CHAPTER

Automotive and Aerospace Systems

9

CHAPTER POINTS

- Networked control in cars and airplanes.
- Networks for vehicles.
- Security and safety of vehicles.

9.1 Introduction

Cars and airplanes are superb examples of complex embedded computing systems; we have real-life experience with them and understand what they do. They represent very large industries, and they are examples of safety-critical real-time distributed embedded systems.

We start with a discussion of use cases for vehicles. Section 9.2 discusses networked control in cars and airplanes. Section 9.3 describes in more detail several networks used in vehicles. Section 9.4 considers the safety and security issues of vehicles.

9.2 Vehicular use cases

Vehicles have been important markets for embedded computers since the early days of microprocessors. The advent of autonomous vehicles has enhanced the role of embedded computing in vehicles.

9.2.1 Vehicles as cyber-physical systems

Cars and airplanes are examples of cyber-physical systems; software provides real-time control for the physical plant.

Car subsystems

Fig. 9.1 shows a car network and three of the major subsystems in the car: the engine, the transmission, and the antilock braking system (ABS). Each is a mechanical system controlled by a processor. First, consider the roles of mechanical systems, all of which are mechanically coupled:

- The engine provides the power to drive the wheels.

Computers as Components. https://doi.org/10.1016/B978-0-323-85128-2.00009-8

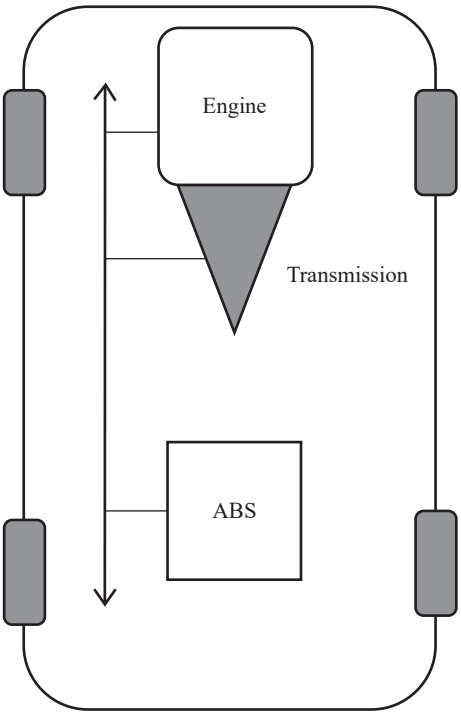


FIGURE 9.1

Major elements of an automobile network.

- The transmission mechanically transforms the engine's rotational energy into a form most useful by the wheels.
- The ABS system controls how the brakes are applied to each of the four wheels. It can separately control the brake on each wheel.

Now, consider the roles of the associated processors:

- The engine controller accepts commands from the driver via the gas pedal. It also takes several measurements. Based on the commands and measurements, it determines the spark and fuel timing of every engine cycle.
- The transmission controller determines when to change gears.
- The ABS system takes braking commands from the driver via the brake pedal. It also takes measurements from the wheels about their rotating speeds. It turns the brakes on and off each wheel to maintain traction on each wheel.

Car subsystem interactions

These subsystems must communicate with each other to do their jobs:

- The engine controller may change the spark timing during gear shifting to reduce shocks during shifting.

- The transmission controller must receive the throttle position from the engine controller to help determine the proper shifting pattern for the transmission.
- The ABS system tells the transmission when brakes are being applied in case the gear needs to be shifted.

None of these tasks needs to be performed at the highest rates in the system, which are the rates for spark timing. A relatively small amount of information can be exchanged to achieve the desired effect.

Avionics

Aircraft electronics are known as **avionics**. The most fundamental difference between avionics and automotive electronics is **certification**. Anything that is permanently attached to the aircraft must be certified. The certification process for production aircraft is twofold. First, the design is certified in a process known as **type certification**; then, the manufacture of each aircraft is certified during production. The certification process is a prime reason why avionics architectures are more conservative than automotive electronics systems.

