VOLVO TECHNOLOGY REPORT VOLVO

LENNART CASPARSSON
ANTAL RAJNAK
KEN TINDELL
PETER MALMBERG

Volcano – a revolution in on-board communications

The electrical system in the new Volvo S80 includes a dozen or so electronic control modules employing multiplex buses to exchange large quantities of data. A new data communications concept known as Volcano was developed to meet the requirements of the new system. Embodying several revolutionary new features, Volcano has attracted international attention and has enabled Volvo to assure the quality of data communications in the new model at an early stage of its development, while providing an ideal platform for future expansion.

In the last twenty years, the computer industry has learned the lesson that software functions tend to proliferate, increasing general awareness of the necessity of high-level languages and the vital importance of sound software engineering principles. In this respect, the automotive industry is about to reach ’crisis point’ due to the growing demand for software-based systems in vehicles.

Challenges of platform concept

The platform concept presented the Volcano project team with a number of specific challenges:

- Software complexity is expected to increase by a factor of over 25 compared with our most advanced products to date. This had to be accommodated within the framework of the project.
- Real-time problems had to be solved.
- Signalling requirements are expected to grow by 7–10% annually.

CAN (Controller Area Network) is an industry-standard solution for interconnecting microprocessors relatively cheaply by means of a single broadcast bus. Although multiplexed bus communication makes all of the signals in the vehicle easily accessible to all microprocessors, the potential for uncontrolled interaction between software components leads to well-recognised problems of impaired software reliability and higher cost. Because of this, the main design objective of the Volcano project was to control software complexity, as the only means of maintaining and improving the reliability and dependability of automotive software.
A shared broadcast bus greatly facilitates the addition of desired functionality. Electronic control modules (ECMs) can not only be added easily, but can also exchange data simply and inexpensively (by enabling functions to be added ’just’ as software). However, a higher level of functionality involves more software and even greater complexity. In the last thirty years, it has been found that certain mainstream computing techniques are needed to manage large, complex, software systems. Two of these – abstraction (where unnecessary information is hidden) and composability (which is

based on the premise that if inherently correct components are added together. the resultant system will also be correct) - are of major importance in this context. Volcano was designed to meet the needs of vehicle manufacturers for composability as well as those of suppliers for the abstraction required to manufacture a single, 'standard' ECM.

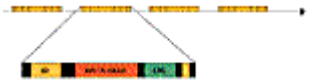

Fig. 2

S80 networks

The electrical system in the S80may be described as a distributed real-time solution - distributed in the sense that many functions are implemented as the sum of a number of software programs executed by a number of different, interacting ECMs, and real-time because most of the functions in question require firm timing guarantees.

Technical concepts

CAN

> CAN communications are based on the transfer of packages of data known as 'frames'

At the beginning of the 1980s, the need for signal exchange between the growing number of on-board ECMs highlighted the need for a dedicated automotive communications solution. Neither point-to-point serial links nor dedicated (one wire per signal) links – the solutions available at that time – were adequate for the industry's needs. This led to the development of a new solution known as CAN (Controller Area Network) by a team at Robert Bosch GmbH in 1986. This made it feasible to interconnect large numbers (as many as tens or even hundreds) of ECMs in the form of a network to communicate with each other.
CAN communications are based on the transfer of packages of data known as 'frames'. The two main elements of a frame are the identifier (ID) and data area (DA). Containing up to 64 bits (8 bytes) of data, the DA contains the useful information to be transmitted between the ECMs.
CAN uses identifiers for two purposes - to distinguish between different frames on the bus and to assign relative priorities to those frames. Problems can arise when these two purposes conflict. Disregarding the true priorities of the frames can have very inefficient and potentially dangerous results. The importance of the ID as carrier of priority information has been neglected by the industry until now.

The method is based on a number of simple assumptions regarding frame transmission:

- A given frame, m, cannot be generated more than once every T_m time units.
- Once generated, frame m cannot take longer than J_m to be queued for transmission by the CAN controller.
- Frame m has a bounded size (s_m bytes).
- An error function, E(t), states the maximum expected overhead due to errors and retransmissions on the bus during an interval, t.
- The CAN controller software drivers ensure that the highest-priority frame at the module is entered into arbitration whenever this takes place on the bus.
- All frame identifiers are known.

