CAN for Critical Embedded Automotive Networks

THE CONTROLLER AREA NETWORK PROTOCOL OFFERS FLEXIBILITY IN SAFETY-CRITICAL AUTOMOTIVE SYSTEM DESIGN. THE BENEFITS OF USING ALREADY ESTABLISHED SPECIFICATIONS WILL AID AUTOMOTIVE SYSTEM DESIGNERS IN X-BY-WIRE DEVELOPMENT.

Lars-Berno Fredriksson
Kvaser

Over the past decade, electronic systems have gradually replaced mechanical ones in cars and trucks. The forces driving this replacement have mainly included environmental demands that require advanced electronic engine and driveline control in addition to reduction of wire-harness size. In the mid-1980s, Bosch and Intel developed a serial network protocol, called controller area network (CAN),[1] and implemented this in silicon chips for supporting hardware. The CAN protocol transfers information between electronic control units (ECUs) in vehicles. In the early 1990s, Mercedes was the first company to introduce CAN in standard cars. Since then, the CAN protocol has proved to be an inexpensive, robust solution for automotive control networks and is now well established and used in all types of vehicles.

So far, however, only non-safety-critical systems, with few exceptions, use CAN. Now, the automotive industry intends to use network technologies for safety-critical tasks such as steer-by-wire and brake-by-wire, often summarized as *X-by-wire*. It is a common opinion among controller network specialists that such systems should be time-triggered rather than event-triggered and that ECU access to the communication bus must be supervised and controlled by specific entities, called *bus guardians*.[2] Because CAN does not offer support for time-triggered message transfer or for bus guardians, the industry has looked at other protocols such as the time-triggered protocol (TTP)[3] and FlexRay.[4]

However, a better solution might be to use CAN as a base and then add missing facilities as needed. This technique, in contrast to traditional layered communication services, would lead to a more flexible architecture in which the communication would be integrated into the control system. Then the vast practical experience gained from using the CAN protocol for more than a decade in different environments and applications could be leveraged to provide better services.

General concept

The starting point for discussing the communication between ECUs in a distributed embedded control system (DECS) must be the system architecture. Predicting response time from an event detected in one module until the response action in another module is an important requirement in safety-critical systems. Thus, ECU communication is not a stand-alone entity but an integrated part of a real-time operating system at the system level. For the solution proposed in this article, I have made some assumptions about features needed to meet these requirements:

- Any communication in the system must be predefined.

0272-1732/02/$17.00 © 2002 IEEE

- The communication must be time scheduled during normal conditions.
- The time schedule must accept event-triggered emergency messages.
- The communication must have an event-triggered schedule as a backup.
- There must be a clear separation between the system level and the module level regarding failure detection and actions on failures.
- It must be possible to update the protocol at any time during the system's life cycle.

Predefined communication

In a DECS, unlike a LAN or telecommunication system, any need for communication must be known and defined a priori. Different ECUs can respond only to known messages and can transmit only those messages that they are programmed to transmit, and the system uses the message contents. LAN or telecom systems do not know any communication and any message contents in advance of system design, and the system does not use these contents. In these systems, in which communication functions as a service of the different network nodes, the communication service might not be available sometimes because the demand is stochastic. A DECS must be designed in accordance with the worst-case reaction times for different events, at least for safety-related operations.

Time-scheduled communication

The timing of the actions in collaborating ECUs depends on message delays. Although it is possible to calculate the maximum delay for any message in an event-triggered CAN system, it soon becomes obvious that an event-triggered approach can be difficult to design if correctness, failure detection, and maximum reaction time must be verified. Only a few messages can have a short, guaranteed maximum latency, and a lot of the bandwidth must be assigned to ensure system integrity. Systems with time-scheduled transmissions are much simpler to design, understand, analyze, and verify. It is important, especially for safety-critical systems, that system behavior during any conditions is understandable. System architectures based on time-scheduled communication have the best chance of meeting the dependability requirements of safety-critical systems.

Event-triggered emergency messages

The time schedule must accept event-triggered emergency messages, which are a big problem in time-scheduled systems. Emergency messages do not typically appear during normal conditions, but when they must be transmitted, their latency time must be short. If they are a part of the schedule, they occupy a substantial portion of the bandwidth that could be more effectively used. A much more efficient approach is to transmit regular messages on time schedules and emergency messages on events. CAN makes such an approach possible.