9.2.2 Driver assistance and autonomy

Driving can be assisted or automated with many levels of sophistication. A complete trip generally requires several different types of driving: leaving a parking space, low-speed driving on local streets, cruising, and parking. The term **advanced driver-assistance system** (**ADAS**) encompasses a broad range of functions for various parts of the trip. Some cars provide parking assistance by using sensors to warn of proximity to obstacles; other cars fully automate some types of parking. Adaptive cruise control adjusts cruise speed when vehicles ahead drive more slowly. Emergency braking can be performed to avoid collisions. Automated driving refers to the more complete operation of the vehicle. However, driving can be automated to several levels of sophistication. SAE International has defined the following levels of driving automation [SAE18]:

- Level 0, no driving automation.
- Level 1, driver assistance. Sustained execution of either a lateral or longitudinal motion control subtask but not both. The driver is expected to perform the rest of the driving task.
- Level 2, partial driving automation. Sustained execution of both lateral and longitudinal vehicular motion control. The driver watches for objects and events and supervises the driving automation system.
- Level 3, conditional driving automation. Sustained performance of a specific dynamic driving task. The user will respond to requests from the system to intervene and respond to performance-relevant system failures.
- Level 4, high driving automation. Sustained performance of an operation design domain of a driving task and fallback. The user is not expected to respond to a request to intervene.
- Level 5, full driving automation. The sustained and unconditional performance of a dynamic driving task and fallback, not limited to a particular operational design

domain. An operational design domain is the environment or situation in which an automated system is designed to operate properly. The user is not expected to respond to a request to intervene.

9.3 Networked control systems in cars and airplanes

Cars and airplanes are examples of **networked control systems**: computer networks with processors and I/O devices that perform control functions. Control systems require real-time responsiveness over a closed loop, from measurement back to control action. Although a simple control system may be built with a microprocessor and a few I/O devices, complex machines require network-based control. The network has several principal uses. First, a network allows more computing power to be applied to the system than would be possible with a single CPU. Second, many control applications require the controller to be physically near the controlled device. Machines with fast reaction rates require controllers to respond quickly. If the controller is placed physically distant from the machine, the communication time to and from the controller may interfere with its ability to properly control the plant. A network allows several controllers to be placed near the components they control (e.g., engines and brakes) while allowing them to cooperate in the overall control of the car.

Modern automobiles may contain over 100 processors that execute 100 million lines of code [Owe15]. Modern airplanes incorporate less software, in large part due to the demands of certification. However, modern airplanes still rely on computers and networks for flight operations. Data on vehicle networks serve a wide range of purposes, ranging from critical vehicle controls to navigation and passenger entertainment. Autonomous driving imposes additional requirements on a vehicle's computing platform [Liu17]. Autonomous vehicles generally use multiple sensors of several types. Perceptual tasks include localization, object detection, and object tracking. Based on these results, the autonomous system must predict the actions of surrounding moving objects, plan paths to move toward the destination, and avoid obstacles as the environment changes.

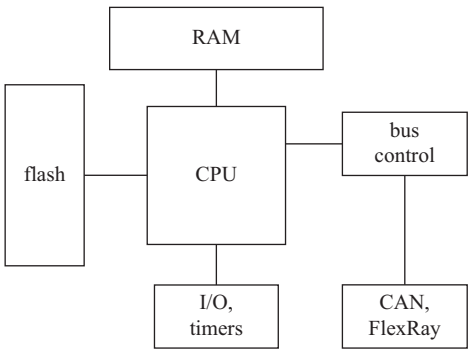
9.3.1 Network devices

Different terms are used in automotive and aerospace systems for devices connected to a network. An **electronic control unit** (**ECU**) is widely used in automotive design. The acronym *ECU* originally referred to an "engine control unit," but the meaning of the term was later expanded to any electronic unit in the vehicle. A **line replaceable unit** (**LRU**) is used in aircrafts for a unit that can be easily unplugged and replaced during maintenance.

ECUs

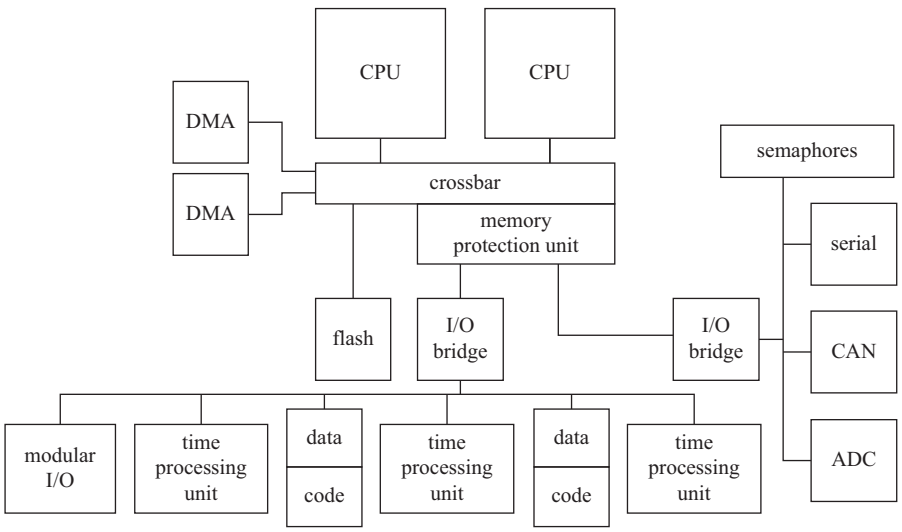
The next two examples describe ECUs designed for automotive systems: one is designed for body electronics, such as doors and lighting, and the other is designed for engine control.

