Chapter 1

An Overview of Embedded Software

In This Chapter

- Basic concepts of embedded systems and software
- Trends in embedded software development
- Embedded systems categorized
- Issues and challenges in embedded software development

Without our even being aware of it, we are surrounded by embedded systems. In our daily lives, we use a number of embedded systems — such as those found in TVs, VCRs, toasters, CD players, digital cameras, and mobile phones. Thanks to advances in semiconductor electronics, embedded systems have recently become increasingly powerful. And whereas writing software for these embedded systems used to be akin to "rocket science," this is no longer the case. However, the art and science of embedded software development is changing very rapidly.

What Is an Embedded System?

The desktop computers with which most of us are familiar, as well as workstations and mainframe computers, are "general purpose" computers in that we use them for performing a variety of everyday tasks such as playing games, word processing, accounting, scientific calculations,, and the like. In contrast, an embedded system performs a single well-defined task. An embedded system can be described as one consisting of a processor, associated peripherals, and software used for a specific purpose. For example, the VCR's very specific task is to record and play videocassettes. The hardware and the software of the VCR's embedded system are designed to carry out this specific task only.

Like any other computing system, an embedded system is a combination of hardware and software. However, the hardware of a general-purpose computer is typically pre-defined, consisting of a CPU, a monitor, a keyboard, a mouse, and secondary storage devices such as CD-ROM and/or 3.5" disk drives, and we procure it off the shelf. In the case of an embedded system, the hardware is generally custom-built for the system's specific purpose, mainly because the requirements of each system vary considerably. Consider the following two examples of typical embedded systems, illustrating the varying requirements of the hardware:

- In a manufacturing unit of a chemical plant, a number of temperatures have to be measured and based on the values, and certain operations need to be performed, such as opening a valve. The input to the system comes from various sensors that measure the temperatures, and the output is a signal that controls a valve. In other words, the input tells the device when it should do something, and the device does it. The hardware, obviously, must be based on this specific requirement if the device is to perform its function.
- The processor in a mobile phone needs to carry out a great deal of communication protocol processing to make telephone calls, and it also needs to perform voice processing to send and receive speech signals. The mobile phone is another example of an embedded system. However, unlike in the preceding example, the mobile phone has a mass market, and the hardware developers must keep in mind the specific requirements of the protocols used in the mobile networks. For instance, the mobile phones used in North America are different from those used in Europe because the network protocols are different. Hence the hardware is different. Differences in hardware can also arise because of the differing capabilities of the mobile phones.

Hence, depending on the quantity and functionality needed, the embedded system hardware needs to be developed with the cost kept in mind. Accordingly, the software that goes into the system also varies widely to meet the desired functionality.

Categories of Embedded Systems

Embedded systems can be broadly divided into the following categories. This categorization is based on whether the system has to work as an independent unit or it has to be networked, whether it has to perform real-time operations or not, and so on.

Stand-alone embedded systems

Stand-alone embedded systems work, as the name suggests, in a stand-alone mode (as opposed to a networked mode), taking input and producing output. The input can be electrical signals from sensors, or commands from a human being, such as the pressing of a button. The output can be electrical signals to drive another system, or an LED or LCD display for displaying information to users. Many embedded systems for process control in manufacturing units and automobiles fall into this category. In process control systems, the inputs come from transducers that convert a physical entity, such as temperature, into an electrical signal. The electrical signals become the output that can control devices such as valves. In some stand-alone systems, the deadlines to carry out a specific task may not be very strict; a few milliseconds this way or that way may not matter much. In other words, the response time is not crucial. For example, an air conditioning unit can be set to turn on when the temperature reaches a certain level. Other examples in this category are toys, CD players, and measuring instruments.