Event-triggered schedule as a backup

The weakness of a pure time-scheduled system is the clock. If one node is not time synchronized with the other nodes in the system, the system behavior is unpredictable. It is easy to detect a time-keeping failure, but such failures are not always easy to rectify. A more robust approach when trying to fix a time-keeping problem is to switch to graceful degradation by changing from the time-triggered schedule to an event-triggered backup schedule.

System and module separation

Although CAN itself and some current CAN-based protocols, including SAE J1939[5] and NMEA 2000,[6] were initially developed to be system independent and should not require any system designer, in practice, there are several reasons to have a system designer. Before a communication scheme with predictable latency can be designed, the number of nodes and their interactions has to be known, that is, a system designer is required for any real-time system. Furthermore, designers can enhance and simplify failure detection by using both a system level and a module level. Some failures can only be detected when you compare the responses of multiple modules to an event. In other cases, designers can make error detections redundant using mechanisms at both system and module levels. A system designer has complete knowledge of the system and can design a node capable of supervising the system. By separating the design task

into one system level and one module level, module designers can concentrate on the performance of their modules and leave system problems to the system designer.

Protocol update

Ideally, designers would solve all problems at a project's specification phase. When the team then approves the specification, it should be simple to get it implemented, verified, and validated. In the real world, however, designers usually detect some shortcomings of the specification long before the verification phase. Other shortcomings remain hidden until the product is far into its life cycle. Almost any change in a DECS will affect desired protocol properties, so the system must allow updating.

Ideal building block

In 1988, Intel brought the first CAN chip into the market. Since then, several manufacturers have designed and produced different types of CAN chips in large numbers, and many market segments, with varying demands, have used CAN chips. CAN has proved itself to be a solid base for different kinds of protocols in a great variety of DECSs. Integrating different CAN controllers in a system has most often been a simple task.

Because CAN is a low-level protocol, systems require a higher-layer protocol (HLP) built on top of CAN. It is the quality of the applied HLP that makes some CAN systems dependable and others not. The requirements of a dependable system relate to the system itself and to how the system resolves critical situations (see http://www.sp.se/electronics/RnD/palbus/eng/report.htm), which is why it is almost impossible to create one and only one HLP for all critical, embedded automotive networks.

A different approach is to make the communication an integrated part of the control system. For this approach, as mentioned previously, CAN is an excellent basic building block. Over the years, this technique has proved to be a robust way to convey information within control systems. The next step is to add new standardized support for global clocks, opening new opportunities for designing dependable systems. Developers could then combine well-proven CAN solutions with well-proven time-scheduled solutions, which would be an ideal starting point for generations of embedded automotive network designs.

CAN: Not an event-triggered protocol

Designers often rule out CAN outright because many falsely regard the protocol as event-triggered. CAN uses an efficient process to resolve message collisions when they occur on the bus. But the protocol does not require collisions. Nothing prevents CAN from being used for time-scheduled protocols. In fact, designers have successfully used it for several years in protocols such as CDA 101.[8] A great advantage of using CAN in time-scheduled protocols is that clock failures can easily be detected and efficiently handled.

Time-scheduled communication requires that each node rely on the same notion of time. A straightforward solution is to have local clocks in each node, synchronized to one of them by time messages; software achieves this in current CAN applications. Although an accuracy of 10 microseconds is possible, programming software in this way is often difficult because any change in a module's software might influence its clock performance. A standardized solution implemented directly in the CAN controller would be a better alternative.

