iCC 1996

3rd international CAN Conference

in Paris (France)

Sponsored by

Motorola Semiconductor
National Semiconductor
Philips Semiconductors

Organized by

CAN in Automation (CiA)
international users and manufacturers group
Am Weichselgarten 26
D-91058 Erlangen
Phone +49-9131-69086-0
Fax +49-9131-69086-79
Email:headquarters@can-cia.de
URL: http://www.can-cia.de

Variability of CAN Network Performance

Bhargav P. Upender ***Alexander G. Dean***
barg@utrc.utc.com *adean@utrc.utc.com*
United Technologies Research Center
411 Silver Lane, MS 55
East Hartford, CT 06108

Abstract

Designers of soft real-time systems often ignore worst-case behavior analysis when designing their systems. We have developed a discrete-event model of Controller Area Network (CAN) to assess performance under both worst-case and normal conditions. The analysis revealed that many network events and attributes can lead to message serialization, which causes large network delays typical of worst-case behavior. This transient network behavior has serious implications on the application performance. In this paper, we present these results and show how the actual performance of the system varies over time between the normal and worst-case scenarios.

Introduction

Controller Area Network (CAN) is well suited for real-time control applications due to its predictable medium access approach, built-in collision avoidance, and global prioritization features. Several papers present analytic assessments of worst-case message response times [Tin94, Tin94-2]. In our application, as in many soft real-time applications, designing to worst-case is cost prohibitive. Consequently, we developed a CAN model to predict network performance and to make design decisions for our application. After briefly discussing the modeling environment and application characteristics, we present some interesting results on network behavior.

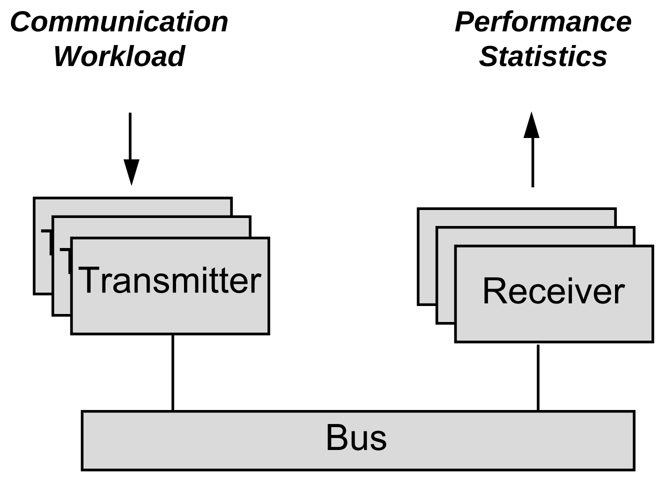

Figure 1: Block diagram of CAN model

Modeling Environment

The CAN model was developed in SES/Workbench, a popular generic discrete-event simulation tool. The simulation environment includes a complete set of queuing disciplines, priority methods, and statistics gathering tools for rapid model development [SES96, Rum95]. The block diagram in Figure 1 shows the key modules in the model. Using a communication workload description table, the transmitter modules periodically inject messages into the bus module. The bus module arbitrates for bus access and simulates message transmission. The receiver modules collect the messages and maintain performance statistics.

Application Characteristics

The application contains 12 communicating nodes, with eight of the nodes executing the same software (resulting in similar traffic patterns). We derived the message traffic by instrumenting a legacy system. Many of the messages are inherently periodic; the few event-driven messages were given approximate periods based on the expected peak system operation conditions. Table 1 summarizes the application's communication characteristics. The workload will grow over time as features and nodes are added. We simulated the future workload by multiplying the generation frequency of each message type with a scaling factor.

Number of nodes	12 nodes
Message types	50
Data rate	200 msgs/sec
Typical deadlines	100 ms
Shortest deadlines	10-20 ms

Table 1: Workload characteristics

Results & Analysis

Bus Speed Tradeoffs

Selecting a bus speed is a tradeoff between capacity for growth and application performance. Low bus speeds can lead to missed deadlines, performance bottlenecks, and limited growth space. However, high speeds can reduce maximum bus length, amplify susceptibility to noise, and increase demands on both the network interface and the application.

