1

Introduction to Real-Time Embedded Systems

Real-time embedded systems have become pervasive. They are in your cars, cell phones, Personal Digital Assistants (PDAs), watches, televisions, and home electrical appliances. There are also larger and more complex real-time embedded systems, such as air-traffic control systems, industrial process control systems, networked multimedia systems, and real-time database applications. It is reported that in the Lexus LS-460 released in September 2006, there are more than 100 microprocessors embedded when all optional features are installed. It is also estimated that 98% of all microprocessors are manufactured as components of embedded systems. In fact, our daily life has become more and more dependent on real-time embedded applications. This chapter explains the concepts of embedded systems and real-time systems, introduces the fundamental characteristics of real-time embedded systems, and defines hard and soft real-time systems. The automotive antilock braking system (ABS) is used as an example to show a real-world embedded system.

1.1 Real-Time Embedded Systems

An *embedded system* is a microcomputer system embedded in a larger system and designed for one or two dedicated services. It is embedded as part of a complete device that often has hardware and mechanical parts. Examples include the controllers built inside our home electrical appliances. Most embedded systems have real-time computing constraints. Therefore, they are also called real-time embedded systems. Compared with general-purpose computing systems that have multiple functionalities, embedded systems are often dedicated to specific tasks. For example, the embedded airbag control system is only responsible for detecting collision and inflating the airbag when necessary, and the embedded controller in an air conditioner is only responsible for monitoring and regulating the temperature of a room.

Real-Time Embedded Systems, First Edition. Jiacun Wang.

Another noteworthy difference between a general-purpose computing system and an embedded system is that a general-purpose system has full-scale operating system support, while embedded systems may or may not have operating system support at all. Many small-sized embedded systems are designed to perform simple tasks and thus do not need operating system support.

Embedded systems are *reactive systems* in nature. They are basically designed to regulate a physical variable in response to the input signal provided by the end users or sensors, which are connected to the input ports. For example, the goal of a grain-roasting embedded system is regulating the temperature of a furnace by adjusting the amount of fuel being injected into the furnace. The regulation or control is performed based on the difference between the desired temperature and the real temperature detected by temperature sensors.

Embedded systems can be classified based on their complexity and performance into small-scale, medium-scale, and large-scale. Small-scale systems perform simple functions and are usually built around low-end 8- or 16-bit microprocessors or microcontrollers. For developing embedded software for small-scale embedded systems, the main programming tools are an editor, assembler, cross-assembler, and integrated development environment (IDE). Examples of small-scale embedded systems are mouse and TV remote control. They typically operate on battery. Normally, no operating system is found in such systems.

Medium-scale systems have both hardware and software complexities. They use 16- or 32-bit microprocessors or microcontrollers. For developing embedded software for medium-scale embedded systems, the main programming tools are C, C++, JAVA, Visual C++, debugger, source-code engineering tool, simulator, and IDE. They typically have operating system support. Examples of medium-scale embedded systems are vending machines and washing machines.

Large-scale or sophisticated embedded systems have enormous hardware and software complexities, which are built around 32- or 64-bit microprocessors or microcontrollers, along with a range of other high-speed integrated circuits. They are used for cutting-edge applications that need hardware and software codesign techniques. Examples of large-scale embedded systems are flight-landing gear systems, car braking systems, and military applications.

Embedded systems can be *non-real-time* or *real-time*. For a non-real-time system, we say that it is correctly designed and developed if it delivers the desired functions upon receiving external stimuli or internal triggers, with a satisfied degree of QoS (Quality of Service). Examples are TV remote controls and calculators.

Real-time systems, however, are required to compute and deliver correct results within a specified period of time. In other words, a job of a real-time system has a *deadline*, being it hard or soft. If a hard deadline is missed, then the result is useless, even if it is correct. Consider the airbag control system in

automobiles. Airbags are generally designed to inflate in the cases of frontal impacts in automobiles. Because vehicles change speed so quickly in a crash, airbags must inflate rapidly to reduce the risk of the occupant hitting the vehicle's interior. Normally, from the onset of the crash, the entire deployment and inflation process is about 0.04 seconds, while the limit is 0.1 seconds.