Global clock support

CAN relies on time measurement for its bit encoding. The non-return-to-zero (NRZ) concept encodes the bits in CAN and thus the bus level does not change when transmitting adjacent bits of the same value. In these cases, the system calculates the number of bits by dead reckoning. Thus, a CAN controller already has the basic features for an integrated clock. According to the CAN standard, a bit can be divided into at least 25-bit time quanta, and each bit can be as low as 1 microsecond. The falling edge (the start of a frame) hard synchronizes a system's CAN controllers at the beginning of each message on the bus. Thus, a time resolution of 40 nanoseconds and an already used synchronizing flank are available, up-front. Only a counter has to be added to implement a local clock. However, to support a global clock, the local time must be synchronized to a system-wide notion of time, which requires a few more counters and registers. But the problem is well known and was

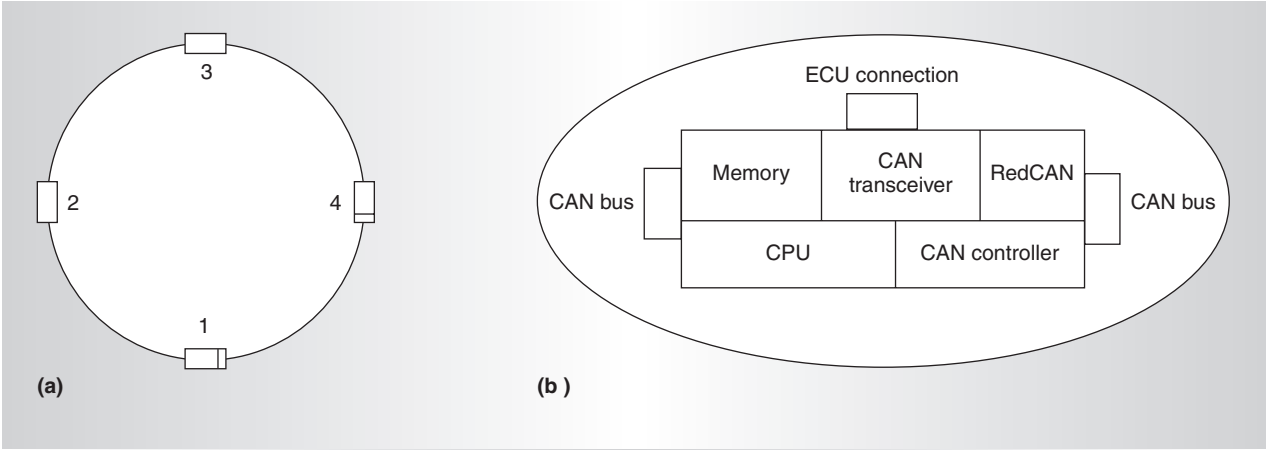

Figure 1. Wiring harness for a distributed embedded network with four nodes. The RedCAN units labeled 1 and 4 disconnect the bus. The section between 1 and 4 is disconnected. In case of a short or a break in another section, for example, between the units labeled 2 and 3, the 2 and 3 units terminate the bus and units 1 and 4 connect the section between 1 and 4 to the bus (a). Detail of a connection point. The RedCAN unit can either connect to the section or terminate the bus (b).

solved long ago in the telecom industry. Thus, a chip designer could add an accurate and adjustable clock to a CAN controller without much extra cost.[8]

Required bandwidth

A common claim from proponents of alternative standards to CAN is that an X-by-wire system requires a bit rate of at least 5 Mbps and that because CAN provides a bit rate of only 1 Mbps, its bandwidth is too limited. This might be true if you consider communication as a stand-alone entity. But the picture is quite different if you regard the communication and the global clock as integrated parts of the system control. By using the global clock for time stamping, designers can dramatically reduce the need for communication bandwidth by applying feedback loop control theories using known time jitter.[9]

CAN's collision resolution mechanism operates according to a bitwise arbitration of the first part of the message, called the *identifier field*. One and only one node can transmit the same bit pattern in the identifier field unless the rest of the message contains no data or exactly the same data. Thus, any message addressing must be implemented in a higher protocol layer. In scrutinizing the CAN standard, it seems that this offers features for efficient message transceiving of up to 94 bits of data on a bus. The so-called CAN identifier need not be an identifier at all, it can be the message itself.

A common communication problem occurs when several nodes simultaneously detect an alarm-causing event and each of them transmits alarm messages, resulting in a saturated bus. Designers can solve this problem with CAN by assigning the same identifier for an empty alarm message to all nodes capable of detecting the same failure. On the triggering event, they transmit the alarm message unless they just received the very same alarm. The result is only one message on the bus, regardless of how many modules detected the failure. CAN's address-less concept, in which every node receives every message, can resolve message collisions with no loss of bandwidth.

