CHAPTER 1

OVERVIEW OF EMBEDDED SYSTEM

An embedded system is a special type of computer system. In this chapter, we examine the basic characteristics of an embedded system, highlight its differences from a general-purpose computer system, and introduce the concept and development flow of a "high-end" FPGA-based embedded system, which the focus of this book.

1.1 INTRODUCTION

1.1.1 Definition of an embedded system

An *embedded system* (or *embedded computer system*) can be loosely defined as a computer system designed to perform one or a few specific tasks. The computer system is not the end product but a dedicated "embedded" part of a larger system that often includes additional electronic and mechanical parts. By contrast, a *general-purpose computer system*, such as a PC (personal computer), is a general computing platform and itself is the end product. It is designed to be flexible and to support a variety of end-user needs. Application programs are developed based on the available resource of the general-purpose computer system.

Since an embedded system is dedicated to specific tasks, its design can be optimized to reduce cost. A good design should contain just enough hardware resources to meet the application's required functionalities. On the other hand, a general-purpose computer system is expected to support a variety of needs and thus an application program is provided with a relatively abundant hardware resource. From this perspective, an embedded system can be thought of as *a computer system with severely resource constraint.*

The terms "embedded system" and "general-purpose computer system" are not strictly defined, as most systems have some elements of extensibility or programmability. For example, a cell phone can be treated as an embedded system since it is mainly for wireless communication. However, an advanced phone allows users to load other types of applications, such as simple video games, and thus exhibits the characteristics of a general-purpose computer system.

In our book, we refer to a general-purpose computer system as a "desktop system" since a desktop computer it is the most commonly used general-purpose system.

1.1.2 Example systems

Embedded systems are used in a wide range of applications and each application has its own specific requirements. We examine three example systems to illustrate the basic characteristics of embedded applications:

- Microwave oven.
- Digital camera.
- Vehicle stability control system.

Microwave oven A microwave oven cooks or heats food with microwave radiation generated by a magnetron. A microwave oven usually has a keypad to select the cooking time and power level and an LCD or LED display that shows the status or time. It contains an embedded computer that processes the keypad input, keeps track of timing, generates the display patterns, and controls the magnetron unit.

The operation of the microwave oven requires no extensive computation and does not involve high-speed data transfer. The tasks can be accomplished by a very simple 8-bit processor (i.e., a processor with 8-bit internal data width) and a small read-only program memory. The entire embedded system can be implemented by a *microcontroller*, which is usually a single IC chip containing the 8-bit core processor, small memory, and simple I/O peripherals.

The microwave oven is a representative "low-end" embedded system.

Digital camera A digital camera takes photographs by recording images electronically via an image sensor and stores the digitized image in a flash memory card. The image sensor contains millions of pixel sensors. A pixel sensor converts light to an electronic signal. The output of the pixel sensors is digitized and stored as an image file. A typical digital camera contains a set of buttons and knobs to control and adjust camera operation and a small LCD display to preview the stored pictures.

The embedded system in the camera performs two major tasks. The first task involves the general "housekeeping" I/O operations, including processing the button and knob activities, generating the graphic on an LCD display, and writing image files to the storage device. These operations are more involved than those of a microwave oven and the system requires a more capable 16- or 32-bit processor as well as a separate memory chip. The second task is to process the image and perform data compression to reduce the file size. Because of the large number of pixels and the complexity of the compression algorithm, it requires a significant amount of computation. An embedded processor is usually not powerful enough to handle the computation-intensive operation. A custom digital circuit can be designed to perform this particular task and take the load off the processor. This type of circuits is known as *hardware accelerators*.

The digital camera is a representative "high-end" embedded system.

Vehicle electronic stability control system A vehicle ESC (electronic stability control) system helps to improve a vehicle's maneuverability by detecting and minimizing skids. During driving, it continues comparing the driver's intended direction with the vehicle's actual direction. When the loss of steering control is detected (e.g., due to a wet or iced surface), the ESC system intervenes automatically and applies the brakes to individual wheels to steer the vehicle to the intended direction.