Non-real-time embedded systems may have time constraints as well. Imagining if it takes more than 5 seconds for your TV remote control to send a control signal to your TV and then the embedded device inside the TV takes another 5 seconds to change the channel for you, you will certainly complain. It is reasonable that consumers expect a TV to respond to remote control event within 1 second. However, this kind of constraints is only a measure of system performance.

Traditional application domains of real-time embedded systems include automotive, avionics, industrial process control, digital signal processing, multimedia, and real-time databases. However, with the continuing rapid advance in information and communication technology and the emergence of Internet of things and pervasive computing, real-time embedded applications will be found in any objects that can be made smart.

1.2 Example: Automobile Antilock Braking System

A distinguished application area of real-time embedded systems is automobiles. Automotive embedded systems are designed and developed to control engine, automatic transmission, steering, brake, suspension, exhaustion, and so on. They are also used in body electronics, such as instrument panel, key, door, window, lighting, air bag, and seat bag. This section introduces the ABS.

An ABS is an automobile safety system. It is designed to prevent the wheels of a vehicle from locking when brake pedal pressure is applied, which may occur all of a sudden in case of emergency or short stopping distance. A sudden lock of wheels will cause moving vehicles to lose tractive contact with the road surface and skid uncontrollably. For this reason, ABS also stands for *antiskid braking system*. The main benefit from ABS operation is retaining the directional control of the vehicle during heavy braking in rare circumstances.

1.2.1 Slip Rate and Brake Force

When the brake pedal is depressed during driving, the wheel velocity (the tangential speed of the tire surface) decreases, and so does the vehicle velocity. The decrease in the vehicle velocity, however, is not always synchronized with the wheel velocity. When the maximum friction between a tire and the road surface is reached, further increase in brake pressure will not increase the *braking force*, which is the product of the weight of the vehicle and the friction

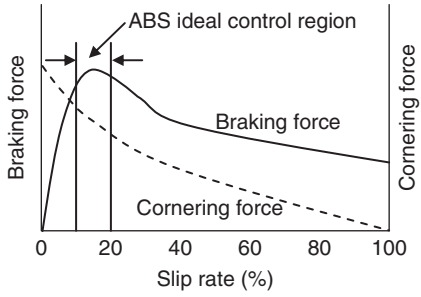

Figure 1.1 Relationship among the slip rate, braking force, and cornering force.

coefficient. As a consequence, the vehicle velocity is greater than the wheel speed, and the wheel starts to skid. The wheel slip rate, s, is defined as

$$s = \frac{V - \omega R}{V}$$

where V, ω, and R denote the vehicle speed, wheel angular velocity, and wheel rolling radius, respectively. Under normal driving conditions, $V = \omega R$ and thus, $s = 0$. During severe braking, it is common to have $\omega = 0$ and $V > 0$, which translates to $s = 1$, a case of wheel lockup.

Figure 1.1 describes the relationship among the slip rate, braking force, and cornering force. The continuous line in the figure represents the relationship between the slip rate and braking force. It shows that the braking force is the largest when the slip rate is between 10% and 20%. The braking distance is the shortest at this rate. With further increase in slip rate, the braking force is decreased, which results in a longer braking distance. The dashed line depicts the relationship between the slip rate and *cornering force*. The cornering force is generated by a vehicle tire during cornering. It works on the front wheels as steering force and on the rear wheels to keep the vehicle stable. It decreases as the slip rate increases. In case of a lockup, the cornering force becomes 0 and steering is disabled.

1.2.2 ABS Components

The ABS is composed of four components. They are speed sensors, valves, pumps, and an electrical control unit (ECU). Valves and pumps are often housed in hydraulic control units (HCUs).