Building CANs for ECUs

The first step in building a CAN network for critical embedded automotive networks is wiring, which must be robust. Each ECU must connect at the right spot. Figure 1a shows a proposed physical architecture prepared with connection points for four modules. Each connection point has a unit that contains memory, a CAN transceiver, a RedCAN device, a CAN controller, and a small CPU, shown in Figure 1b.

RedCAN helps obtain redundancy efficiently.[10] A RedCAN device can detect physical defects as shortcuts or breakups in the wires and can terminate the bus. At startup, RedCAN singles out one section as a redundant sector. If one sector fails during runtime,

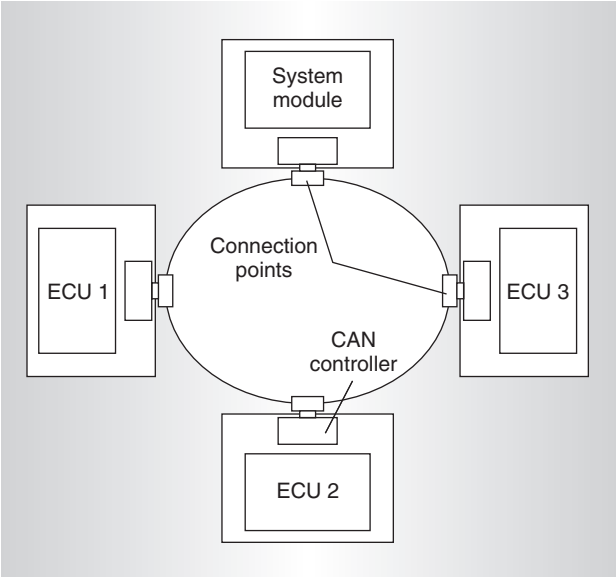

Figure 2. Architecture for dependable system with three electronic control units and one system module. The CAN controllers support global time. The system module controls the connection points.

the modules at each side automatically disconnect this sector, and the redundant sector becomes active. Adjacent modules can also disconnect a malfunctioning module. RedCAN not only provides efficient bus redundancy but also solves the problem of a completely crushed node. In many other systems, a crushed module might ruin the ordinary bus and the redundant ones because they all are connected to the module.

The memory in the connection-point device contains, at a minimum, information about what kind of ECU should be connected to this point but can include other essential information such as bit rate, setup parameters, serial number of the ECU last attached, and time schedules that otherwise must be communicated by a tool or across the bus during setup. By reading the memory before going active, the ECU can make sure that it is correctly attached in the system.

With a CPU at each connection point, the system designer can implement several tasks for supervising and controlling the system independently of the ECU designs. Because the CAN transceiver belongs to the wiring system, the designer can choose the bus levels (or even an optical alternative) while still using ECUs with standard CAN controllers. The designer can program the connection units with communication rules that act as bus guardians optimized for the system. Thus, the connection unit can physically disconnect an attached module if it does not behave according to the rules, either automatically or by command from another module.

Modules

Figure 2 shows the system with modules attached. The system module (SM), created by the designer, has a supervising role because it receives every message on the bus and has complete knowledge of expected system behavior on the bus. It can have the capability of detecting almost any serious ECU malfunction. When a module connects to the access point unit, a point-to-point CAN communication runs between the module and the unit. The access point unit can then interrogate the module and find out whether it is of the right kind and can correctly set the module before it is connected directly to the bus. The module can also check that it is correctly connected to an approved system before it turns into an active mode.

To make the modules as system independent as possible, they must be configurable by some kind of a tool. The SM can act as such a tool during setup. The CanKingdom (http://www.cankingdom.org/) standard provides a set of command messages to configure and control modules.[11] A module supporting CanKingdom lets the system designer assign CAN identifiers to messages, construct messages from internal signals, select responses on events or received messages from a list of alternatives, set the bit rate, set the periodicity of messages, and adjust the local clock to global time. With this flexibility, the system designer can create a CAN higher-layer protocol and integrate the control schemes with the communication.

The property of allowing integration of new modules into a system is often called *composability*. Composability requires a clear and stable interface specification in a module's value and time domains. Traditionally, this requirement can lead to inflexible systems because the system requirements must be known when the module is designed. But with CanKingdom, which provides well-defined interface building blocks rather than complete inter-

faces, designers can build flexible interfaces in any module, which the SM can modify to match current system needs—not only at startup but also during runtime.