The embedded system obtains the intended direction from the steering wheel angle and obtains the actual direction from the vehicle lateral acceleration and the individual wheel's rotating speed. It determines the occurrence and nature of the skid and then calculates and applies brake forces to individual wheels to offset the skid condition.

The ESC embedded system has two special characteristics. First, the ESC system imposes a *real-time constraint* — an operational deadline from the triggering event (i.e., onset of skid condition) to the system response (i.e., application of the brake forces). The system fails to work if the brake is not applied within a specific amount of time. Second, since the steering concerns the driver's safety, the embedded system is *mission critical* and thus must be robust and reliable.

1.2 SYSTEM DESIGN REQUIREMENTS

When designing a computer system, we must consider a variety of factors:

- Cost
- General computation speed
- Special computation need
- Real-time constraint
- Reliability
- Power consumption

The term *special computing need* means the type of computation task, such as data compression, encryption, pattern recognition, etc., which cannot be easily accomplished by a general-purpose processor.

In general, we wish that every computer system would be inexpensive, fast, reliable, and would use little power. However, these criteria are frequently fighting against each other. For example, a faster processor is more expensive and consumes more power. An embedded system can be used in a wide range of applications and each system has its own unique needs. For each system, we need to identify the key requirements and seek the best trade-off. One way to illustrate these requirements is to use a "radar chart" shown in Figure 1.1. There are six axes in the chart, each indicating the importance of a factor. As a point in an axis moves outward from the center, its importance increases from "not important" to "extremely important."

A desktop PC is for general use and thus does not place weight on a particular factor. Its chart is "well rounded," as shown in Figure 1.1(a). A microwave oven

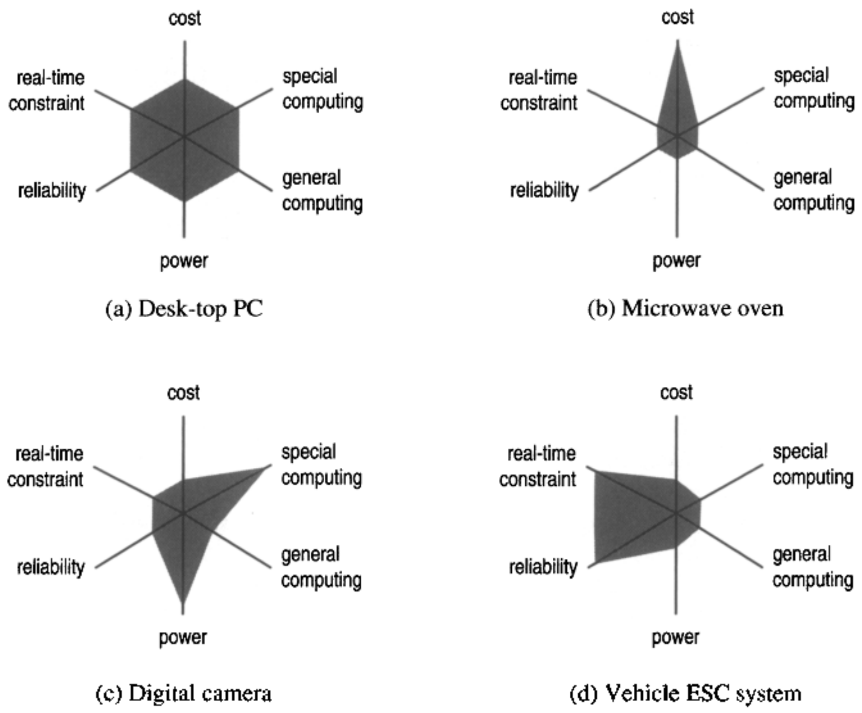

Figure 1.1 Radar charts of various systems.

can be considered as a "commodity" and its profit margin is not very high. Thus, it is extremely sensitive to the part cost. The embedded system for the microwave is very simple and its key requirement is to reduce the cost. Its chart is shown in Figure 1.1(b). A digital camera requires special image processing and compression capability. Since it is a handheld device powered by a battery, reducing power usage is important. Thus, the two key requirements of the camera's embedded system are the power and special computation need. Its chart is shown in Figure 1.1(c). A vehicle ESC system imposes a strict operational deadline and is mission critical. The key requirements of the ESC embedded system are the real-time constraint and reliability. Its chart is shown in Figure 1.1(d).