1.2.2.1 Sensors

There are multiple types of sensors used in the ABS. A *wheel speed sensor* is a sender device used for reading a vehicle's wheel rotation rate. It is an electromagnet illustrated in Figure 1.2. As the sensor rotor rotates, it induces AC voltage in the coil of the electromagnet. When the rotation of the sensor rotor increases, the magnitude and frequency of induced voltage increase as well.

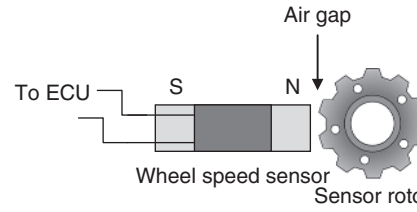

Figure 1.2 Wheel speed sensor.

A *deceleration sensor* measures the vehicle's rate of deceleration. It is a switch type of sensor. It uses phototransistors that can be activated by light. In a deceleration sensor, two Light-Emitting Diode (LEDs) aim at two phototransistors that are separated by a slit plate. When the vehicle's rate of deceleration changes, the slit plate swings in the vehicle's rear-to-front direction. The slits in the slit plate act to expose the light from the LEDs to the phototransistors. This movement of the slit plate switches the phototransistors ON and OFF. The combinations formed by the two phototransistors switching ON and OFF distinguish the rate of deceleration into four levels, which are sent as signals to the ABS ECU.

A *steering angle sensor* (SAS) measures the steering wheel position angle and rate of turn. The SAS is located in a sensor cluster in the steering column. The cluster always has more than one steering position sensor for redundancy and to confirm data. The ECU module must receive two signals to confirm the steering wheel position. These signals are often out of phase with each other. The SAS tells the ABS control module where the driver is steering the vehicle while the body motion sensors tell it how the body is responding.

A *yaw-rate sensor* is a gyroscopic device that measures a vehicle's angular velocity around its vertical axis. The angle between the vehicle's heading and vehicle actual movement direction is called *slip angle*, which is related to the yaw rate.

A *brake pressure sensor* captures the dynamic pressure distribution between a brake pad and the rotor surfaces during actual braking.

1.2.2.2 Valves and Pumps

Auto brakes typically work with hydraulic fluid. The brake's master cylinder supplies fluid pressure when the pedal is applied. In a standard ABS system, the HCU houses electrically operated hydraulic control *solenoid valves* that control the brake pressure to specific wheel brake circuits. A solenoid valve is a plunger valve that is opened and closed electrically. When power is applied to the solenoid, a magnetic coil is energized, which moves the plunger.

There are multiple *hydraulic circuits* within the HCU, and each hydraulic circuit controls a pair of solenoid valves: an *isolation valve* and a *dump valve*. The valves have three operation modes: *apply*, *hold*, and *release*. In the apply mode, both valves are open and allow the brake fluid to freely flow through the

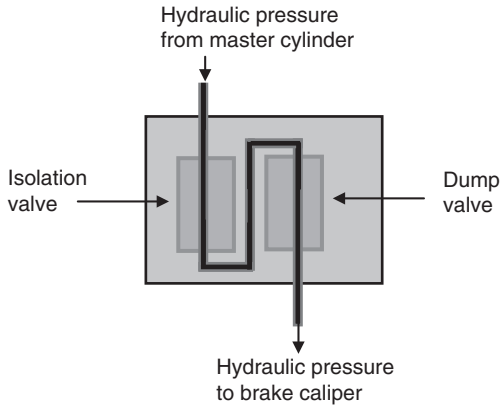

Figure 1.3 Valves operate in the apply mode.

