CHAPTER 1

INTRODUCTION

The correctness of many systems and devices in our modern society depends not only on the effects or results they produce but also on the time at which these results are produced. These real-time systems range from the anti-lock braking controller in automobiles to the vital-sign monitor in hospital intensive-care units. For example, when the driver of a car applies the brake, the anti-lock braking controller analyzes the environment in which the controller is embedded (car speed, road surface, direction of travel) and activates the brake with the appropriate frequency within fractions of a second. Both the result (brake activation) and the time at which the result is produced are important in ensuring the safety of the car, its driver, and passengers.

Recently, computer hardware and software are increasingly embedded in a majority of these real-time systems to monitor and control their operations. These computer systems are called embedded systems, real-time computer systems, or simply real-time systems. Unlike conventional, non-real-time computer systems, real-time computer systems are closely coupled with the environment being monitored and controlled. Examples of real-time systems include computerized versions of the braking controller and the vital-sign monitor, the new generation of airplane and spacecraft avionics, the planned Space Station control software, high-performance network and telephone switching systems, multimedia tools, virtual reality systems, robotic controllers, battery-powered instruments, wireless communication devices (such as cellular phones and PDAs), astronomical telescopes with adaptive-optics systems, and many safety-critical industrial applications. These embedded systems must satisfy stringent timing and reliability constraints in addition to functional correctness requirements.

Figure 1.1 shows a model of a real-time system. A real-time system has a decision component that interacts with the external environment (in which the decision

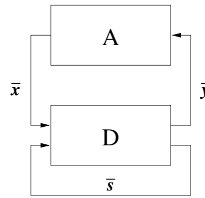

Figure 1.1 A real-time system.

component is embedded) by taking sensor readings and computing control decisions based on sensor readings and stored state information. We can characterize this real-time system model with seven components:

1. A sensor vector $\bar{x} \in X$,
2. A decision vector $\bar{y} \in Y$,
3. A system state vector $\bar{s} \in S$,
4. A set of environmental constraints A,
5. A decision map D, $D : S \times X \to S \times Y$,
6. A set of timing constraints T, and
7. A set of integrity constraints I.

In this model, X is the space of sensor input values, Y is the space of decision values, and S is the space of system state values. Let $\bar{x}(t)$ denote the value of the sensor input $\bar{x}$ at time t, and so on.

The environmental constraints A are relations over X, Y, S and are assertions about the effect of a control decision on the external world which in turn affect future sensor input values. Environmental constraints are usually imposed by the physical environment in which the real-time decision system functions.

The decision map D relates $\bar{y}(t+1)$, $\bar{s}(t+1)$ to $\bar{x}(t)$, $\bar{s}(t)$, that is, given the current system state and sensor input, D determines the next decisions and system state values. Decision maps can be implemented by computer hardware and software components. A decision system need not be centralized, and may consist of a network of coordinating, distributed monitoring/decision-making components.

The decisions specified by D must conform to a set of integrity (safety) constraints I. Integrity constraints are relations over X, S, Y and are assertions that the decision map D must satisfy to ensure safe operation of the physical system under control. The implementation of the decision map D is subject to a set of timing constraints T, which are assertions about how fast the map D has to be performed. In addition, timing constraints exist on the environment (external to the decision system) that must be satisfied for the correct functioning of this environment.

There are two ways to ensure system safety and reliability. One way is to employ engineering (both software and hardware) techniques, such as structured programming principles, to minimize implementation errors and then utilize testing tech-

niques to uncover errors in the implementation. The other way is to use formal analysis and verification techniques to ensure that the implemented system satisfy the required safety constraints under all conditions given a set of assumptions. In a real-time system, we need to not only satisfy stringent timing requirements but also guard against an imperfect execution environment, which may violate pre-runtime design assumptions. The first approach can only increase the confidence level we have on the correctness of the system because testing cannot guarantee that the system is error-free [Dahl, Dijkstra, and Hoare, 1972]. The second approach can guarantee that a verified system always satisfies the checked safety properties, and is the focus of this text.

However, state-of-the-art techniques, which have been demonstrated in pedagogic systems, are often difficult to understand and to apply to realistic systems. Furthermore, it is often difficult to determine how practical a proposed technique is from the large number of mathematical notations used. The objective of this book is to provide a more readable introduction to formal techniques that are practical for actual use. These theoretical foundations are followed by practical exercises in employing these advanced techniques to build, analyze, and verify different modules of real-time systems. Available specification analysis and verification tools are also described to help design and analyze real-time systems.