High bus speeds can place severe demands on the applications during bursty traffic conditions. The demands placed on the application by a fast bus should be determined early in the design process, as otherwise they may not show up until prototypes have been built [Dea96]. During heavy bursts of traffic, messages arrive back-to-back and interrupt the processor. The faster the bus, the quicker the messages will arrive, and the faster the host processor must suspend the application task and unload the new messages from the controller (to prevent buffer overwrite). The solid line in Figure 2 demonstrates how the worst-case interrupt rate grows as the bus speed increases. Similarly, the dashed line shows the inverse, which is the amount of time the host processor has to clear the receive buffer. At 100kbps, the host processor has 660μS to clear the receive buffer. At 1000kbps, the application has only 66μS. Multiple mailbox-type receive buffers available on the CAN controllers will only decrease the chance of overwrites, but will not eliminate them. If the CAN controllers supported standard receiver queues (first-in first-out) instead of mailbox buffers, the application would not have to react as rapidly.

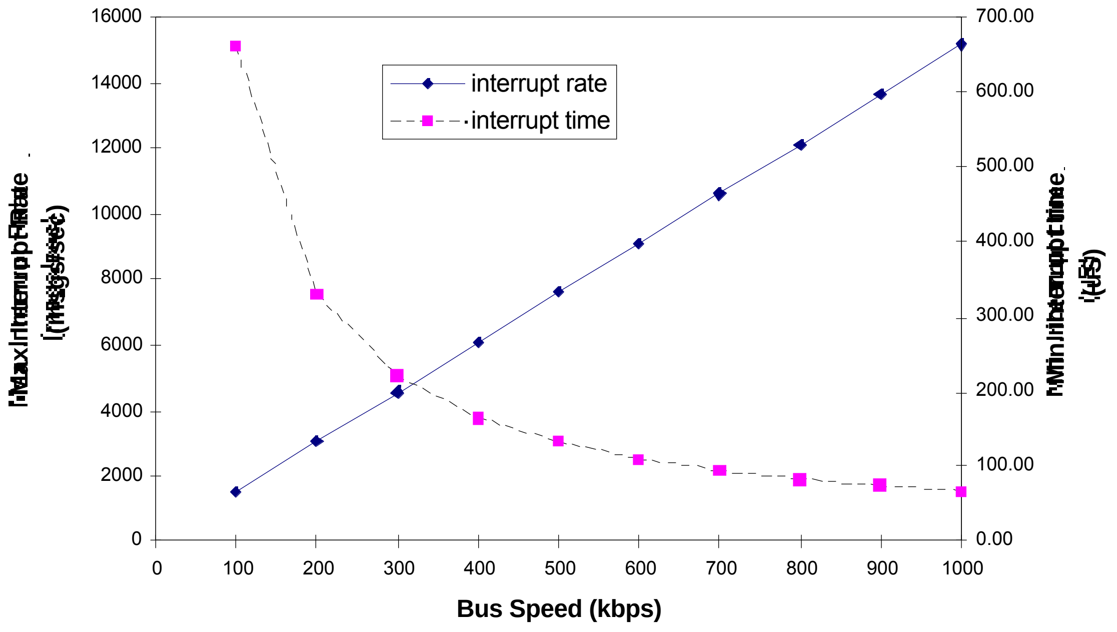

Figure 2: Maximum interrupt rate and minimum interrupt time as the bus speed varies

Figure 2 does represent worst-case conditions. Unfortunately, worst-case is not as rare as expected. Under bursty conditions, the system experiences worst-case behavior. There are many events which induce message bursts. These events and their impact on the application are discussed later in this paper.

Bus Speed Selection

We selected the bus speed for our application using simulation results. Two main factors were considered in selecting the bus speed: remaining capacity and missed deadlines. As a guideline, we add a liberal growth margin when designing a network. A network is an infrastructure component: as the infrastructure improves, it will be used more often. In collision-free protocols like CAN, a growth margin of five is adequate for us. Based on our experience, a growth margin of ten is recommended for collision-based protocols to compensate for inaccessability of the bus during collision detection and resolution.

Our application’s baseline workload requires minimum bus speed of 21kbps. Using the above guideline, our network should operate at minimum speed of 105kbps. Figure 3 shows running at 125kbps will result in 19% excess capacity (for five times the current workload). Therefore, one criterion for bus speed selection is satisfied with 125kbps bus speed.