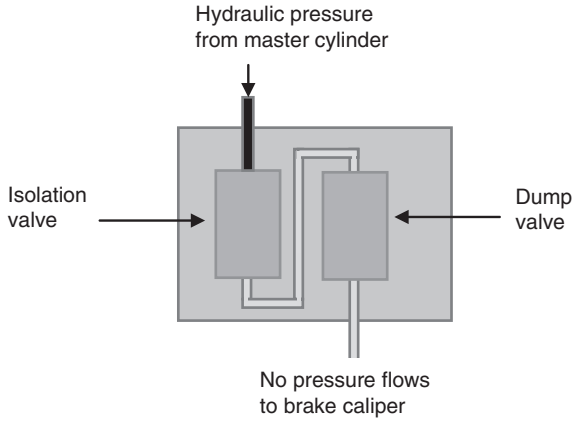

Figure 1.4 Valves operate in the hold mode.

HCU control circuit to the specific brake circuit, as illustrated in Figure 1.3. In this mode, the driver is in full control of the brakes through the master cylinder.

In the hold mode, both valves are in the closed position that isolates the master cylinder from the brake circuit. This prevents the brake pressure from increasing any further should the driver push the brake pedal harder. The brake pressure to the wheel is held at that level until the solenoid valves are commanded to change its position. This is illustrated in Figure 1.4.

In the release mode, the isolation solenoid valve is closed, but the dump valve is open to release some of the pressure from the brake, allowing the wheel to start rolling again. The dump valve opens a passage back to the accumulator where the brake fluid is stored until it can be returned by an electric *pump* to the master-cylinder reservoir. This is illustrated in Figure 1.5.

The pump is the heart of the ABS. Antilock brakes wouldn't exist without the hydraulic ABS pump. At the detection of wheel slip under heavy breaking,

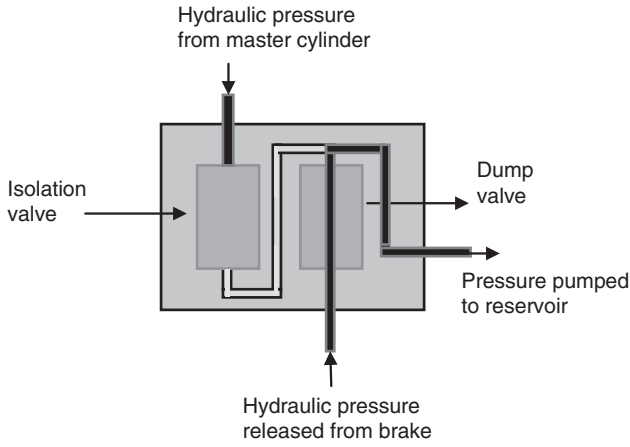

Figure 1.5 Valves operate in the release mode.

the pump in the HCU sends the brake fluid back to the master cylinder, pushing one or both pistons rearward in the bore. The controller modulates the pump's status in order to provide the desired amount of pressure and reduce slipping.

All valves are open during normal braking. When a wheel locks up, the braking pressure supplied to it should be reduced until it returns to spin. The ABS hydraulic unit works by closing the solenoid valve leading to the wheel that is locking up, thereby reducing the brake force the wheel receives. This way, the wheel's deceleration rate slows down to a safe level. Once that level is achieved, the solenoid opens again to perform its normal function.

1.2.2.3 Electrical Control Unit

The ECU is the brain of the ABS. It is a computer in the car. It watches all the sensors connected to it and controls the valves and pumps, as shown in Figure 1.6. Simply put, if the ABS sensors placed at each wheel detect a lockup, ABS intervenes within milliseconds by modulating the braking pressure at each individual wheel. In this way, the ABS prevents the wheels from locking up during braking, thus ensuring steerability and stability combined with the shortest possible braking distance.