Example 9.1: Infineon XC2200



The XC2200 family [Inf12] covers a range of automotive applications. One of the members of that family is designed for body control (e.g., lighting, door locks, and wiper control). The Body Control Module includes a 16/32 bit processor [Inf08], electronically erasable programmable ROM and static RAM, analog-to-digital converters, pulse-width modulators, serial channels, light drivers, and network connections.

Example 9.2: Freescale MPC5676R



The MPC5676R [Fre11B] is a dual-processor platform for powertrain systems.

The two main processors are members of the Power Architecture Book E architecture and are user-mode compatible with PowerPC. They provide short vector instructions for use in signal processing. Each processor has its own 16 K data and instruction caches. The time-processing unit can be used to generate and read waveforms. Interfaces to the CAN, the LIN, and FlexRay networks are supported.

9.3.2 Vehicle network architectures

Automotive networks

Fig. 9.2 shows two architectures for automotive networks. The traditional organization of networks in an automobile is known as **domain architecture** [Vem20]. ECUs are provided for various functions, including engine control, braking, entertainment, and so forth. Network connections are made to the ECU from each device for which it is responsible.

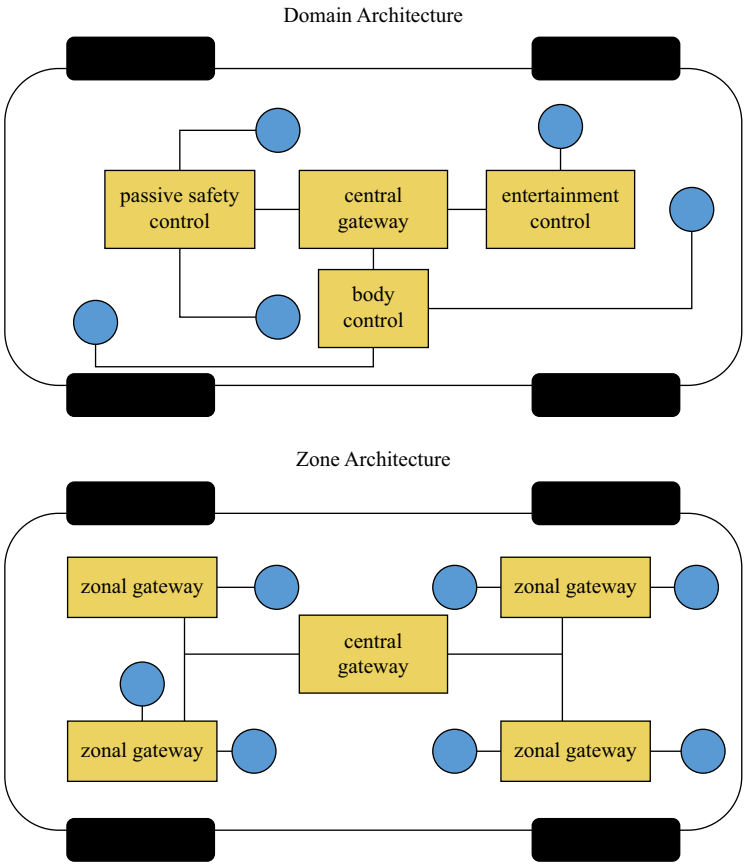


FIGURE 9.2

Domain and zone automotive network architectures.

The **zone architecture** has emerged as an alternative architecture. Several **zonal gateways** are positioned at various locations in the car. For example, a **central gateway** connects the zonal gateways. Each device connects to the zonal gateway to which it is closest. As a result, a given zonal gateway may perform several distinct types of functions.

Zone architectures have emerged as an approach to simplify automotive wiring. Wires in a car are typically organized into **harnesses**. The wiring harness of a car is typically its third heaviest component, after the body and engine [Kla19]. It is also the third-most expensive component in the car. Connecting devices to a nearby gateway can substantially reduce the size and cost of the wiring harness.