Time schedules

Only the system designer knows how the different ECUs interact, how faults are detected, and how to correct those faults when failures happen. The combination of CAN and time-scheduled communication create new possibilities, especially in contrast to classical time-scheduling techniques. Any time-scheduled system must anticipate message loss due to electromagnetic disturbances. To cope with this problem, the system must retransmit the message one or more times in the same time slot, or the time slot must reappear often enough so the system can survive one or two lost slots.

CAN's collision resolution feature allows a more efficient use of bandwidth because critical messages can share the same time slot as noncritical messages. In these cases, the CAN system makes the time slot for a critical message long enough to allow message retransmission if the critical message gets corrupted. The system designer assigns a slot-sharing noncritical message to a CAN identifier with a lower priority, and its time slot starts slightly later than the critical message. If the critical message successfully transmits, the noncritical message automatically transmits immediately thereafter. If not, the system will retransmit the critical message and the noncritical message will lose the time slot.

Not all modules must support a global clock in a time-scheduled system. Some of them can piggyback on messages transmitted by modules that support the global clock by using CanKingdom's action-reaction feature. A module supporting the action-reaction concept has a list of tasks and events that other tasks or events can evoke. A command message establishes the action-reaction relationship. By setting up the reception of a message with a certain CAN identifier as the action event and the transmission of a certain message as the reaction task, designers can use any message appearing on the bus as a trigger for message transmission from other modules. The designer can then use time-scheduled messages from modules that support a global clock to trigger message transmission from other modules in the system.

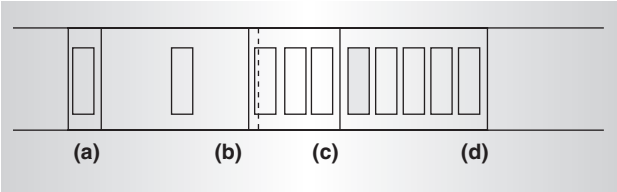

Figure 3. Messages can be scheduled in different ways to accommodate different scenarios. Tight time slot for a message from a module with a high accuracy rate: retransmission is not allowed (a). Time slot for a message from a module with a low-quality clock: the clock allows for a big jitter (b). Tight time slot for a critical message: two retransmissions are allowed. This slot is shared by a slot for noncritical messages that start slightly later than the critical message. CAN priority controls bus access (c). Time slot where the first message is from a module with a high-quality clock. The following messages are from modules with no clocks: These will transmit when the system detects the first message's CAN identifier (d).

The SM—or any module that knows the schedule of other modules—can immediately detect a module not synchronized to global time. If it detects that all other modules have lost synchronization, then the detecting module itself is unsynchronized. Because most transmitted CAN identifiers relate to specific modules, the SM knows the source of a message appearing on the bus at the wrong time. By carefully selecting the message priorities, designers can create a system to be fully functional for a long time even if the global time fails. The nondestructive message-priority collision-resolution mechanism provides a graceful degradation of the time schedule. When the clocks in the modules drift from global time, messages will gradually start to collide but will appear back-to-back on the bus. Even when the clock in a module gets out of phase, the system handles this problem safely and predictably. Figure 3 shows some different ways to schedule messages.

TTCAN standard

Ongoing work within ISO on a time-triggered CAN has reached committee draft status.[12] The draft focuses on standardizing how to schedule messages and how to get a global system clock. In the standard, messages operate in a matrix in which each column represents a time slot. The system transmits

messages row by row. Time messages, from a time master, occupy the first column in the matrix to synchronize the module clocks.

The advantages of the ISO proposal are that it is easy to comprehend and easy to create a schedule by hand. However, there are some disadvantages:

- The time resynchronization messages appear according to a schedule pattern rather than when needed, possibly wasting bandwidth.
- The column width governs the time slots, which are equal for all messages in the same column and cannot be optimized.
- Because the standard will be implemented in silicon, it might be difficult to have built-in support for a global clock if designers use a nonmatrix method for creating the schedule.
- The matrix only helps create a schedule, which means that, in practice, it will unfold into a timeline when executed, making the columns only a restriction and reducing the possibility of making more efficient schedules.
- Some details make it difficult to harmonize a CAN clock with other clocks—such as GPS, GSM, or Bluetooth clocks—possibly making it hard to build efficient gateways to other networks.