These assumptions enable the worst-case latency of each frame (denoted R_m) to be computed. R_m is measured from the instant frame m is entered in the CAN device to the time it is transmitted correctly on the bus.
It is frequently necessary to assign a deadline to each frame and to work out the latencies to determine if the deadlines have been met. Theory indicates that the optimum choice of priorities (i.e. the optimum ordering of CAN identifiers) is 'deadline monotonic'; in other words, the frame with the shortest deadline is assigned the highest priority, that with the next shortest deadline the next highest priority, and so on. Volcano employs this ordering to achieve optimum real-time performance.

“The queuing time is measured from the instant that the frame is queued to the start of successful arbitration”

The basic equations for the worst-case latencies are given below. Equation (1) defines the worst-case latency of a given frame, m, as the sum of the queuing and transmission times:

$$R_m = t_m + C_m \quad (1)$$

The queuing time is measured from the instant that the frame is queued to the start of successful arbitration. The transmission time is the actual time taken to transmit the frame on the bus. This can be determined using the maximum size of the frame and a knowledge of the maximum number of stuffing bits that can be inserted into the bit stream during transmission. The transmission time, C_m, is given by the following equation:

$$C_m = \left(\left\lfloor \frac{34 + 8s_m}{4} \right\rfloor + 47 + 8s_m \right) \tau_{bit} \quad (2)$$

The term, 's_m' denotes the bounded size of frame m in bytes. The term 't_{bit}' is the bit time of the bus (for example, this is 1 ms on a bus running at 1 Mbit/s).
To determine the maximum queuing delay, it is necessary to know how long a lower-priority frame (i.e. one with a higher identifier number) can hold the bus before arbitration starts. This is equivalent to the maximum transmission time of the largest frame of lower priority. The maximum time by which higher-priority frames may 'jump the queue' and be transmitted ahead of frame m while this is still queued must also be known. This is given by:

$$\sum_{\forall j \in hp(m)} \left\lceil \frac{t_m + J_j + \tau_{bit}}{T_j} \right\rceil C_j \quad (3)$$

where hp(m) is the set of frames of higher priority than frame m, t_m is the longest time that frame m is queued before winning arbitration, and J_j is the queuing jitter of frame j (the maximum time between notional generation of the frame and the instant that it can enter into arbitration).
Finally, the time attributable to handling errors (transmission of error frames and retransmission of higher-priority frames) must be accounted for. This is done with the

aid of an 'error function', E(t), defined as the time taken to retransmit frames containing errors within an interval of duration t (if this function is always zero, the bus is assumed to be perfect).
Combining these times gives the maximum queuing time:

$$t_m = B_m + \sum_{\forall j \in hp(m)} \left\lceil \frac{t_m + J_j + \tau_{bit}}{T_j} \right\rceil C_j + E(t_m) \quad (4)$$

Since this equation has no simple solution, a recurrence relation must be formulated. An initial value of $t_m = 0$ is sufficient for this purpose.

$$t_m^{n+1} = B_m + \sum_{\forall j \in hp(m)} \left\lceil \frac{t_m^n + J_j + \tau_{bit}}{T_j} \right\rceil C_j + E(t_m) \quad (5)$$

Equations 1, 2, and 5 comprise the timing analysis of the CAN bus. This analysis has been used by Volvo both to verify and design the CAN system in the S80.

CAN controllers differ

> The worst case latency for every frame in the system is calculated and compared with the required deadline for each frame