1.1 WHAT IS TIME?

Time is an essential concept in real-time systems, and keeping time using accurate clocks is thus required to ensure the correct operations of these systems. The master source for time is Paris's International Atomic Time (TAI), an average of several laboratory atomic clocks in the world. Since the earth's rotational rate slows by a few milliseconds each day, another master time source called the Universal Coordinated Time (UTC) performs leap corrections to the time provided by TAI while maintaining TAI's accuracy, making the time length of every natural solar day constant [Allan, Ashby, and Hodge, 1998].

UTC is used as the world time, and UTC time signals are sent from the National Institute of Standards and Technology (NIST) radio station, WWVB, in Fort Collins, Colorado, and other UTC radio stations to specialized receivers. Selected radios, receiver-clocks, some phone-answering systems, and even some VCRs have the capability to receive these UTC signals to maintain accurate clocks. Some computers have receivers to receive these UTC signals and thus the time provided by their internal clocks is as accurate as UTC. Note that depending on the location of the receiver-clock, there is a delay in receiving the UTC signal. For instance, it takes around 5 ms for WWVB's UTC signal to get from Fort Collins to a receiver-clock in my Real-Time Systems Laboratory in Houston, Texas.

For computers whose time is kept by quartz-based computer clocks, we must ensure that these clocks are periodically synchronized such that they maintain a bounded drift relative to UTC. Software or logical clocks can be derived from computer clocks [Lamport, 1978]. In this text, when we refer to wall clock or absolute

time, we refer to the standard time provided by a bounded-drift computer clock or UTC. Thus there is a mapping *Clock:* real time → standard clock time.

1.2 SIMULATION

Simulation consists of constructing a model of an existing system to be studied or a system to be built and then executing actions allowed in this model. The model can be a physical entity like a scale clay model of an airplane or a computer representation. A computer model is often less costly than a physical model and can represent a non-computer entity such as an airplane or its components as well as a computer entity such as a computer system or a program. A computer model also can represent a system with both computer and non-computer components like an automobile with embedded computer systems to control its transmission and brakes.

This physical or computer model is called the *simulator* of the actual system. A simulator can carry out simulated executions of the simulated system and display the outcomes of these executions. A physical model of an airplane in a wind tunnel shows the aerodynamics of the simulated plane that is close to the actual plane. A software simulator on a single-processor system shows the performance of a network of personal computer workstations under a heavy network traffic condition. A software simulator can also be designed to simulate the behavior of an automobile crashing into a concrete barrier, showing its effects on the automobile's simulated occupants. Sometimes a simulator refers to a tool that can be programmed or directed without programming to mimic the events and actions in different systems. This simulator can be either computer-based (software, hardware, or both) or non-computer-based.

Simulation is an inexpensive way to study the behavior of the simulated system and to study different ways to implement the actual system. If we detect behavior or events that are inconsistent with the specification and safety assertions, we can revise the model and thus the actual system to be built. In the case in which we consider several models as possible ways to implement the actual system, we can select the model that best satisfies the specification and safety assertions through the simulation and then implement it as the actual system.

Different levels of details of the actual system, also called the *target system*, can be modeled and its events simulated by a simulator. This makes it possible to study and observe only the relevant parts of the target system. For example, when designing and simulating the cockpit of an aircraft, we can restrict attention to that particular component by simulating only the cockpit with inputs and outputs to the other aircraft components, without simulating the behavior of the entire aircraft. In the design and simulation of a real-time multimedia communication system, we can simulate the traffic pattern between workstations but need not simulate the low-level signal processing involved in the coding and transmission processes if the performance is not affected by these low-level processes. The ability to simulate a target system at different detail levels or only a subset of its components reduces the resources needed for the simulation and decreases the complexity of the analysis of the simulation.

There are several simulation techniques in use, such as real-time-event simulation and discrete-event simulation. A real-time-event simulator like a physical scale model of an automobile performs its actions in real-time and the observable events are recorded in real-time. An example is the crash testing of the physical model of an automobile. Such a simulator requires recording instruments capable of recording events in real-time. When the physical model is actually an implemented target system, this is no longer a simulation but rather a *testing* of the actual system, as discussed below.