Real-time embedded systems

Some embedded systems are required to carry out specific tasks in a specified amount of time. Such systems are called *real-time embedded systems*. Consider, for example, a system that has to open a valve within 30 milliseconds when the humidity crosses a particular threshold. If the job is not carried out within that 30-millisecond period, a catastrophe may ensue. Such systems in which real-time constraints have to be strictly met are called *hard real-time embedded systems*. Real-time embedded systems are also extensively used in process control, when time-critical tasks have to be carried out. (Real-time "constraints" are often imposed in voice and video communication, but if these constraints are violated occasionally, nothing catastrophic will happen — for instance, a voice packet may be delayed, resulting in a short silence. Systems in which real-time constraints are present but not critical are called *soft real-time embedded systems*.)

Networked appliances

Some embedded systems are connected to a network — typically, one based on a TCP/IP (Transmission Control Protocol/Internet Protocol) protocol suite, such as the Internet or a company intranet. These systems are of a new brand that has emerged in recent years. They run the complete TCP/IP protocol stack and can communicate with other nodes on the network. Even a Web server running HTTP (HyperText Transfer Protocol) can be embedded into the system. The system can monitor some parameters (such as temperatures and pressures) and send the data over a network to a centralized system for online monitoring. A typical example is the monitoring of equipment in a manufacturing unit. The system sends the data over the TCP/IP network to a central management system, which can be a desktop computer running a Web browser. Another example is a network-enabled Web camera that can transmit its video output over the Internet. These are known as *Internet information appliances*.

Mobile devices

With the advent of wireless networks that can support very high speeds, mobile devices are capable of supporting high data rate services in addition to the voice services. Accessing Internet services such as e-mail, the World Wide Web, and so on can be done while a person is on the move. Such devices need to run a powerful mobile operating system and download information from Web sites. Many such devices are Java-enabled; hence, they can download Java applets and execute them. Field personnel can use devices such as handheld PCs and palmtops for data collection, and the data entered can be kept in local memory and later uploaded into the corporate databases. Hence, these mobile devices are capable of storing databases locally in their memory. The special requirements of these mobile devices are threefold: they should have powerful computing and communicating capabilities, they should be able to perform some real-time tasks in addition to non–real-time tasks, and they should be capable of handling multimedia applications. These requirements call for a powerful processor, a powerful operating system, and a lot of memory with minimal power consumption.

This book addresses software development for all four categories of embedded systems, using a variety of operating systems and development tools.

Requirements of Embedded Systems

Despite the varying types of embedded systems, they all have some common requirements. The following sections describe each of these fundamental characteristics.

Reliability

Embedded systems have to work without the need for the rebooting or resetting typical of many desktop systems. This calls for very reliable hardware and software. If, for example, the embedded system comes to a halt because of a hardware error, the system should reset itself without the need for human intervention. Certainly, reliability is critical in any system, but we are used to resetting our desktop systems once in a while, thanks to the unpredictability of operating systems and software. However, embedded software developers must make the reliability of the hardware, as well as that of the software, of paramount importance because the system is not readily accessible to human intervention.

Cost-effectiveness

If an embedded system is designed for a very special purpose, such as for a deep space probe or for use in a nuclear plant, cost may not be an issue. However, if the embedded system is for the mass market, such as those used in CD players, toys, and mobile devices, cost is a major consideration. Choosing the right processor, memory devices, and peripherals to meet the functionality and performance requirements while keeping the cost reasonable is of critical importance. In such cases, the designers will develop an Application Specific Integrated Circuit (ASIC) or an Application Specific Microprocessor to reduce the hardware components and hence the cost. Typically, a developer first creates a prototype by writing the software for a general-purpose processor, and subsequently develops an ASIC to reduce the cost.

Low power consumption

Many embedded systems are powered by batteries, rather than a main supply. In such cases, the power consumption should be minimized to avoid draining the batteries. Hardware designers must address this issue — for example, by reducing the number of hardware components, or by designing the processor to revert to low power or "sleep" mode when there is no operation to perform.