The ECU periodically polls the sensor readings all the time and determines whether any unusual deceleration in the wheels occurs. Normally, it will take a car 5 seconds to stop from 60 mph under ideal conditions, but when there is a wheel lockup, the car could stop spinning in less than 1 second. Therefore, a rapid deceleration in the wheels is a strong indication that a lockup is occurring. When the ECU detects a rapid deceleration, it sends a control signal to the HCU to reduce the pressure to the brakes. When it senses the wheels accelerate, then it increases the pressure until it senses the deceleration

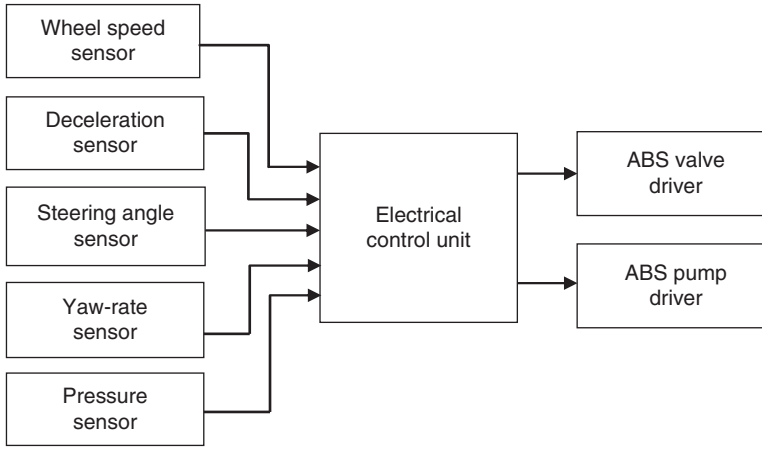

Figure 1.6 Electrical control unit.

again. The deceleration–acceleration repetition occurs very quickly, with the rapid opening and closing of the valves, until the tires slow down at the same rate as the car. Some ABS systems can cycle up to 16 times per second.

1.2.3 ABS Control

ABS brake controllers pose unique challenges to the designers. The main difficulties in the design of any ABS controller arise from the strong nonlinearity and uncertainty of the system to be controlled. First of all, the interaction between the tire and the road surface is very complex and hardly understood. Existing friction models are mostly experimental-based approximations of highly nonlinear phenomena. The dynamics of the whole vehicle is also nonlinear, and it even varies over time. In addition, ABS actuators are discrete, and control precision must be achieved with only three types of control commands: build pressure, hold pressure, or reduce pressure (recall the three operation modes of solenoid valves).

Many different control methods have been developed for ABS. Research on improved control methods is still continuing. The method applied in early systems is *threshold control*, which is as simple as bang–bang control. It uses wheel acceleration and wheel slip as controlled variables. Once the calculated wheel deceleration or wheel slip is over one of the threshold values, the brake pressure is commanded to increase, hold constant, or decrease. Since the brake pressure is cyclically changed based solely on the binary states of the input variables, wheel speed oscillations over time are less controllable.

A class of more advanced control methods uses a cascade closed-loop control structure shown in Figure 1.7. The outer loop, which includes the

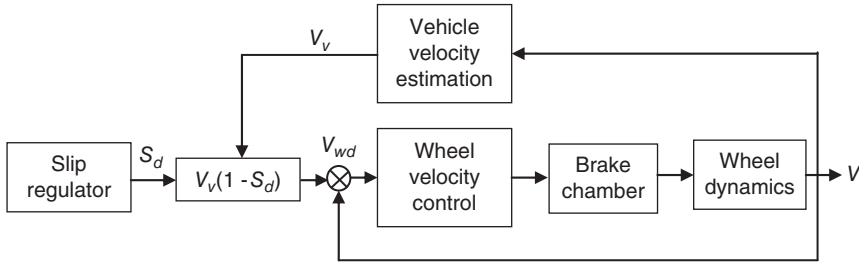

Figure 1.7 A Cascade control structure for ABS.