A discrete-event simulator, on the other hand, uses a logical clock(s) and is usually software-based. The variety of systems that can be represented by such a simulator is not limited by the speed at which the hardware executes the simulator since the simulated actions and events do not occur in real-time. Rather, they take place according to the speed of the simulator hardware and the instructions in the simulator program. Examples include the simulation of a network of computers in a single-processor system or the simulation of a faster microprocessor in a slower processor as in the design of popular next-generation personal computer microprocessors. Here, the appropriate actions and events "occur" at a particular logical time (representing real-time in the target system) depending on previous and current simulated actions and events. Entire books have been written that are devoted to discrete-event simulation.

Variations of simulation include the hybrid simulation approach, in which the simulator works with a partial implementation of the target system by acting as the non-implemented part. As in the case of using only a simulator, the simulator here makes it possible to predict the performance and behavior of the target system once it is completely implemented.

One main disadvantage of simulation as a technique to analyze and verify real-time systems and other systems is that it is not able to model all possible event-action sequences in the target system where the domain of possible sequences of observable events is infinite. Even when this domain is finite, the number of possible events is so large that the most powerful computer resources or physical instruments may not be able to trace through all possible sequences of events in the simulated target system.

1.3 TESTING

Testing is perhaps the oldest technique for detecting errors or problems in implemented software, hardware, or non-computer systems. It consists of executing or operating (in the case of a non-computer system) the system to be tested using a finite set of inputs and then checking to see if the corresponding outputs or behavior are correct with respect to the specifications. To test a real-time system, the values as well as the timing of the inputs are important. Similarly, both the output values and the time at which they are produced must be checked for correctness.

Many approaches have been developed for testing software, hardware, and non-computer systems. The simplest technique is of course to perform an exhaustive test run of the system with every possible input and then to check if the corresponding output is correct. This approach is not practical except for small systems with limited

input space. For larger systems, the time needed to test is prohibitively long. For systems with an infinite number of possible inputs, this approach is of course not viable. Since relatively little training is required on the part of the testing personnel, testing has been and will continue to be used extensively in industry.

There are three common techniques for software testing in the current state-of-the-practice: functional testing, structural testing, and code reading. *Functional testing* uses a "black box" approach in which the programmer creates test data from the program specification using one or a combination of techniques such as boundary value analysis and equivalence partitioning. Then the program is executed using these test data as input, and the corresponding program behavior and output are compared with those described in the program specification. If the program behavior or output deviates from the specification, the programmer attempts to identify the erroneous part of the program and correct it.

Partition testing is a popular way to select test data, in which the program's input domain is divided into subsets called subdomains, and one or more representatives from each subdomain are chosen as test input. Random testing is a degenerate form of partition testing since it has only one subdomain, the entire program.

Structural testing is a "white box" approach in which the programmer examines the source code of the program and then creates test data for program execution based on the percentage of the program's statements executed.

Code reading by *stepwise abstraction* requires the programmer to identify major modules in the program, determine their functions, and compose these functions to determine a function for the whole program. The programmer then compares this derived function with the intended function as described by the program specification.

1.4 VERIFICATION

The previous two techniques are good for revealing errors in the simulated or actual system but usually cannot guarantee that the system satisfy a set of requirements. To apply formal verification techniques to a real-time system, we must first specify the system requirements and then the system to be implemented using an unambiguous specification language. Since the applications expert (programmer or system designer) is usually not knowledgeable in formal methods, a formal methods expert collaborates with the applications expert to write the requirements and system specification. Both experts work closely to ensure that the specifications reflect the real requirements and system's behavior.

Once these specifications are written, the formal methods expert can verify whether the system specification satisfy the specified requirements using his/her favorite formal verification methods and tools. These formal methods and tools can show the satisfaction of all requirements or the failure to satisfy certain requirements. They may also pinpoint areas for further improvement in terms of efficiency. These results are communicated to the applications expert who can then revise the system specification or even the system requirements. The formal specifications are next revised to reflect these changes and can be analyzed again by the formal methods

expert. These steps are repeated until both experts are happy with the fact that the specified system satisfies the specified requirements.

