PART II

PROGRAMMING RECONFIGURABLE SYSTEMS

As suggested in the Introduction, field-programmable gate arrays (FPGAs) and reconfigurable architectures have both the postfabrication programmability of software and the spatial parallelism of hardware. To fully exploit them, we need models and programming approaches that support the software's programmability, including infrastructure and runtime support to allow the configuration to change over time. In addition to temporal reprogrammability, the reconfigurable programming systems must simultaneously deal with spatial issues normally associated only with hardware (e.g., physical placement of computations and timing of functional units).

To illustrate how we can program reconfigurable systems, the chapters in this part of the book describe the current state of the art in approaching and capturing designs for FPGAs and reconfigurable architectures. Chapter 5 reviews compute models and organizations suitable for reconfigurable applications, Chapters 6 through 10 and Chapter 12 explore different design entry points for reconfigurable applications, and Chapters 11 and 12 examine infrastructural support issues, including operating and runtime systems and debuggers.

The flexibility of FPGAs and reconfigurable architectures, as well as their dual hardware/software nature, means that the old computational models we are familiar with for hardware or software may not be the most effective for reasoning about reconfigurable designs. Furthermore, the design space for reconfigurable solutions is much larger than those most of us are used to navigating. Chapter 5 explores some useful models for capturing and conceptualizing reconfigurable applications and a variety of system architectures for providing efficient implementations. A clear conceptual model of the parallelism in the application, how to expose it, and how to exploit it make up an invaluable starting point for describing the application in a concrete programming language.

Chapter 6 provides an introduction to VHDL as an example of a Register transfer level (RTL) hardware description language. A software designer might think of VHDL as a semi-portable assembly language for reconfigurable designs; it provides fine control of hardware and

parallelism, but it demands that the designer manage quite a number of low-level details. Many of the higher-level programming approaches still use VHDL as an intermediate mapping stage on the way to a reconfigurable configuration.

Chapter 7 turns to more software-friendly approaches and shows how programs written in C can automatically be translated into reconfigurable hardware designs. Today, we cannot expect to obtain good performance from arbitrary C code with no concern for the capabilities of the reconfigurable architecture and compiler. However, with an appreciation for what reconfigurable architectures can do, an appropriate system architecture, and an understanding of the capabilities of the C compiler, it is possible to effectively develop and optimize reconfigurable applications in C.

Chapters 8 and 9 discuss two examples of programming systems that support streaming dataflow compute models (Section 5.1.3). These models, too, provide a higher-level approach to reconfigurable design than VHDL, offering greater opportunities for automated design scalability. Chapter 8 describes how we can apply the SDF (Synchronous Dataflow) model (Section 5.1.3) using Simulink, illustrating how methodology and suitable libraries can raise the abstraction for design construction. These techniques can readily be adopted by today's system designers. At the same time, the Simulink integration example shows how reconfigurable design can leverage popular system analysis tools such as MATLAB.

Chapter 9 describes a more custom and automated experimental design flow that supports application scalability for dynamic streaming dataflow applications (Section 5.1.3). It illustrates how many system architectures (Section 5.2) come together to support efficient and automated mapping of designs to reconfigurable computing platforms, and it offers a vision of how integrated programming systems for reconfigurable platforms might evolve.

Many efficient reconfigurable applications are naturally data parallel (Section 5.1.5) and are efficiently implemented with a Single Instruction Multiple Data (SIMD) or vector organization. Chapter 10 describes data parallel programming approaches customized for reconfigurable compilation.

In Chapter 12 we see an example of a rich generator language, JHDL, which provides even lower-level control of structure than VHDL, but does so with the full programming power of a conventional software language, Java. Thus, it provides a high-level platform from which to develop highly tuned designs. It also provides rich support for the construction of custom tools for reconfigurable design optimization.

As reconfigurable computers emerge as platforms for creating and delivering software, we must develop software support normally associated only with general-purpose processors, including operating systems, runtime support, and interactive debuggers. Chapter 11 describes the growing demands for reconfigurable operating systems, highlighting some of the early work along this path and pointing out important directions for the future. JHDL (Chapter 12) is notable for its support for interactive debugging and the extensible programming environment it provides, including hooks for software modules that interact with reconfigurable designs.