One of the basic assumptions listed above is that the CAN controller always enters the queued frame with the highest priority into arbitration. The worst-case latency of a frame cannot be determined by analysis unless this is true. In fact, the worst-case latency will usually also be much higher and examples in which a periodic system with a bus load of only 11% fails to meet its timing requirements can be described. Unfortunately, most simple CAN controllers do not fulfil this assumption.
The Intel 82527 – the most widely used CAN controller - is slightly more sophisticated than the simpler types, having a number of transmit buffers arbitrated internally by buffer number. If the system is configured so that the highest-priority frame is placed in the first slot, the second highest in the second slot, and so on, the analysis will be applicable. However, since the 82527 - like other, similar controllers - has a maximum of just 14 transmit buffers, an ECM is limited to sending a maximum of 14 different frames. Being a standalone chip, the 82527 has the further disadvantage of requiring more board space, adding complexity and cost.
An ideally designed CAN controller should be provided with at least three transmit buffers, arbitration between these being carried out by frame priority (i.e. identifier) (Fig. 3).
To permit analysis, a CAN controller must transmit an uninterrupted stream of messages without releasing the bus between any two. In controllers of this type, arbitration for the bus will take place immediately following transmission of the preceding message and the bus will be released only if arbitration is lost. Furthermore, the internal message queue within the CAN controller must be organised so that if more than one message is ready for transmission, that with the highest priority will enter arbitration.

Fig. 3

The above behaviour cannot be achieved with a single transmit buffer, which must be reloaded immediately when the previous message has been sent. This process takes a certain time and must be completed within the inter-frame sequence to enable an uninterrupted stream of messages to be transmitted. Even if this is feasible at limited CAN bus speeds, it requires fast CPU response to transmit interrupts.
Although a double buffer scheme would decouple transmit buffer reloading from the actual message transmission, this could give rise to a race condition if transmission were to conclude just as the CPU reloaded the second buffer. No buffer would than be ready for transmission and the bus would be released. At least three transmit buffers are required to meet the first of the above requirements under all conditions.
As no low-cost CAN controller with this capability was available at the start of the S80 project, Volvo and Motorola agreed, in January 1995, to join in developing a device of this type (for on-chip integration with the next generation of 8-bit microprocessor, the HC08), to meet the requirements of timing analysis within the cost constraints of the project. The end result was the msCAN controller (which is now part of the automotive versions of both the 68HC08 and the 68HC12 microprocessors).

Volcano Lite

Employing a single-wire physical interface, the Volcano Lite low-speed multiplex communications protocol was designed to meet the need for low-performance, in-vehicle communications, such as sending a signal from an alarm sensor or light switch. The protocol complements the existing CAN-based Volcano system. Volcano Lite is characterised by low cost, predictable timing behaviour (bounded message latencies) and integration with Volcano. Signalling between the CAN network and Volcano Lite is totally transparent to the applications programmer.
Volcano Lite is a ’single-master’, ’multiple-slave’ network, the messages in which are referred to as frames. Each frame consists of a single-header byte, a block containing 1 to 8 bytes of data, and a checksum byte. The ninth bit in each character is used as an address bit, which is set to ’1’ in all header bytes and reset to ’0’ in all others, thereby minimising the load imposed on the CPU by unwanted interrupts to the low-cost slaves. The master node follows a schedule which indicates when each header is to be transmitted. Multiple schedules may be specified and used alternately. The master is the only node which transmits headers; however these are received by all nodes, which then determine individually whether to transmit the data block, receive the frame data or discard the frame (Fig. 4).

Volcano as seen by the programmer

The Volcano target software – which is resident in each ECM – provides the application programmer access to the system’s functionality by means of a simple API (Application Program Interface).

> “Configuring a network of ECMs may be thought of as ‘virtual soldering’”

Signals

The unit of communication used in most embedded control systems, particularly those in vehicles, consists of small data items, referred to in Volcano as signals. A signal may be visualised as a ’virtual wire’, which may either be

an input to or an output from an ECM. Configuring a network of ECMs may be thought of as 'virtual soldering', in which, for example, an output 'wire' from one ECM may be connected to the inputs of several other modules.

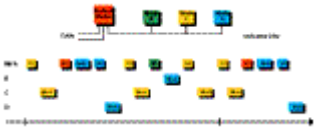

Fig. 4

Piggybacking

The efficiency with which signals are used in CAN is crucial. Since many small data items must be included in a single CAN frame, the overheads consist of at least 47 bits per frame (in the case of frames with 11-bit identifiers)' Thus at least 55 bits must be transmitted over the network to transfer an 8-bit signal in a single CAN frame - a hopelessly inefficient method. The solution is to include several small data items with similar timing constraints in the same frame, a technique which is commonly used in many existing CAN systems.