Aircraft networks

The traditional architecture [Hel04] for an avionics system has a separate LRU for each function: artificial horizon, engine control, flight surfaces, and so on.

A more sophisticated system is the bus-based one. The Boeing 777 avionics [Mor07], for example, are built from a series of racks. Each rack is a set of core processor modules (CPMs), I/O modules, and power supplies. CPMs may implement one or more functions. A bus known as SAFEbus connects the modules. Cabinets are connected using a serial bus known as ARINC 6210.

A distributed approach to avionics is the **federated network.** In this architecture, a function or several functions have their own network. The networks share data necessary for the interaction of these functions. A federated architecture is designed so that failure in one network will not interfere with the operation of the other networks.

The Genesis Platform [Wal07] is a next-generation architecture for avionics and safety-critical systems; it is used on the Boeing 787 Dreamliner. Unlike federated architectures, it does not require a one-to-one correspondence between application groups and network units. In contrast, Genesis defines a virtual system for avionic applications that are then mapped onto a physical network that may have a different topology.

9.4 Vehicular networks

Vehicular networks often have relatively low bandwidth when compared to fixed networks, such as local area networks. However, the computations are organized so that each processor has to send only a relatively small amount of data to other processors to do the system's work. We will first look at the CAN bus, which is widely used in cars and sees some use in airplanes. We will then briefly consider other vehicular networks.

9.4.1 CAN bus

The **Controller Area Network** or **CAN bus** [Bos07] was designed for automotive electronics, and was first used in production cars in 1991. A CAN network consists of a set of electronic control units connected by the CAN bus; the ECUs pass messages to each other using the CAN protocol. The CAN bus is used for safety-critical operations, such as antilock braking. It is also used in less-critical applications,

such as passenger-related devices. CAN is well-suited to the strict requirements of automotive electronics: reliability, low power consumption, low weight, and low cost.

CAN comes in several variations. The version known as high-speed CAN uses bit-serial communication and runs at rates of up to 1 Mb/s over a twisted pair connection of 40 m. An optical link can also be used. The bus protocol supports multiple masters on the bus.

Physical layer

As shown in Fig. 9.3, each node in the CAN bus has its own electrical drivers and receivers that connect the node to the bus in a wired-AND fashion. The driving circuits on the bus cause the bus to be pulled down to 0 if any node on the bus pulls the bus down; this bus voltage state is known to be **dominant** [Wat17]. If no node pulls down the bus, the bus voltage remains high; this state is known as **recessive**. When all nodes are transmitting ones, the bus is said to be in the recessive state. When a node transmits a zero, the bus is in the dominant state. Data are sent on the network in packets known as **data frames**.

CAN is a synchronous bus; all transmitters must send at the same time for bus arbitration to work. Nodes synchronize themselves with the bus by listening to the bit transitions on the bus. The first bit of a data frame provides the first synchronization opportunity in a frame. The nodes must also continue to synchronize against later transitions in each frame.

Data frame

The format of a CAN data frame is shown in Fig. 9.4. A data frame starts with a dominant bit and ends with a string of seven recessive bits. There are at least three bit fields between the data frames. The first field in the packet contains the packet's

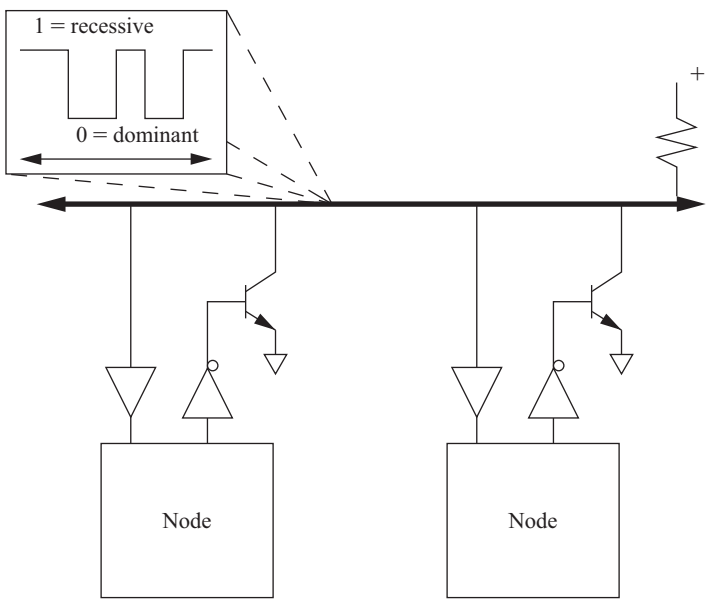


FIGURE 9.3

Physical and electrical organization of a CAN bus.