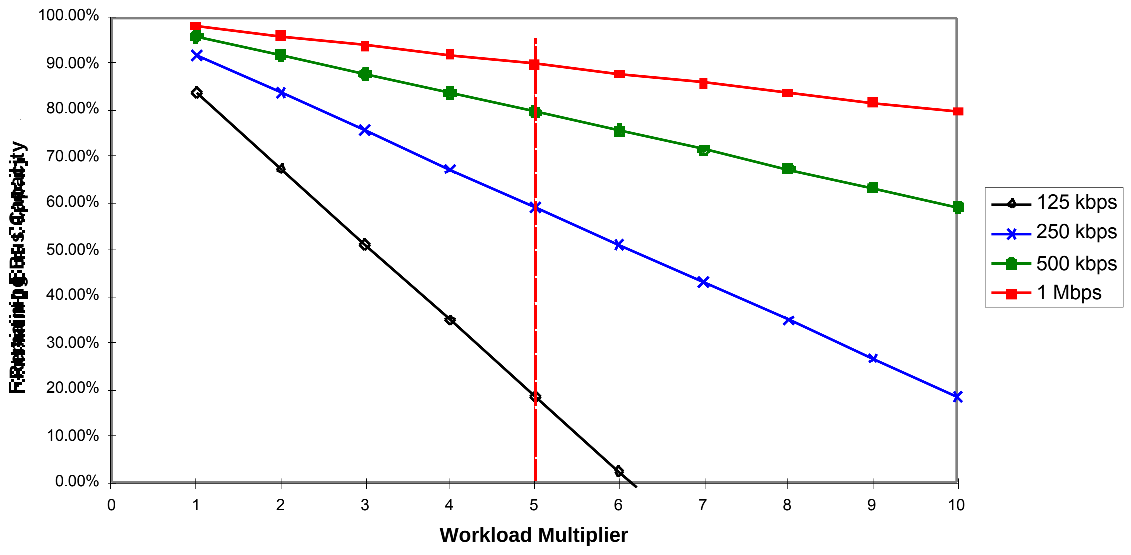

Figure 3: Remaining bus capacity as the workload increases

However, at 125kbps, 12% of the messages in the simulation missed their deadline (see Figure 4). At 250kbps, the missed deadlines are near zero at 5x message traffic. Based on both missed deadlines and remaining capacity, a bus speed of 250kbps was selected for our application.

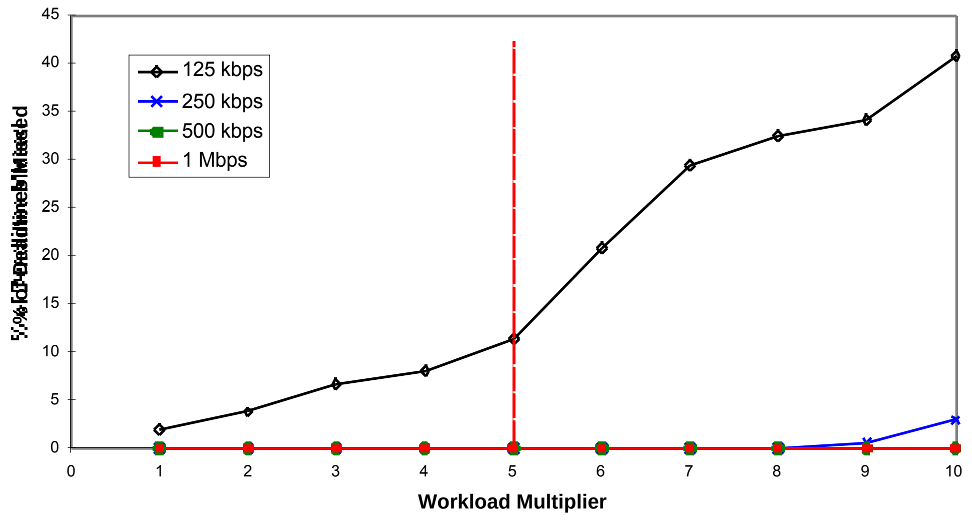

Figure 4: Percent of the missed deadlines as the workload increases

Assessing message delays

Figure 5 shows some of the delays a message encounters as it travels from the source node to the destination node. The model assumes task switching delays are minimal and other tasks do not block or preempt the communication routine. Some delays can be analytically derived while others require an executable model for practical evaluation.

The interrupt service routine (ISR) and message transmission delays can be calculated analytically. For example, the ISR which handles data transfer between the host processor and the CAN controller takes approximately 40 to 50μs on an Intel 386/25Mhz processor. This delay is derived by writing assembly code and counting processor cycles. Similarly, the message transmission delay (time taken to send the message on the wire) can be calculated to be between 200 to 400μs (varying with data length) on a 250kbps bus.