Efficient use of processing power

A wide variety of processors with varying processing powers are available to embedded systems. Developers must keep processing power, memory, and cost in mind when choosing the right processor. *Processor* is the term generally used to refer to a micro-controller, a microprocessor, or a Digital Signal Processor (DSP) used in embedded systems. The processing power requirement is specified in million instructions per second (MIPS). The MIPS requirement for the application has to be estimated first, and, given this estimate, the developer can choose the processor. With the availability of many processors with the same capabilities, choosing a processor has become a tough task nowadays. Considerations such as the cost of development tools, support provided by the vendor, and previous experience all play a role in the choice.

Efficient use of memory

Most embedded systems do not have secondary storage such as hard disk. The memory chips available on the embedded system are only Read-Only Memory (ROM), to hold the program; and Random Access Memory (RAM), to hold the data. Depending on the functionality, the developer may determine the program size and the data size based on which memory requirements are more important. The cost of memory is certainly going down, but even one dollar can make a big difference, particularly with regard to consumer items. As most embedded systems do not have secondary storage, *flash memory* is used to store the program, including the operating system. Micro-controllers and DSPs come with onboard memory (i.e., the same silicon contains the memory as well). Such processors are used for small embedded systems, as the cost generally is low and the execution generally is fast.

Appropriate execution time

In real-time embedded systems, certain tasks must be performed within a specified time. Consequently, analyzing the tasks to meet such performance constraints is of considerable importance. Normally, desktop PCs cannot achieve real-time performance. Therefore, special operating systems, known as real-time operating systems, run on these embedded systems. In hard real-time embedded systems, which are subject to very strict deadlines for performing specific tasks, the timing analysis is of great importance. In soft real-time systems, occasionally the task may not be performed in a timely manner. The software developer needs to ascertain whether the embedded system is a hard real-time system or a soft real-time system and perform the performance analysis accordingly. The performance requirement also calls for code optimization to the maximum possible extent. Though software is generally developed in a high-level language, some computation-intensive portions of the code may need to be written in assembly language in order for the desired real-time performance to be realized. In such cases, of course, you need to learn the assembly language of the target processor.

Challenges and Issues in Embedded Software Development

Developers of embedded systems have to deal with unique issues. These issues pose interesting challenges, which are discussed in the following sections.

Co-design

Because an embedded system consists of hardware and software, deciding which functions of the system should be implemented in hardware and which functions should be implemented in software is a major consideration. For example, consider the Cyclic Redundancy Check (CRC) that is used to check whether a bit stream has errors in it. Using a mathematical algorithm, CRC is calculated for a bit pattern, and the CRC is used to find errors in transmission. The algorithm can be implemented either in hardware or in software. Should you choose hardware implementation or software implementation? This is always a difficult question to answer. Hardware implementation has the advantage that the task execution is faster compared with that of software implementation. On the downside, a chip costs money, consumes valuable power, and occupies space. A software implementation is better if these are major concerns for a particular project. Moreover, software implementation offers flexibility — if the algorithm changes, it is easier to modify the software. This issue of choosing between hardware implementation and software implementation is known as a *co-design* issue. Co-design issues should be resolved between hardware engineers and software engineers, keeping in mind the foregoing considerations.

Another co-design example is the implementation of the TCP/IP stack. For decades, the stack has been implemented only through software, as it offers flexibility to change the protocols. The TCP/IP stack is bundled with the personal computer's operating system. This is acceptable because desktop computers have a lot of primary memory (RAM), as well as secondary storage. Now, however, a single-chip implementation of the TCP/IP stack is available, the use of which significantly accelerates the processing of the protocols. Another advantage of the TCP/IP chip solution is that it can be integrated into embedded system hardware, making the embedded system network-enabled. TCP/IP software implementation calls for huge amounts of memory (RAM), which is difficult to accommodate on embedded systems or expensive for a given application.

Embedding an operating system

It is possible to write embedded software without any operating system embedded into the system. Developers can implement services such as memory management, input/output management, and process management with a combination of assembly language and a high-level language such as C. It is also possible to build the software above the operating system. Which is the better choice? Writing your own routines necessary for a particular application results in compact and efficient code, thus saving on processing power and memory.