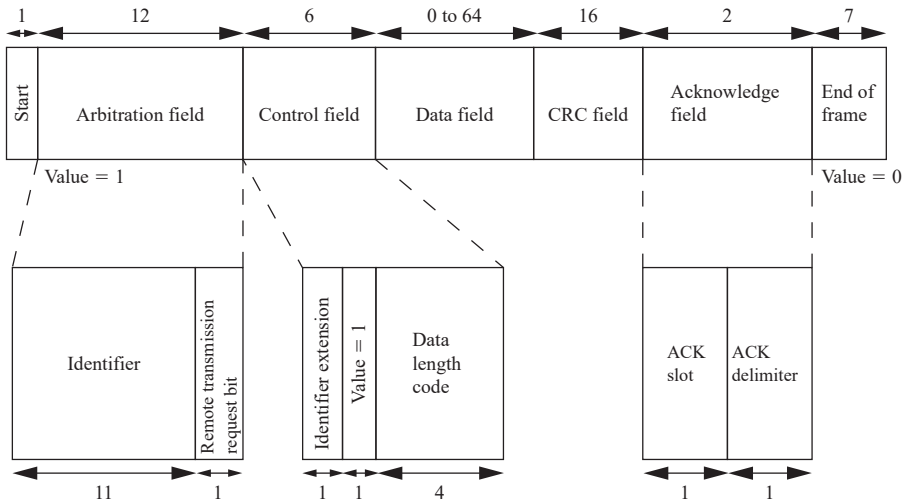


FIGURE 9.4

CAN data frame format.

destination address, which is known as the arbitration field. The destination identifier is 11 bits long. The trailing remote transmission request (RTR) bit is set to 0 if the data frame is used to request data from the device specified by the identifier. When RTR = 1, the packet is used to write the data to the destination identifier. The control field provides an identifier extension and a 4-bit length for the data field with a 1 in between. The data field is from 0 to 8 bytes, depending on the value given in the control field. A cyclic redundancy check (CRC) is sent after the data field for error detection. The acknowledge field is used to let the identifier signal whether the frame was correctly received: the sender places a recessive bit (1) in the acknowledgment (ACK) slot of the acknowledge field; if the receiver detects an error, it forces the value to a dominant (0) value. If the sender sees a 0 on the bus in the ACK slot, it knows it must retransmit. The ACK slot is followed by a single bit delimiter, followed by the end-of-frame field.

Arbitration

Control of the CAN bus is arbitrated using a technique known as Carrier Sense Multiple Access with Arbitration on Message Priority (CSMA/AMP). CAN encourages a data-push programming style. Network nodes transmit synchronously, so they all start sending their identifier fields simultaneously. When a node hears a dominant bit in the identifier and tries to send a recessive bit, it stops transmitting. By the end of the arbitration field, only one transmitter will be left. The identifier field acts as a priority identifier, with the all-0 identifier having the highest priority.

Remote frames

A remote frame is used to request data from another node. The requestor sets the RTR bit to a recessive bit to specify a remote frame; it also specifies zero data bits. The node specified in the identifier field will respond with a data frame that has the requested value. Note that there is no way to send parameters in a remote

frame; for example, you cannot use an identifier to specify a device and provide a parameter to say which data value you want from that device. Instead, each possible data request must have its own identifier.

Error handling

An error frame can be generated by any node that detects an error on the bus. Upon detecting an error, a node interrupts the current transmission with an error frame, which consists of an error flag field followed by an error delimiter field of eight recessive bits. The error delimiter field allows the bus to return to the quiescent state so that data frame transmission can resume. The bus also supports an overload frame, which is a special error frame sent during the interframe quiescent period. An overload frame signals that a node is overloaded and will not be able to handle the next message. The node can delay the transmission of the next frame with up to two overload frames in a row, hopefully giving it enough time to recover from its overload. The CRC field can be used to check a message's data field for correctness.

If a transmitting node does not receive an acknowledgment of a data frame, it should retransmit the data frame until the data are acknowledged. This action corresponds to the data link layer in the Open Systems Interconnection model.

Fig. 9.5 shows the basic architecture of a typical CAN controller. The controller implements the physical and data link layers; because the CAN is a bus, it does not need network layer services to establish end-to-end connections. The protocol control block handles determining when to send messages, when a message must be resent because of arbitration losses, and when a message should be received.