The average medium access control (MAC) delay is hard to calculate for an arbitrary workload so we use an executable model to derive this value. For our application, the MAC delay varied drastically from 0 to 40,000μs. As this delay is significantly larger than delays caused by other factors; it is important to understand its root causes.

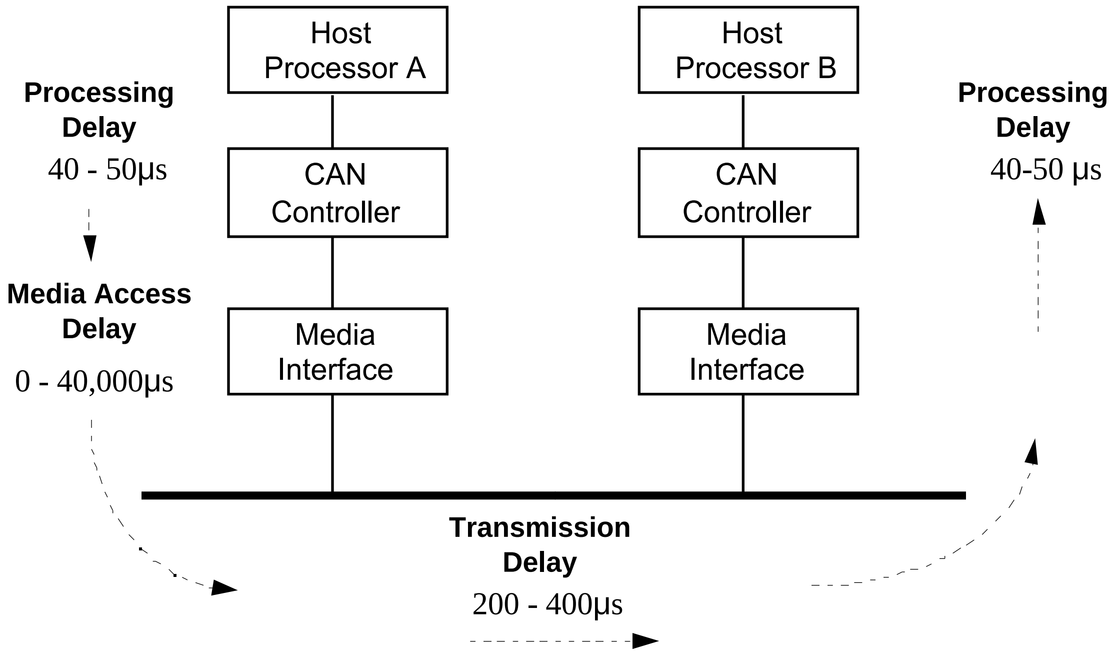

Figure 5: Some of the delays associated with application to application communication

Synchronization and Medium Access Delay

MAC delay depends on message priorities, message generation times, synchronization, bus bandwidth, and other factors. In particular, message synchronization can lead to large delays [Upe94]. Synchronization is the degree of correlation between message generation times among all the nodes in the network. In the worst-case scenario, all nodes are synchronized to a global clock (with negligible clock drift). Under this scenario, periodic message generators will align with other periodic generators and request bus access at similar times. As many messages are contending for the bus, the highest priority message preempts those of lower priority, which in effect wait in a global priority queue for bus access. This waiting is called serialization, and the network experiences a burst of back-to-back messages. As mentioned earlier, these traffic bursts place a high demand on the host processor, forcing frequent interrupts and application preemption.

The model provides an adjustable message synchronization by starting all message generators at time zero with no phase difference; this provides worst-case synchronization scenario that is hard to create without abstract models (i.e. hard to induce in a laboratory setup). The messages are then delayed by a uniformly distributed random jitter time which is bounded by a user-specifiable fraction of the message period.

The solid line in Figure 6 shows the average medium access delay varying from 18ms to 32ms as the workload is scaled from 1x to 10x on a fully synchronized network. The dashed line shows the average MAC delay when message periods are dithered by 10%. With only this moderate amount of jitter in the message generators, the mean MAC delay at baseline (1x) traffic has fallen from 17 ms to just a few microseconds. This shows the network has significant headroom; delays become significant only when the traffic is highly synchronized.