Abstraction

Volcano conceals the details of a configuration (signal packaging, network setup etc.), using an area of non-volatile memory (the 'configuration area') in the ECM to retain the data in an implementation-specific format. This was a primary – and crucial - design objective; if the application software is 'unaware' of the most configuration, the latter can be changed without affecting the application. One of the main disadvantages of most existing systems is that the configuration is exposed to the application software of the ECM. This means that the ECM program must be written specially for each configuration and that the module, as a result, is not normally interchangeable between different vehicle models without modification. Thus, the Volcano solution provides the systems integrator an enormous degree of flexibility.

Volcano calls – the API

> The Volcano solution affords the systems integrator an enormous degree of flexibility

Signals are the only forms of communication supplied to the application program; network frames are hidden. Volcano performs the tasks of packing the signals into frames and transmitting the frames, and of receiving frames and unpacking the signals. This means that the application programmer is unaware of the number of frames transmitted or received, and of the mapping between signals and frames. Volcano provides procedure calls (known as Volcano calls) which enable ECM application software to access these signals. Signals are either generated by the application software in the ECM and transmitted to the network (output signals), or generated elsewhere and read by the application software (input signals).

Signals have two general uses: to communicate both state information and state change information. A signal may contain state information which is updated regularly ('vehicle speed' as an example). A user of this type of signal needs to know how fresh the value is at the instant the signal is read and imposes requirements on the system configuration to ensure this. Other signals may communicate state change information which occurs sporadically (such as 'sensor failure'). In this case, the user needs to know the maximum time between the incident and the arrival of the event signal.

Update bits and flags

A Volcano signal may include an update bit indicating that the signal has been updated. This means that the ECM in question has generated a fresh value of the signal since its last transmission. The Volcano software in an ECM which receives a signal of this type automatically deletes the update bit once it has been detected. This ensures that the software is 'informed' of every signal update (the application can detect the update bit by means of flags as described below).
A flag is a Volcano object which is strictly local to an ECM and is bound to one of two items:

- The update bit of a received Volcano signal – which sets a flag on reception
- The frame containing a signal – in which the flag is set when the frame is received (regardless of whether an update bit has been set)

A number of flags may be bound to each update bit or to a frame. Volcano sets all the flags bound to an object when the occurrence in question is detected. The flags are cleared explicitly by the application software.

Signal types

“Signals may be represented in Boolean, integer or byte form”

Signals may be represented in Boolean, integer or byte form. The size of a Boolean signal is fixed at 1 bit of data, while an integer signal has a fixed size of between 0 and 32 bits, and a byte signal a fixed size of between 0 and 8 bytes. Integer signals are used to communicate values, and byte signals to convey bytes of unstructured data between ECMs (typically for diagnostic purposes). The advantage of Boolean and integer signals is that the signal values are independent of processor architecture; in other words, the values are consistent regardless of whether the microprocessor data in the individual ECMs is 'big-endian' or 'small-endian' in structure.
The Volcano programming interface provides a set of calls for handling communications. 'Read' and 'write' calls are provided for manipulating signals. A 'read' call returns the latest value of a signal to the caller, while a 'write' call sets the signal value. The volcano_output, volcano_input and volcano_gateway calls ensure that Volcano sends and receives frames on the networks. The volcano_output call copies signal values into frames and places them in the appropriate network devices. The volcano_input call receives incoming frames and makes the signal values available to 'read' calls. The volcano_gateway call copies the values of signals in frames received from a network to signals in frames transmitted to a network. The network device calls enable the application to connect to and disconnect from the networks, and to place the controllers in 'sleep' mode (to reduce idle current consumption).

The 'immediate' interface

Volcano also provides a very low latency communications mechanism in the form of the immediate signal interface. This is a 'view' of frames on the network which permits transmission to and reception from the Volcano domain without the normal Volcano input/output latencies, or without mutual exclusion requirements with the volcano_input and volcano_output calls. The immediate signal API contains two communications calls, imm_input and imm_output. The imm_output call copies values of immediate signals into a frame and places the frame in the appropriate CAN controller for transmission. The imm_input call makes the signal values in a received frame containing immediate signals available to 'read' calls.