vehicle velocity estimation (V_v) and desired slip calculation, provides the command signal (V_{wd}) for the inner wheel velocity loop. For inner-loop control, several mechanisms have been proposed. The PID controller, for example, is one of such mechanisms. PID stands for proportional–integral–derivative. PID controllers adopt feedback control law and are widely used in industrial control systems. A PID controller continuously calculates an error value as the difference between a desired set point and a measured process variable. It is an integration of three control policies. With the proportional control (P), the controller output is proportional to the error, which in our case is V_{wd}. With the integral control (I), the controller output is proportional to the amount of time the error is present, which eventually leads to a zero error. With the derivative control (D), the controller output is proportional to the rate of change of the error, which reduces the response time. It has been shown that even for complex and changing surface types, good results can be attained with the conventional PID control algorithms. In the recent years, the well-known features of the conventional PID have been combined with the robustness, self-tuning, or adaptability to nonlinear models of other control methods, which highly enhanced the ABS performance.

No matter what control law is adopted, the feedback control loop can be implemented as an infinite periodic loop:

```
set timer to interrupt periodically with period T;
DO FOREVER
 wait for interrupt;
 read sensor data;
 compute control value u;
 output u;
ENDDO;
```

Here, the period T is a constant in most applications. It is an important engineering parameter. If T is too large, then the control variables will not get adjusted quickly; if it is small, then it will result in excessive computation.

1.3 Real-Time Embedded System Characteristics

The ABS example should have given us some idea about what a real-time embedded system looks like and how it interacts with the larger system it resides in to fulfill the specified function. This section discusses the characteristics of general real-time embedded systems.

1.3.1 System Structure

A real-time embedded system interacts with its environment continuously and timely. To retrieve data from its environment – the target that it controls or monitors, the system must have sensors in place. For example, the ABS has several types of sensors, including wheel speed sensors, deceleration sensors, and brake pressure sensors. In the real world, on the other hand, most of the data is characterized by analog signals. In order to manipulate the data using a microprocessor, the analog data needs to be converted to digital signals, so that the microprocessor will be able to read, understand, and manipulate the data. Therefore, an analog-to-digit converter (ADC) is needed in between a sensor and a microprocessor.

The brain of an embedded system is a controller, which is an embedded computer composed of one or more microprocessors, memory, some peripherals, and a real-time software application. The software is usually composed of a set of real-time tasks that run concurrently, may or may not be with the support of a real-time operating system, depending on the complexity of the embedded system.

The controller acts upon the target system through *actuators*. An actuator can be hydraulic, electric, thermal, magnetic, or mechanic. In the case of ABS, the actuator is the HCU that contains valves and pumps. The output that the microprocessor delivers is a digit signal, while the actuator is a physical device and can only act on analog input. Therefore, a digit-to-analog conversion (DAC) needs to be performed in order to apply the microprocessor output to the actuator. Figure 1.8 shows the relations among all these system components.

1.3.2 Real-Time Response

A real-time system or application has to finish certain tasks within specified time boundaries. This is the character that distinguishes a real-time system from a non-real-time system. The ABS is a typical real-time system. When the sensors detect a dramatic deceleration of wheels, the system must act quickly to prevent the wheels from being locked up; otherwise, a disaster may occur. Moreover, the control law computing is also real-time: a cycle of a sensor data processing and control value computing must be finished before the next cycle starts; otherwise, the data to be processed will pile up. If a missile guidance system fails to make timely corrections to its attitude, it can hit the wrong

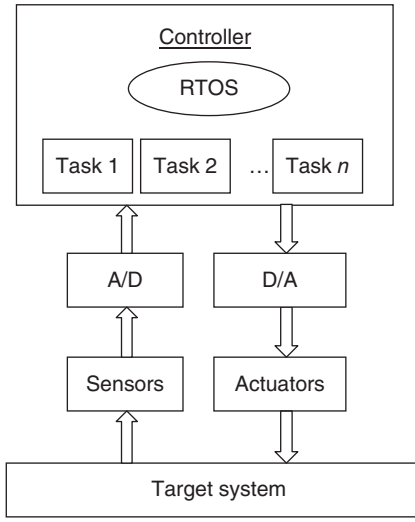

Figure 1.8 Structure of real-time embedded systems.

target. If a GPS satellite doesn't keep a highly precise measure of time, position calculations based on its signal will simply be wrong.