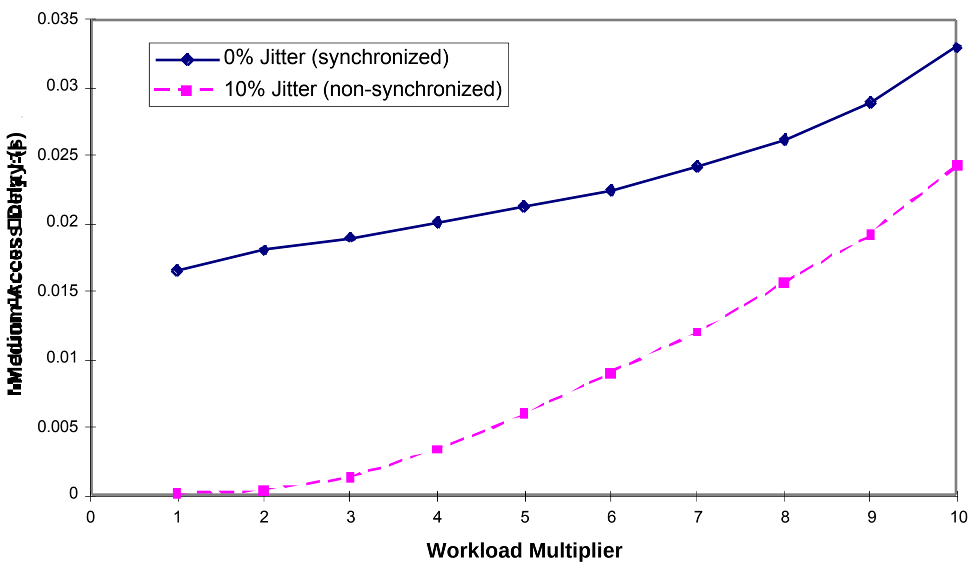

Figure 6: MAC delay for synchronized vs. non-synchronized nodes

In an actual implementation the level of synchronization is hard to predict and control. The nature of the communication patterns, the power-up sequence, clock tolerances, and initialization routines affect the level of synchronization. One can argue that a network with synchronized message sources represents a worst-case scenario which is not likely to happen in a real implementation. However, we found that even if these sources of synchronization are controlled, many other sources can cause serialization which results in behavior similar to worst-case scenario.

Sources of Serialization

- *Noise bursts:* a burst of noise from the environment (such as motor switching on) can corrupt data transmission and disable the network momentarily. This can lead to error messages, retransmissions, extra acknowledgments, and so forth.

- *Command/response broadcast:* a Remote Transmission Request (RTR) broadcast, or any similar control structure, can induce several responses simultaneously, leading to bursts of messages.
- *Heavy bus traffic:* when the bus is heavily loaded, low priority messages will queue up.
- *Long messages:* a long message must be fragmented into small eight-byte messages (dictated by CAN's data length limitation) and sent on the network. This will create a burst of messages. In fact, if the long message has high priority, many other lower priority messages can queue up.
- *Clock drift:* as periodic message generators drift, they can align in time cause near-simultaneous generation of many messages.

All of these conditions can lead to message serialization and bursts of back-to-back messages which require an application to react quickly. Serialization leads to larger medium access delays and more missed deadlines. This results in the network sporadically exhibiting worst-case behavior similar to the synchronized network.