Deliverables in the S80

Volcano is supplied to each ECM developer as a programming library containing the implementation of each of the Volcano calls. In addition, the application developer is provided with a special DOS-based Volcano configuration tool which uses a description of the signals used or generated by the ECM to present these signals to the programmer as programming objects. This description is agreed with Volvo (which controls the total information flow through the network). The configuration tool also reads data from an information file supplied directly by the ECM developer. This data (which is confidential to the developer and need not be divulged to Volvo) contains all of the information which Volcano requires to operate, and which does not affect the network or reconfiguration, the location of the network controller hardware in the memory, the flags used by the application, and so on. The configuration tool outputs a set of C language declaration and definition files, which are compiled together with the application program and linked with the Volcano library to produce the final executable program for entry in the ECM memory (see Fig. 7). The ECM is ready for inclusion in the network when the configuration area is added (see below for a more detailed description).

Resource requirements

The resources required by Volcano are modest. For a typical application based on an 8-bit microprocessor, the system uses 2.5 kB of ROM, while the CPU time demand is also low. These resources are little more than those required to write the same functionality directly into the application program.

> The ECM is ready for inclusion in the network when the configuration area is added

Development tools

Making the target software as efficient as possible and minimising the overhead in the ECMs were major objectives of the Volcano project. The development tools in which the off-line calculations were performed were important elements of the Volcano concept.

Signals database (SDB)

A database for storing information regarding networks, ECMs and signals (CAN and Volcano Lite) was developed during the S80 project. The original objective was to support the analysis and the design of CAN networking in the new model. This involved the concepts of deadlines, latencies and 'frame compilation'.
The signals database contains information on functions, nodes and signals. The term 'functions' refers to functions implemented by the electrical system, such as climate control, interior lighting and window winder operation, as perceived by the owner, while

the nodes are the system ECMs and the signals are the data items which carry the information over the bus. Each signal is transmitted by a single node and received by one or more nodes. Every instance of signal reception is part of the implementation of an electrical function (see Fig. 5).
One reason for storing this information in a central database was to afford the entire design team access to the same information regarding the distributed functions at any given stage of the design process. The database information is accessed through the PC network, and may be viewed from different aspects which are customised for ECM designers, function owners and other users.
The information in the database can be used to monitor the implementation of a specific function in the ECMs concerned, enabling an ECM designer to produce a list of all the functions implemented in the module.

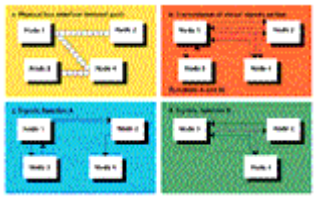

Fig. 5

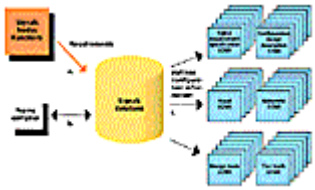

Fig. 6

Inputs and outputs

The database tool has a range of functions for editing, analysing and documenting the database content (Fig. 6). The most important of these are:

- support for mapping signals into CAN frames on different networks to produce a network configuration;
- generation of Volcano configuration information for each ECM (this information is produced in a standard format specified by Volcano as described below);
- generation of configuration data for a range of test tools, including VCT2000 and CANalyzer;
- generation of a set of mandatory documents, as part of the software specifications for ECMs.

Frame compiler

The frame compiler is the design and analysis tool on which the Volcano concept relies. Although the Volcano target software and the simple API which it provides are important elements of the concept, the real-time guarantees which are crucial to the S80 electrical system could not be implemented without the frame compiler.

> “The frame compiler output is used to generate the Volcano configuration information”

The frame compiler is a standalone PC program which processes data received from the signals database and returns the results to it. The frame compiler assigns priorities and periodicity to the CAN frames in accordance with the information received from the database. The frame configuration is reported back to the database, together with calculated values of worst case latencies for all the frames in the system.
Initially the frame compiler performs consistency checks on the information stored in the signals database. The timing analysis described in the previous sections is then applied. The worst case latency for every frame in the system is calculated and compared with the required deadline for each frame. In practice, the deadline is the shortest deadline among the signals mapped into the same frame.
The timing analysis is used in an iterative algorithm to compute the relative priorities

and transmission frequency of the frames. Each frame is finally assigned a unique, fixed priority – represented by the frame identifier – and the frame compiler returns the results to the signals database.
The frame compiler output is used to generate the Volcano configuration information (fixed and network information) automatically. These files comprise the input for the Volcano configuration tool.