Deadlines of real-time tasks are typically derived from the required responsiveness of the sensors, actuators, and the dynamics of the target that the embedded system controls. Real-time systems are expected to execute all tasks by their deadlines. However, "real-time" does not mean "real fast" or "the faster, the better." Take a cardiac pacemaker as an example. Obviously, if it fails to induce current through the heart muscle at the right time, the patient's heart can go into fibrillation. However, if it induces current faster than normal heart rhythm, it will cause problem as well.

1.3.3 Highly Constrained Environments

Real-time embedded systems are often run in highly resource-constrained environments, which make the system design and performance optimization quite challenging. Although some embedded systems, such as air-traffic control systems and wireless mobile communication systems, run with very powerful processors, a lot of them are equipped with 8-bit processors only. Examples are the systems embedded in dishwashers, microwaves, coffee makers, and digital watches. Most embedded systems are constrained in terms of processor speed, memory capacity, and user interface. Many embedded systems operate in an uncontrolled harsh environment. They have to survive excessive heat, moisture, vibration, shock, or even corrosion. The ABS and automotive embedded systems that control ignition, combustion, and suspension are such examples. Therefore, embedded systems must be optimized in terms of size, weight, reliability, performance, cost, and power consumption to fit into the

computing environment and perform their tasks. Thus, embedded systems typically require far more optimization than standard desktop applications.

1.3.4 Concurrency

Concurrency refers to a property of systems in which several computations are executing simultaneously and potentially interacting with each other. Embedded systems by design are closely interacting with their physical environment. We have demonstrated this feature through the ABS analysis. Physical environment is by its nature concurrent – multiple processes occur at the same time. For example, the following events in the ABS can occur at the same time:

- Wheel speed sensor event
- Deceleration sensor event
- Brake pedal event
- Solenoid valve movement
- Pump operation

Almost all of these events have strict constraints on the response time. All deadlines should be met.

Because of the existence of multiple control processes and each process may have its own control rate, many real-time embedded systems are *multirate* systems. For example, a real-time surveillance system needs to process both audio and video inputs, but they are processed at different rates.

1.3.5 Predictability

A real-time system must behave in a way that can be predicted in terms of all timing requirements. For instance, it must be mathematically predictable if a specific task can be completed before a given deadline. Factors that go into this calculation are system workload, the power of processors, run-time operating system support, process and thread priorities, scheduling algorithm, communication infrastructure, and so on. A real-time system such as an ABS, or an airplane's flight-control system, must always be 100% predictable, or human lives are at stake.

Many real-time embedded systems contain heterogeneous computing resources, memories, bus systems, and operating systems and involve distributed computing via global communication systems. Latency and jitter in events are inevitable in these systems. Therefore, related constraints should be specified and enforced. Otherwise, these systems can become unpredictable.

A term related to predictability is determinism. Determinism represents the ability to ensure the execution of an application without concern that outside factors, such as unforeseen events, will upset the execution in unpredictable ways. In other words, the application will behave as intended in terms

of functionality, performance, and response time, all of the time without question.

1.3.6 Safety and Reliability

Some real-time embedded systems are safety-critical and must have high reliability. Examples are cardiac pacemakers and flight control systems. The term *safety* means "freedom from accidents or losses" and is usually concerned with safety in the absence of faults as well as in the presence of single-point faults. Reliability, on the other hand, refers to the ability of a system or component to perform its required functions under stated conditions for a specified time. It is defined as a stochastic measure of the percentage of the time the system delivers services. Embedded systems often reside in machines that are expected to run continuously for years without errors. Some systems, such as space systems and undersea cables, are even inaccessible for repair. Therefore, embedded system hardware and software are usually developed and tested more carefully than those for general-purpose computing systems.