From the requirement's point of view, we can treat an embedded system as a computer system with extreme design requirements.

1.3 EMBEDDED SOPC SYSTEMS

The main focus of this book is on the "high-end" embedded systems similar to the digital camera. This type of system usually has a processor and simple I/O peripherals to perform general user interface and housekeeping tasks and special hardware accelerators to handle computation-intensive operations. These components can be integrated into a single integrated circuit, commonly referred to as *SoC* (*system on a chip*). As the capacity of *FPGA* (*field-programmable gate ar-*

ray) devices continues to grow, the same design methodology can be realized in an FPGA chip and is sometimes known as *SoPC* (*system on a programmable chip*) or *PSoC* (*programmable system on a chip*). We use the term SoPC in the book.

While designing a system based on a conventional embedded processor, we examine the required functionalities and then select a processor, external I/O peripherals, and ASSP (application specific standard product) devices to construct the hardware platform. Because of the fixed-sized processor architecture, a limited choice of ASSP devices, and the cost of manufacturing printed circuit boards, the hardware configuration is usually rather "rigid" and the desired system functionalities are usually done by *customized software*.

An FPGA device contains logic cells and interconnects that can be configured (i.e., "programmed") to perform a specific function. The desired hardware functionalities are usually described in *HDL* (*hardware description language*) code, which is then synthesized and implemented by the FPGA device. Because of the programmability of FPGA devices, *customized hardware* can be incorporated into the embedded system as well. We can tailor the processor, select only the needed I/O peripherals, create a custom I/O interface, and develop specialized hardware accelerators for computation-intensive tasks. The SoPC-based embedded system provides a new dimension of flexibility because both the hardware and software can be customized to match specific needs.

1.3.1 Basic development flow

The embedded SoPC system development consists of the following parts:

- Partition the tasks to software and hardware accelerators.
- Develop the hardware, including the hardware accelerators and I/O peripherals, and integrate it with the processor.
- Develop the software.
- Implement the hardware and software and perform testing.

Since the design examples in this book are targeted for Altera prototyping boards, our discussion uses the Altera development platform and its *Nios II* processor. Note that Nios II is a *soft-core* processor, which means the processor is described in HDL code and synthesized later by using FPGA's generic logic cells.

The basic Nios II-based development flow is shown in Figure 1.2. The four basic parts are elaborated in the following subsections.

Software–hardware partition Step 1 (labeled 1 in the diagram) is to determine the software–hardware partition. An embedded application usually performs a collection of tasks. In an SoPC-based design, a task can be implemented by hardware, software, or both. Based on the performance requirement, complexity, and hardware core availability, we can decide the type of implementation accordingly.

Hardware development flow The left branch represents the hardware design flow. Step 2 derives the basic hardware architecture. The custom hardware can be divided into three categories:

- *Nios II processor and standard I/O peripherals* (labeled "Nios configuration" in the diagram). Altera provides the soft cores of the processor and a collection of frequently used I/O peripherals. A third-party vendor supplies

