in the Loop Simulation. *Journal of Systems and Software*, Article in press, doi:10.1016/j.jss.2007.08.026.

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

Short, M., Pont, M.J. and Huang, Q. (2004b) *Simulation of vehicle longitudinal dynamics*, Technical report ESL 04/01, Embedded Systems Laboratory, University of Leicester, 2004.

Short, M.J., Fang, J., Pont, M.J. and Rajabzadeh, A. (2007b) "Assessing the impact of redundancy on the performance of a brake-by-wire system". *SAE Transactions: Journal of Passenger Cars (Electronic and Electrical Systems)*, Vol. 115, no. 7, pp. 331-338.

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

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, 1997.

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, 1996.

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.

Wong, K & Joyce, J. (1998). Refinement of Safety-Related Hazards into Verifiable Code Assertions. In: *Proceedings of SAFECOMP 1998*, pp. 345-358, 1998.

Xu, J. and Parnas, D.L. (2000). Fixed Priority Scheduling versus Pre-Run-Time Scheduling. *Real-Time Systems*, Vol. 18, pp. 7-23, 2000.

Figure Captions

Table Captions

Appendix A – Summary of Hardware and Software Fault Injection Results

System	Hardware Faults Injected				Recorded Failures		Estimated Values	
	Total	Node	CAN Link	CAN Bus	Dangerous	Safe States	$\lambda_h(x10^{-6})$	P_h
3 Node TTC	684	304	277	103	672	12	7.51	0.98245614
6 Node TTC	1151	562	509	80	429	722	14.02	0.37271937
9 Node TTC	1715	882	756	77	264	1451	20.53	0.15393586
10 Node TTC	1865	940	850	84	75	1790	22.7	0.04021448
3 Node ETP	670	321	266	83	658	12	7.51	0.98208955
6 Node ETP	1156	576	491	89	486	702	14.02	0.42041522
9 Node ETP	1728	869	753	106	426	1302	20.53	0.24652778
10 Node ETP	1894	964	822	108	258	1636	22.7	0.13621964

System	Software Faults Injected		Recorded Failures		Estimated Values	
	Transients	Total Bits Flipped	Dangerous	Safe States	$\lambda_s(x10^{-6})$	P_s
3 Node TTC	6080	92807	150	16	1	0.02467105
6 Node TTC	6135	183765	184	81	1	0.02999185
9 Node TTC	6132	277787	128	136	1	0.0208741
10 Node TTC	6191	308665	297	125	1	0.04797286
3 Node ETP	6139	92949	395	38	1	0.06434273
6 Node ETP	6389	192096	414	507	1	0.06479887
9 Node ETP	6691	303913	421	1063	1	0.06292034
10 Node ETP	6815	343928	702	1171	1	0.10300807