1.5 RUN-TIME MONITORING

Despite the use of the best state-of-the-art techniques for static or pre-run-time analysis and verification of a real-time system, there will often be system behavior that was not anticipated. This unexpected behavior may be caused by events and actions not modeled by the static analysis tools or may be the result of making simplified assumptions about the real-time system. Therefore, it is necessary to monitor the execution of the real-time system at run-time and to make appropriate adjustments in response to a monitored behavior that violates specified safety and progress constraints. Even if the real-time system meets the specified safety and progress constraints at run-time, monitoring may provide information that can improve the performance and reliability of the monitored system.

Here, the monitored real-time system is the target system and its components, such as programs, are called target programs. The monitoring system is the system used to monitor and record the behavior of the target system. It consists of instrumentation programs, instrumentation hardware, and other monitoring modules. Basically, the monitoring system records the behavior of interest of the target system and produces event traces. These event traces may be used on-line as a feedback to the real-time controller or may be analyzed off-line to see if the target system needs to be fine-tuned.

There are two broad types of monitoring techniques: intrusive and nonintrusive. *Intrusive monitoring* uses the resources of the target system to record its behavior and thus may alter the actual behavior of the target system. A simple example is the insertion of print statements in a target program to display the values of the program variables. Another example is the extra statements inserted in the programs of computing nodes to record their states and the exchanges of these state variables in taking a snapshot of a distributed real-time system. A non-computer example is the addition of speed and impact-force sensing instruments in an automobile to record its performance. In all these monitoring cases, the monitoring system's use of the target system's resources (processor, memory, electricity, fuel) may change the target system's behavior. The degree of intrusion varies in different monitoring systems, and different target systems may accept monitoring systems with different degrees of intrusion or interference.

Nonintrusive monitoring, however, does not affect the timing and ordering of events of the monitored target system. This is especially important in the monitoring of real-time systems where both timing and ordering of events are critical to the safety of the real-time systems. An example is the use of additional processor(s) to run monitoring programs used to record the target system's behavior in a real-time environment.

The availability of monitoring systems does not mean that we can relax on the task of stringent pre-run-time analysis and verification of the target system. Rather,

monitoring should serve as an additional guarantee on the safety and reliability of the safety-critical target real-time system.

1.6 USEFUL RESOURCES

A comprehensive listing of available formal methods for the specification, analysis, and verification of both untimed and real-time systems can be found in:

`http://www.afm.sbu.ac.uk`

The website of the IEEE Computer Society's Technical Committee on Real-Time Systems (IEEE-CS TC-RTS), which contains useful information on resources, conferences, and publications in all aspects of real-time systems, is:

`http://cs-www.bu.edu:80/pub/ieee-rts/`

Major conference proceedings in the field of real-time systems include:

- Proceedings of the Annual IEEE-CS Real-Time Systems Symposium (RTSS)
- Proceedings of the Annual IEEE-CS Real-Time Technology and Application Symposium (RTAS)
- Proceedings of the Annual Euromicro Conference on Real-Time Systems (ECRTS)
- Proceedings of the ACM SIGPLAN Workshop on Languages, Compilers, and Tools for Embedded Systems
- Proceedings of the International Conference on Real-Time Computing Systems and Applications (RTCSA)

Major conference proceedings in the field of formal verification include:

- Proceedings of the Conference on Computer Aided Verification (CAV)
- Proceedings of the Conference on Automated Deduction (CADE)
- Proceedings of the Formal Methods Europe (FME) Conference
- Proceedings of the IEEE Symposium on Logic in Computer Science (LICS)
- Proceedings of the Conference on Rewriting Techniques and Applications (RTA)
- Proceedings of the Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX)
- Proceedings of the International Conference on Logic Programming (ICLP)
- Proceedings of the Conference on Formal Techniques for Networked and Distributed Systems (FORTE)

Major journals that publish articles on real-time systems include:

- *Journal of Real-Time Systems* (JRTS)
- *IEEE Transactions on Computers* (TC)
- *IEEE Transactions on Software Engineering* (TSE)
- *IEEE Transactions on Parallel and Distributed Systems* (TPDS)

Practice and products-oriented magazines that publish articles on embedded and real-time systems include:

- *Dedicated Systems*
- *Embedded Developers Journal*
- *Embedded Systems Programming*
- *Embedded Linux Journal*
- *Embedded Edge*
- *Microsoft Journal for Developers MSDN*
- *IEEE Software*