Volcano configuration tool

The Volcano configuration tool (’vcfg’) is used both by the ECM developer and the Volvo systems integrator to build the necessary configuration information (see Fig. 7). Volvo uses the tool to generate the data for the configuration area of the ECM. This data is compiled from the information in the fixed and network configuration files. The output from Volvo’s use of the vcfg consists of binary configuration data (in Motorola S-record format), which must be entered in the configuration area of the specific ECM using Volvo’s proprietary software download methods and tools.
The vcfg tool - which can be viewed as the link between the ‘standard’ and implementation-specific parts of the Volcano configuration process - reads files in a defined and standardised format. and produces data which is specific to a particular implementation of Volcano on a particular processor.

Working methods

The final product of the signals database is a binary Volcano configuration data file, which is downloaded into the car in the assembly plant. Certain preliminary measures, starting with the definition of the vehicle’s electrical functions, are required before this can be carried out. The procedure is described below.

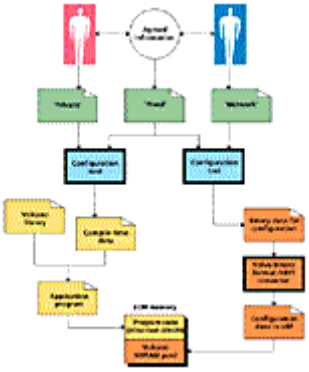

Fig. 7

From deadlines to CAN frames

Analysis of the functional requirements is the first step in configuring the CAN network in the S80. The results of analysis provide deadline requirements for each and every signal, and a solution which fulfils these deadlines is then developed.

Functional requirements

The electrical functions are defined at an early stage. Once the functional architecture of the electrical system has been determined, the functional requirements can be translated into requirements for the ECMs and other components.
The functional requirements include conditions relating to data transfer between ECMs – in other words, the network requirements.

ECM and network requirements

The data transfer requirements comprise the signal database input. Every function imposes individual requirements on at least one ECM and on the exchange of one or more Volcano signals. Depending on the type of signal, the function ’owner’ (i.e. the individual responsible for writing the function specification) specifies either the required signal ’freshness’ or the time between an incident and the resultant action. In either case, the deadline - defined as the maximum time between the initiation of a write call by the source ECM and the instant the signal is made available by Volcano to the receiving ECM - is an essential parameter.
When all functions have been defined, including the requirements relating to the

generation, transfer and reception of Volcano signals, the signals are mapped into CAN frames and the frame compiler is assigned to analyse if the system is schedulable; in other words, if all of the deadlines have been met.
If the system is schedulable, the frame compiler will export a set of network configuration files. If not, some requirements may have to be revised and the frames recompiled.

Volcano configuration files

The configuration information is divided into two separate parts (fixed and network information) contained respectively in the fixed and network configuration files (Fig. 7).

The fixed file, which contains information on the signals transmitted and received by the ECM, includes no network information or information relating to the frames on the network. The network file contains information regarding the mapping of transmitted and received signals into frames.
It is important to note that the supplier (or the ECM 'owner' at Volvo) is responsible for the correctness of the fixed file and for ensuring that it matches the software specification. The network file, however, is the responsibility of the systems integrator. (The ECM 'owner' is the individual responsible, among other things, for specifying the module, liaising with suppliers and implementing the appropriate parts of a particular function in 'his' or 'her' module.)

Binary information in VBF format

The fixed and network information is processed by vcfg to produce a binary Volcano configuration area, and is finally converted into VBF (the Volvo proprietary software download format) and delivered by the systems integrator to the Volvo software archive for subsequent downloading into the ECM flash memories.