The common denominator in any time-triggered system is the notion of time, which means you can use a global clock not only for assisting communication but also for synchronizing tasks. Time synchronization would then operate independently of the scheduling method. The ISO proposal could be restructured to fulfill this requirement. Alternatively, a chip manufacturer could decouple the schedule and the clock parts in an implementation.

An ISO standard for establishing a global clock on top of CAN is already in preparation. Designers could use such a global clock for establishing time-controlled message transmission and synchronizing tasks in different nodes. The combination of CAN's collision resolution features with message scheduling opens up several possibilities for critical automotive network designs.

By applying an architecture with a clear division between the system and module levels, designers can simplify failure detection. A system node can supervise communication and check that it is running according to current rules and that the modules are responding to actions within reasonable limits. Using the proven CanKingdom tool, designers can flexibly implement system communication and control rules. The wiring could include active connection points to the modules so that the SM can ensure that the right modules connect at the right places at startup and during runtime. This technique could make time-triggered CANs an excellent choice for critical embedded automotive networks. MICRO

References

1. *ISO 11898, Road Vehicles: Interchange of Digital Information–Controller Area Network (CAN) for High-speed Communication*, Int'l Organization for Standardization, Geneva.
2. H. Kopetz, *Real-Time Systems Design Principles for Distributed Embedded Applications*, Kluwer Academic, Boston, 1997.
3. *Time-Triggered Protocol (TTP/C)*, Version 1.0, TTA Group, Vienna; http://www.ttagroup.org/ttp/specification.htm.
4. R. Belschner et al., *FlexRay Requirements Specification (draft), version 1.9.7*, FlexRay Consortium, 2001; http://www.flexray-group.com/.
5. *SAE J1939, Recommended Practice for Truck and Bus Control and Communications Network*, Soc. of Automotive Engineers, Warrendale, Pa., 2000; http://www.sae.org/products/j1939.htm.
6. *NMEA 2000 Standard for Serial-Data Networking of Marine Electronic Devices*, Nat'l Marine Electronics Assoc., Severna Park, Md., 2002; http://www.nmea.org/pub/2000/index.html.
7. D. Purdy, *CDA 101, Common Digital Architecture,* Naval Air Warfare Center Weapons Division (NAWCWD), Point Mugu, Calif.; http://www.cankingdom.org/download/Download%20files/CDA101/cda_101.htm.
8. L-B. Fredriksson and J. Österling, *An Outline for a CAN Global Clock*, Kvaser, Kinnahult, Sweden, 2000; http://www.cankingdom.org/download/Download%20files/Time%20Triggered%20Systems/tts.htm.

9. J.H. Nilsson, *Real-Time Control Systems with Delays,* doctoral dissertation, ISRN LUTFD2/TFRT–1049–SE, Dept. Automatic Control, Lund Inst. of Technology, Lund, Sweden, 1998.
10. H. Sivencrona et al., "A Novel Distributed Add-on Concept to Detect and Recover from Bus Failures in Controller Area Networks using REDCAN," *Proc. 8th Int'l CAN Conf.*, CiA-Can in Automation Int'l Users and Manufacturers Group, Erlangen, Germany, 2002.
11. *CANKingdom specification*, ver. 3.01, CanKingdom Int'l, Port Hueneme, Calif., 1996; http://www.cankingdom.org.
12. *Road Vehicles—Controller Area Network (CAN)—Part 4: Time Triggered Communication*, committee draft standard CD 11898-4 (N1147), ISO/TC 22/SC3/WG1, Int'l Organization for Standardization, Geneva, 2000.

Lars-Berno Fredriksson is president and cofounder of Kvaser. His research interests include distributed embedded control systems and advanced real-time systems, especially architectures. Fredriksson has an MSc in mechanics from the Chalmers University of Technology in Gothenburg, Sweden. He holds the appointed position of the Swedish expert for TTCAN in ISO/TC22/SC3.

Direct questions and comments about this article to Lars-Berno Fredriksson, Kvaser AB, Box 4076, SE 51104, Kinnahult, Sweden; lbf@kvaser.se.