Reliability is often measured in failures per million operating hours. For example, the requirement for a typical automotive microcontroller is 0.12 failures per million operating hours. The measurement is 37.3 for an automotive oil pump. Failures could be caused by mechanical "wear-out," software defects, or accumulated run-time faults.

1.4 Hard and Soft Real-Time Embedded Systems

There are *hard* real-time systems and *soft* real-time systems. A hard real-time system is a system in which most timing constraints are hard. A soft real-time system is a system in which most timing constraints are soft.

A hard real-time constraint is a constraint that a system *must* meet. If the deadline is missed, it will either cause the system failure or result in a zero usefulness of the delivered service. On the other hand, a soft constraint is a constraint that a system should meet, but when the deadline is occasionally missed, it won't cause any disastrous result, and the delivered service is still useful to a certain extent.

Normally, a hard constraint is expressed deterministically. For example, we may have the following constraints for the ABS:

- The wheel speed sensors must be polled every 15 milliseconds.
- Each cycle, the control law computation for wheel speed must be finished in 20 milliseconds.
- Each cycle, the wheel speed prediction must be completed in 10 milliseconds.

These constraints are hard because the sensor data, control value, and wheel speed predicted value are all critical to the correct functioning of the ABS. It is

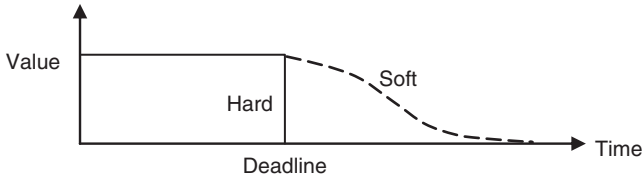

Figure 1.9 Value of hard and soft real-time tasks when deadline is missed.

also because these events are periodical. If the deadline of one cycle is missed, the next cycle starts immediately, and thus, the late result becomes useless.

Soft constraints are often expressed statistically. For example, we may have the following constraints for an automated teller machine (ATM):

- After a credit card or debit card is inserted, the probability that the ATM prompts the user to enter a passcode within 1 second should be no less than 95%.
- After it receives a positive response from the bank that issued the card, the ATM should dispense the specified amount of cash within 3 seconds at a chance of no less than 90%.

The deadlines with these two constraints are soft, because a few misses of the deadlines do no cause serious harm; only the degree of customers' satisfaction of using the system is negatively affected.

Figure 1.9 illustrates the value function of a real-time task in terms of response time. If the task has a hard deadline, then its value drops to zero when the deadline is missed. If the task has a soft deadline, then its value decreases when the deadline is missed, but not to zero right way.

Many hard real-time systems also have soft constraints and vice versa. When a timing constraint is specified as hard, then a rigorous validation is required.

Exercises

1 Give an example of real-time database application. Is it a hard or a soft real-time system? Give your arguments.

2 The car engine management system (EMS) is a real-time embedded system. Read related online materials, and find out major hardware components of the system and how they interact with each other to ensure the best engine performance.

3 Give an example of real-time embedded systems in which an earlier response than expected is as bad as a late response.

4 Give an example of a real-time embedded system that has both hard and soft real-time constraints.

Suggestions for Reading

Shin and Ramanathan [1] introduces the basic concepts and identifies the key issues in the design of real-time systems. Axer *et al*. [2] summarizes the current state of the art in research concerning how to build timing-predictable embedded systems. A survey of ABS control laws is presented in Ref. [3].

References

1 Shin, K. and Ramanathan, P. (1994) Real-time computing: a new discipline of computer science and engineering. *Proceedings of the IEEE*, **82** (1), 6–25.
2 Axer, P., Ernst, R., Falk, H. *et al.* (2012) *Building Timing Predictable Embedded Systems*, *ACM Transactions on Embedded Computing Systems*.
3 Aly, A., Zeidan, E., Hamed, A., and Salem, F. (2011) An antilock-braking systems (ABS) control: a technical review. *Intelligent Control and Automation*, **2**, 186–195.