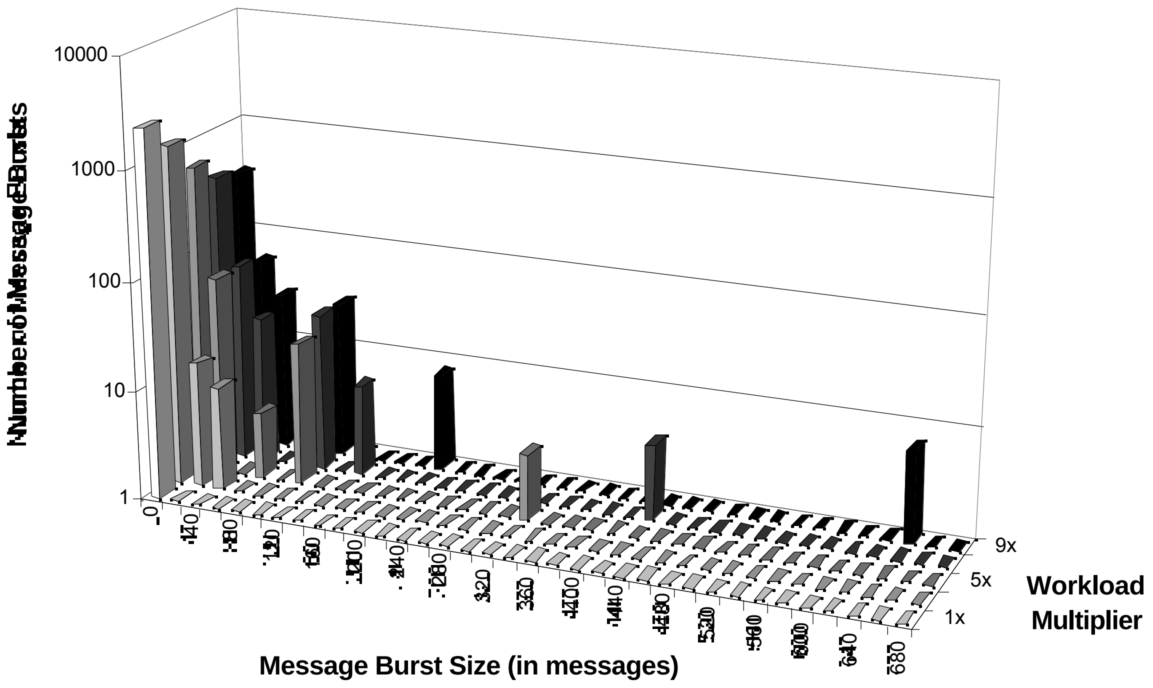

Figure 7: Message Burst Length Distribution vs. Workload

Effects of Serialization on Application Design and Performance

The histogram in figure 7 shows how often bursts of messages happen based on burst length and bus traffic during a ten second loosely synchronized simulation run. The x-axis measures burst length, which is the number of messages sent back-to-back without any idle bus time. The y-axis measures bus traffic based the workload scaling factor. The z-axis measures the number of bursts of a given length detected during the simulation. With the baseline workload, all bursts are shorter than 20 messages. However, scaling the workload leads to much longer bursts, some longer than

640 messages. During any burst, the controller must handle incoming messages at the maximum rate supported by the bus. If an application is likely to encounter long message bursts, it makes sense to minimize the time needed to handle each message. One optimization is to use a software-implemented queue to hold new messages between unloading from the CAN controller and processing by the application. Another is to choose a CAN controller with good message filtering capabilities, allowing the application to ignore messages for other nodes. This requires a strategy for allocating message identifiers. A CAN interface with software-based filtering will be overwhelmed during message bursts compared with filtering based on hardware.

Even “soft” real-time system designers should carefully study the interaction of the network with the application to minimize performance problems. Fortunately, since the protocol’s contention resolution time overhead is zero, CAN is better suited to handle message serialization than many other protocols. However, this efficiency can lead to much higher sustained message arrival rates, complicating the system design. Connecting a processing node to a network requires a more through analysis of task deadlines.

Summary

We have developed a model of communication system based on Controller Area Network to predict performance, make design decisions, and assess application behavior. In selecting the bus speed for our system, we relied on two parameters: remaining capacity and missed deadlines. While increasing network speed enhances network performance, it can impose tighter constraints on the application. Selecting the proper bus speed therefore involves a tradeoff between network and application performance. In assessing the response times, we have shown how message serialization can lead to large delays and reduced CPU performance. We also have identified many sources that can lead to message serialization. This serialization can induce transient worst-case behavior more often than otherwise expected.

References

[Dea96] Dean, A., & Upender, B. P., “Embedded Communications: What You Don’t Know Will Hurt You”, Embedded Systems Conference, September 1996.

[Rum95] Rumpf, O., & Gassenhuber, A. “Generation of Realistic Communication Scenarios for the Simulation of Automotive Multiplex Systems”, Society of Automotive Engineers, 1995, Paper No. 950294.

[SES96] Scientific and Engineering Software, Inc., *SES/Workbench Sim Language Reference*, Austin, Texas, 1996. SES/Workbench is a trademark of SES Inc.

[Tin94] Tindell, K., & Burns, A. “Guaranteeing Message Latencies on Control Area Network”, First International CAN Conference, Mainz, Germany, Sep. 1994.

[Tin94-2] Tindell, K., Hansson, H., & Wellings, A. J. “Analyzing Real-Time Communications”, Real-Time Systems Symposium, San Juan, Puerto Rico, December 1994.

[Upe94] Upender, B. P., “Analyzing Real-Time Characteristics of Class C Communication in CAN through Discrete Event Simulations”, Multiplexing and Fiberoptics, SP-1012, Society of Automotive Engineers, Feb. 1994, Paper No. 940133, pp. 25-34.