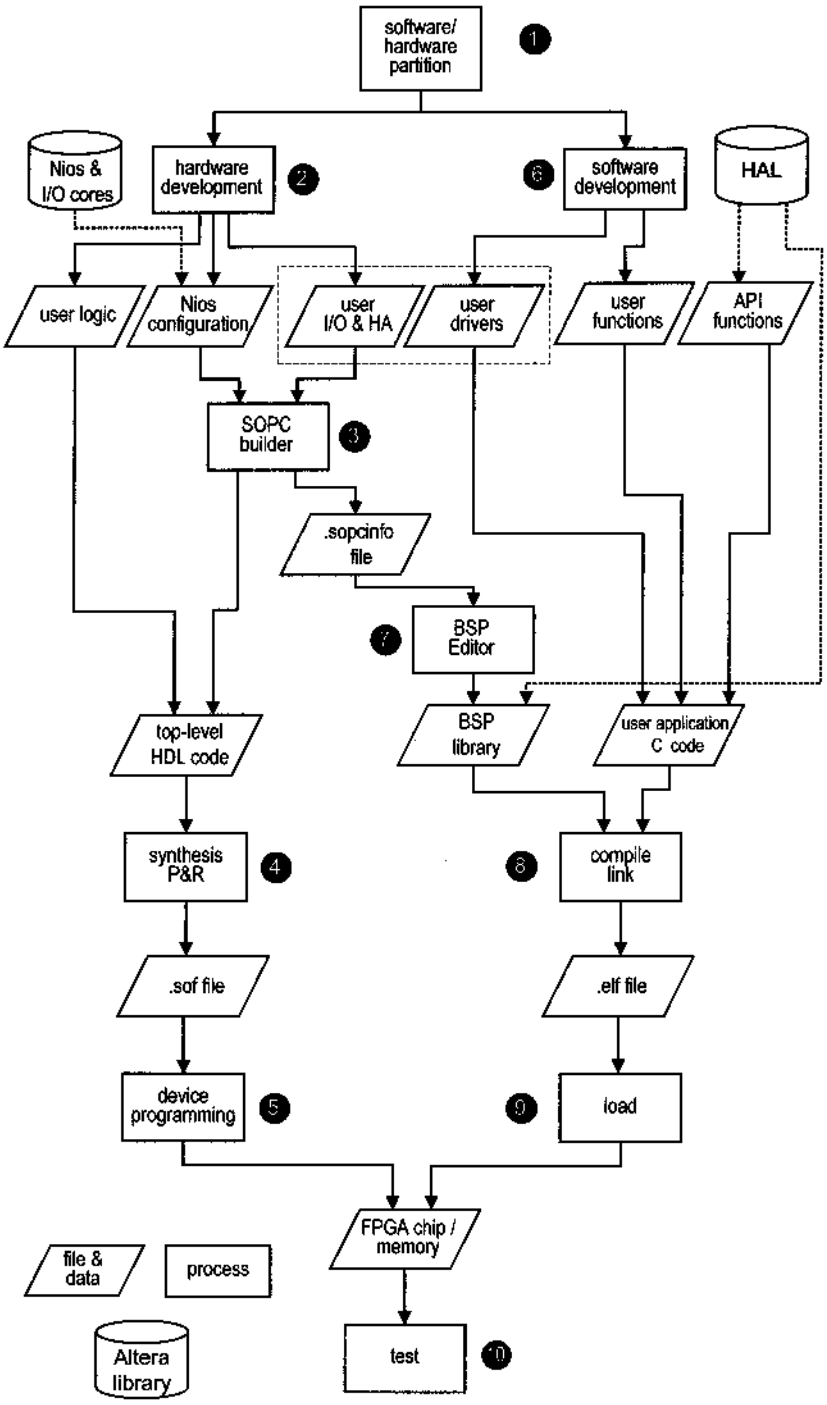

Figure 1.2 Development flow of a system with Nios II.

additional I/O cores as well. We can select the needed I/O peripherals and configure the basic Nios II system.

- *User I/O peripherals and hardware accelerators* (labeled "User I/O & HA" in the diagram). For certain specialized I/O functions or computation-intensive tasks, a pre-designed core may not exist or cannot satisfy the performance requirement. We must design the hardware from scratch and integrate it into the Nios II system as a custom I/O peripheral.
- *User logic*. Some portion of the hardware may be separated from the Nios II system. It is not attached to the Nios interconnect structure and does not interact directly with the processor.