9.4.2 Other automotive networks

Time-triggered architectures

The **time-triggered architecture** [Kop03] is an architecture for networked control systems that provides more reliable communication delays. Events in the time-triggered architecture are organized around real time. Since devices in the network need time to respond to communication events, time is modeled as a sparse system.

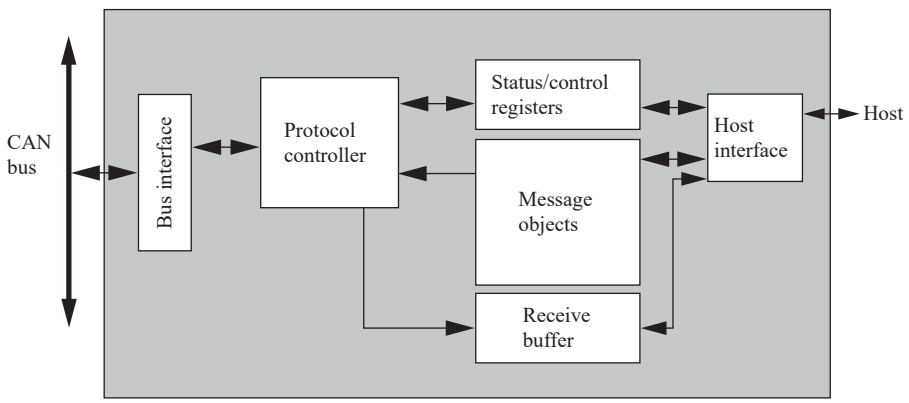


FIGURE 9.5

Architecture of a CAN controller.

Intervals of active communication are interspersed with idle periods. This model ensures that even if the clock's value varies somewhat from device to device, all devices on the network will be able to maintain the order of events in the system.

FlexRay

The **FlexRay network** [Nat09] has been designed as the next generation of system buses for cars. FlexRay provides high data rates—up to 10 Mbits/s—with deterministic communication. It is also designed to be fault tolerant. Communications on the bus are designed around a communication cycle. Part of the cycle, known as the *static segment*, is dedicated to events for which communication time has been guaranteed. The communication time for each of these events is determined by a schedule set up by the designer. Some devices may need only sporadic communication, and they can make use of the *dynamic segment* for these events.

LIN

The Local Interconnect Network (LIN) bus [Bos07] was created to connect components in a small area, such as a single door. The physical medium is a single wire that provides data rates of up to 20 kbits/s for up to 16 bus subscribers. All transactions are initiated by the master and responded to by a frame. The software for the network is often generated from a LIN description file that describes the network subscribers, the signals to be generated, and the frames.

Several buses have come into use for passenger entertainment. Bluetooth is becoming the standard mechanism for cars to interact with consumer electronics devices, such as audio players or phones.

MOST

The Media-Oriented Systems Transport (MOST) bus [Bos07] was designed for entertainment and multimedia information. The basic MOST bus runs at 24.8 Mbits/s and is known as MOST 25, and 50 and 150 Mbits/s versions have been developed. MOST can support up to 64 devices. The network is organized as a ring.

Data transmission is divided into channels. A control channel transfers control and system management data. Synchronous channels are used to transmit multimedia data; MOST 25 provides up to 15 audio channels. An asynchronous channel provides high data rates, but without the quality-of-service guarantees of the synchronous channels.

100 BASE-T1

Automotive Ethernet refers to any variation of the Ethernet used in automobiles. The 100BASE-T1 standard is one example of a commonly used version of Ethernet for automobiles. Signals are carried over an unshielded twisted pair of wires. Unlike many other forms of Ethernet, this standard allows full-duplex connections; devices at both ends of a connection can transmit and receive simultaneously over the twisted pair.

The next example looks at a controller designed to interconnect LIN and CAN buses.

Example 9.3: Automotive Central Body Controllers

A central body controller [Tex11C] runs various devices that are part of the car body, including lights, locks, windows, and so forth. It includes a CPU that performs management, communications, and power management functions. The processor interfaces with the CAN bus and LIN bus transceivers. Devices such as remote car locks, lights, or wiper blades are connected to LIN buses. The processor transfers commands and data between the main CAN bus and the device-centric LIN buses, as appropriate.

9.5 Safety and security

Cars and airplanes pose important challenges for the design of safe and secure embedded systems. The large number of vehicles makes them potentially dangerous; their internal complexity makes them vulnerable to a wide variety of threats.