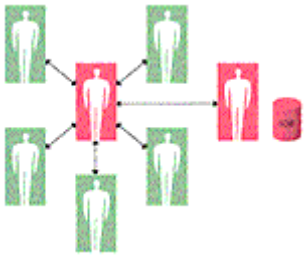

Fig. 8

Organisation surrounding signals database

Experience during the S80 project, and even before its commencement, indicated that an increasing level of integration and distribution of functions has a significant influence on the product development and production organisation. One purpose of the database tool and, indeed, of the entire Volcano concept, is to reduce this impact by introducing abstraction and composability. In a distributed environment, the ECMs can no longer be treated as just so many 'black boxes'. As a result, Volvo necessarily assumed greater responsibility for specifying the performance of the electrical system. It was decided to focus on electrical functions, with a function owner responsible for the system layout of the function, and for coordinating its implementation, testing etc. This includes the specification of signal deadlines to be entered in the signals database.

The function owner must work closely with the ECM owners responsible for implementing individual elements of a potentially high number of electrical functions in a single ECM. The ECM owner, in turn, is responsible for compiling the various demands on his ECM. The signals database is intended to be of major assistance in this respect.

The systems integrator occupies a central role. One of his or her primary responsibilities is to administer the signals database and to coordinate activities relating to it. Another task is to guarantee that all of the deadlines are met.

Production and aftersales aspects

One aim of Volcano was to achieve system flexibility. Starting with the Volvo S80, the large platform will remain in existence for several years and it is not possible to predict all of the new requirements which will arise in that time.
The feasibility of using existing software in different network configurations enables the network to be customised for every variant of the S80. The Volcano configuration area is treated like any other software component in the production and aftersales areas. With its software download capabilities, together with the new software management systems introduced in the plants and aftermarket, the Volcano concept will help Volvo to build a flexible platform to meet the demands of the future.

CONCLUSION

The advantages to Volvo of the development and application of Volcano include:

- **Production cost benefits due to high bus efficiency (four times as many signals can be transmitted at half the baud rate)**
- **Development cost benefits (in the form of a single, proven implementation which is much cheaper than multiple implementations by suppliers and conformance testing by Volvo)**
- **Improved network reliability, resulting in higher product quality**
- **Reduction in Volvo´s test load**
- **Reduction in supplier´s test load**
- **High degree of flexibility (useful in many situations)**
- **Recognition of the real-time problem (Volvo developed solutions before the problem had been recognised generally).**

Lennart Casparsson is an electrical engineering graduate of Chalmers University of Technology, Gothenburg. He joined Volvo Car Corporation in 1983 as a

Antal Rajnak studied electrical engineering at the technical universities of Budapest and St. Petersburg, and at Chalmers University of Technology, Gothenburg (CTH). He

Ken Tindell was awarded his PhD in real-time systems by the University of York in 1994 and has lectured at Uppsala University in Sweden since then. He was appointed

Peter Malmberg received his MSc in electrical engineering from Chalmers University of Technology, Gothenburg and joined Volvo Technological Development in

graduate trainee and moved to Electrical Design department in 1985 to work on a number of design projects. He has been responsible for the development of on-board data communications for VCC since 1993.

remained at CTH as a senior research engineer, working mainly on advanced data acquisition and measurement research projects, and was appointed director of its Monitoring Centre for Energy Research in 1988. He worked on robotics systems and CAD software for a Swedish clothing manufacturer from 1984 to 1992, when he formed his own company, Kimble Automotive Applications Labs. He was engaged as a consultant by VCC in 1994 to lead the development of the Volcano communications concept for the S80 and became MD of Volcano Communications Technologies AB in 1998.

managing director of Northern Real-Time Technologies in 1995 and directed the team which developed the Volcano run-time software for the ECMs in the Volvo S80. In 1998, Dr. Tindell was appointed technical director of Volcano Communications Technologies AB, with responsibility for ongoing development of the Volcano concepts.

1992, working mainly on the development of measuring systems for advanced emission analysis. He joined Sigma Systems in 1995 to work on assignments for Volvo Car Corporation, most recently on the S80 project. He was responsible for aspects of CAN communications design and participated in the development of the S80 communications concept.

ACKNOWLEDGEMENT

AcknowledgementsThe authors wish to express their thanks to Istvàn Horváth of Kimble, Tomas Ekström of VCC and Kjell Svensson of Sigma for their contributions to the Volcano project.