Step 3 generates the HDL code from the customized Nios II system. It is done by using Altera's *SOPC Builder* software package. In this software, we can configure the processor, select the desired standard I/O cores, and incorporate the user-designed I/O peripherals. SOPC Builder then generates the HDL codes for the customized Nios II system and also generates the .sopcinfo file that contains system configuration information. We can combine this code with the other use logic codes to form the final top-level HDL description.

The top-level HDL code contains the description of the complete hardware. Step 4 performs synthesis and placement and routing and eventually generates the FPGA configuration file (i.e., the .sof file).

Software development flow The right branch represents the software design flow. Step 6 derives the basic software structure. Altera provides a software library, which is integrated into its *HAL* (*hardware abstraction layer*) platform, for the Nios II system. It consists of *I/O device drivers*, which are low-level routines to access I/O peripherals, and a collection of high-level functions in an *application programming interface* (*API*). From the hardware–software interface's point of view, we can divide the software code into three categories:

- *API functions*. These are the functions from the Altera HAL platform.
- *User I/O drivers*. When designing a custom I/O peripheral or hardware accelerator, we also need to develop software I/O routines to control its operation and to exchange its data with the processor.
- *User functions*. These implement the needed functionalities for the embedded application.

We can utilize these drivers and functions to construct the application program.

When a Nios II system is created, the processor and I/O configuration is recorded in the .sopcinfo file. In Step 7, the *BSP Editor* software program examines this file, extracts the needed device drivers from the HAL library, and builds up a *BSP* (*board support package*) library to support the system.

Step 8 compiles and links the software routines and BSP library and builds the final software image file (i.e., the .elf file).

Physical implementation and test Physically implementing the system involves two steps. We first download the FPGA configuration file to the FPGA device (i.e., "program" the device), as in Step 5, and then load the software image into Nios II's memory, as in Step 9. The physical system can be tested afterwards, as in Step 10.

The most unique characteristics of an SoPC-based embedded system are that custom I/O peripherals and hardware accelerators can be integrated into the system. The major task involves the development of custom hardware and a software driver, as shown in the dotted box in Figure 1.2. This is the main focus of the book.

1.4 BOOK ORGANIZATION

The remaining book is divided into four parts. Part I introduces the basic HDL constructs and synthesis procedure and discusses the development of custom digital circuits. Part II provides an overview of a Nios II-based system and embedded software development with the emphasis on low-level I/O access and drivers. A simple flashing-LED design is used to illustrate the key concepts. Part III applies the techniques from Parts I and II to design an array of complex I/O peripheral modules on the Altera DE1 prototyping board, including a PS2 keyboard and mouse controller, a graphic video controller, an audio codec controller, and an SD (secure digital) card controller. Part IV presents three case studies of the integration of hardware accelerators, including a custom GCD (greatest common divisor) circuit, a Mandelbrot set fractal circuit, and an audio synthesizer based on DDFS (direct digital frequency synthesis) methodology.

1.5 BIBLIOGRAPHIC NOTES

In this book, a short bibliographic section appears at the end of each chapter to provide the most relevant references for further exploration. A more comprehensive bibliography is included at the end of the book.

Embedded systems encompass a spectrum of design issues. The two books, *Embedded System Design: A Unified Hardware/Software Introduction* by F. Vahid and T. D. Givargis and *Computers as Components: Principles of Embedded Computing System Design, 2nd edition* by W. Wolf, provide a comprehensive discussion. Most processor-oriented embedded system books are around specific low-end microcontrollers. However, *Programming 32-bit Microcontrollers in C: Exploring the PIC32* by L. Di Jasio, as its title indicates, is based on 32-bit PIC processors and covers more advanced design examples.

Software-hardware co-design is an emerging research area. *A Practical Introduction to Hardware/Software Codesign* by P. R. Schaumont addresses the basic concepts and issues of combining hardware and software into a single system design process.