Threat models

Vehicles are vulnerable to threats from many sources:

- **Maintenance.** Maintenance technicians must access the vehicle's internals, including computers. They may maliciously modify components. Even if the technicians are not malicious, if the computers they use for their work have been compromised, they may act as conduits or gateways for attacks.
- **Component suppliers.** Components may be shipped with backdoors or other problems. The components may come from a corrupt supplier or may be the victim of unauthorized modification by an errant employee.
- **Passengers.** Modern vehicles supply network connections for passengers. These networks can easily serve as avenues for attackers to enter the vehicle's core systems.
- **Passersby.** Wireless passenger networks may extend their range beyond the car, allowing others to attack. Wireless door lock mechanisms provide another avenue for attack. Some cars provide telematics services for remote access to the vehicle's operation, providing another avenue of attack.

Example attacks

These threat models are not hypothetical—they are realistic. The next example describes two experiments in car hacking. Afterwards, we present an example of a possible airplane hacking incident.

Example 9.4: Experiments in Car Hacking

Researchers have demonstrated techniques for hacking automobiles [Kos10, Che11]. To demonstrate the seriousness of the vulnerabilities found, they concentrated only on techniques that provided them with control over all of the car's systems. The team identified a variety of methods to access the car's internals: using traditional hacking techniques to infect the diagnostic computers used by mechanics; using a specially coded CD to modify the code of the CD player and then using the CD player to infect other car devices; sending signals to the car's telematics system to control it.

Computer security researchers demonstrated vulnerabilities by taking over a Jeep Cherokee driven by a journalist [Gre15]. The car's telematics system was used to obtain entry into the vehicle's computer systems. The entertainment system was then compromised, and its software modified; the computers did not check the validity of the software updates. The entertainment system was then used to send messages on the car's CAN bus to control other parts of the car, such as killing the engine or disabling the brakes.

Example 9.5: Airplane Hacking

A computer security researcher was arrested on suspicion of having hacked into Boeing 737 during a flight [Pag15]. An affidavit states that the person hacked into the in-flight entertainment system, and then, modified code in the Thrust Management Computer.

Safety

Not all problems are caused by malicious activity. Software bugs can result in serious safety problems, including accidents. The next example describes an airplane crash in which software problems are implicated.

Example 9.6: Software Implicated in Airplane Crash

Software bugs are suspected in the crash of an Airbus A400M [Pag15B, Chi15]. Software in the ECUs is suspected to have caused three engines on an A400M to shut down during flight, causing a fatal crash.

The following example describes the issues raised in lawsuits on automotive software.

Example 9.7: Design Errors Implicated in Car Crashes

A court in Oklahoma ruled that Toyota was liable in a case of unintended acceleration [Dun13]. An expert in the case testified that the electronic throttle control system source code was of unreasonably low quality, that software metrics predicted additional bugs, and that the car's fail-safe capabilities were both inadequate and defective. Koopman [Koo14] provided a detailed summary of the topics from the case. His summary states that the car's electronic throttle control system code contained 67 functions with a cyclomatic complexity over 50, and that the throttle angle function had a cyclomatic complexity of 146, with a cyclomatic complexity value over 50 considered "untestable."

In another case, a car manufacturer implemented a **defeat** on its own cars; it installed software that deactivated air pollution controls on its cars.

Example 9.8: Volkswagen Diesel Defeat

In 2015, Volkswagen admitted to installing a **defeat** of its own software on its diesel cars [Tho15]. The software defeat was detected when the vehicle was being tested for emissions; in this case, the software enabled all emissions control features. When the car was not being tested, a variety of emissions controls was disabled. With disabled emissions controls, cars could emit up to 40 times more emissions.

Autonomy introduces new safety concerns. Example 9.9 describes a fatal collision involving a pedestrian and a vehicle controlled by a developmental automated driving system.

Example 9.9: Collision Between Vehicle Controlled by a Developmental Automated Driving System and Pedestrian

The National Transportation Safety Board (NTSB) [Nat19] reported an accident that occurred on the evening of March 18, 2018, in Tempe, Arizona. A proprietary developmental automated driving system was operational in a test vehicle and active at the time of the crash. The vehicle struck and fatally injured a pedestrian crossing N. Mill Avenue outside a crosswalk. The NTSB's provided this probable cause statement:

> *The National Transportation Safety Board determines that the probable cause of the crash in Tempe, Arizona, was the failure of the vehicle operator to monitor the driving environment and the operation of the automated driving system because she was visually distracted throughout the trip by her personal cell phone. Contributing to the crash were the Uber Advanced Technologies Group's (1) inadequate safety risk assessment procedures, (2) ineffective oversight of vehicle operators, and (3) lack of adequate mechanisms for addressing operators' automation complacency—all a consequence of its inadequate safety culture. Further factors contributing to the crash were (1) the impaired pedestrian's crossing of N. Mill Avenue outside a crosswalk, and (2) the Arizona Department of Transportation's insufficient oversight of automated vehicle testing.*

The vehicle's developmental automated driving system was designed to operate in autonomous mode only on designated and premapped routes. The automated driving system's sensors included a single light detection and ranging system, 8 dual-ranging radars, and 11 cameras.

The report describes that the automated driving system first detected the pedestrian 5.6 s before the crash, first as a vehicle, then as an unknown object and a bicyclist. The automated driving system continued to track the pedestrian until the crash. However, it did not correctly predict the pedestrian's path or reduce the vehicle's speed in response. At 1.2 s before impact, the automated driving system determined that a collision was imminent, and that the situation exceeded the response specifications of the automated driving system's braking system to avoid collision. The design of the vehicle relied on the operator to take control of the vehicle.

The NTSB report provides several recommendations:

- To the National Highway Traffic Safety Administration:

  Require entities who are testing or who intend to test a developmental automated driving system on public roads to submit a safety self-assessment report to your agency. (H-19-47)

  Establish a process for the ongoing evaluation of the safety self-assessment reports as required in Safety Recommendation H-19-47 and determine whether the plans include appropriate safeguards for testing a developmental automated driving system on public roads, including adequate monitoring of vehicle operator engagement, if applicable. (H-110-48)

- To the State of Arizona:

  Require developers to submit an application for testing automated driving system (ADS)-equipped vehicles that, at a minimum, details a plan to manage the risk associated with crashes and operator inattentiveness and establishes countermeasures to prevent crashes or mitigate crash severity within the ADS testing parameters. (H-19-49)

Establish a task group of experts to evaluate applications for testing vehicles equipped with automated driving systems, as described in Safety Recommendation H-19-49, before granting a testing permit. (H-19-50)

- To the American Association of Motor Vehicle Administrators:

  Inform the states about the circumstances of the Tempe, Arizona, crash and encourage them to (1) require developers to submit an application for testing automated driving system (ADS)-equipped vehicles that, at a minimum, details a plan to manage the risk associated with crashes and operator inattentiveness and establishes countermeasures to prevent crashes or mitigate crash severity within the ADS testing parameters, and (2) establish a task group of experts to evaluate the application before granting a testing permit. (H-19-51)

- To the Uber Technologies, Inc., Advanced Technologies Group:

  Complete the implementation of a safety management system for automated driving system testing that, at a minimum, includes safety policy, safety risk management, safety assurance, and safety promotion. (H-19-52)

9.6 Summary

Automobiles and airplanes rely on embedded software, and they illustrate several important concepts in advanced embedded computing systems. They are organized as networked control systems with multiple processors communicating to coordinate real-time operations. They are safety-critical systems that demand the highest levels of design assurance.

What we learned

- Cars and airplanes make use of networked control systems.
- The computing platforms for vehicles make use of heterogeneous sets of processors that communicate over heterogeneous networks.
- The complexity of vehicles creates challenges for secure and safe vehicle design.
- Engine controllers execute mathematical control functions at high rates to operate the engine.

Further reading

Kopetz [Kop97] provided a thorough introduction to the design of distributed embedded systems. The book by Robert Bosch GmbH [Bos07] discusses automotive electronics in detail. The Digital Aviation Handbook [Spi07] describes the avionics systems of several aircraft.

Questions

Q9-1 Give examples of the component networks in a federated network for an automobile.

Q9-2 Draw a UML sequence diagram for a use case of a passenger sitting in a car seat and buckling the seatbelt. The sequence diagram should include the passenger, the seat's passenger sensor, the seat belt fastening sensor, the seat belt controller, and the seat belt fastened indicator, which is on when the passenger is seated, but the seat belt is not fastened.

Q9-3 Draw a UML sequence diagram for a use case for an attack on a car through its telematics unit. The attack first modifies the software on the telematics unit and then modifies software on the brake unit. The sequence diagram should include the telematics unit, the brake unit, and the attacker.

Lab exercises

L9-1 Build an experimental setup that lets you monitor messages on an embedded network.

L9-2 Build a CAN bus monitoring system.