PART V

CASE STUDIES OF FPGA APPLICATIONS

Parts I through IV covered technologies and techniques for creating efficient FPGA-based solutions to important problems. Part V focuses on specific, important field-programmable gate array (FPGA) applications, presenting case studies of interesting uses of reconfigurable technology. While this is by no means an exhaustive survey of all applications done on FPGAs, these chapters do contain several very interesting representative points in this space. They can be read in any order, and can even be interspersed with other chapters of this book.

This introduction should help readers identify the concepts the case studies cover and the chapters each help to illustrate. To understand the case studies, a basic knowledge of FPGAs (Chapter 1), CAD tools (Chapters 6, 13, 14, and 17), and application development (Chapter 21) is required.

Chapter 27 presents a high-performance image compression engine optimized for satellite imagery. This is a streaming signal-processing application (Chapters 5, 8, and 9), a type of computation that typically maps well to reconfigurable devices. In this case, the system saw speedups of approximately 400 times, for which the authors had to optimize the algorithm carefully, considering memory bandwidth (Chapter 21), conversion to fixed point (Chapter 23), and alteration of the algorithm to eliminate sequential dependencies.

Chapter 28 focuses on automatic target recognition, which is the detection of regions of interest in military synthetic aperture radar (SAR) images. Like the compression engine in Chapter 27, this represents a very complex, streaming signal-processing application. It also is one of the most influential applications of runtime-reconfiguration (Chapters 4 and 21), where a large circuit is time-multiplexed onto a single FPGA, enabling it to reuse the same silicon multiple times. This was necessary because the possible targets to be detected were represented by individual custom, instance-specific circuits (Chapter 22), the huge number of which was too large for the available FPGAs.

Chapter 29 discusses Boolean satisfiability (SAT) solving—the determination of whether there is an assignment of values to variables that

makes a given Boolean equation true (satisfied). SAT is a fairly general optimization technique that is useful in, for example, chip testing, formal verification, and even FPGA CAD flows. This work on solving Boolean equations via FPGAs is an interesting application of instance-specific circuitry (Chapter 3) because each equation to be solved was compiled directly into FPGA logic. However, this meant that the runtime of the CAD tools was part of the time needed to solve a given Boolean equation, creating a strong push toward faster CAD algorithms for FPGAs (Chapter 20).

Chapter 30 covers logic emulation—the prototyping of complex integrated circuits on huge boxes filled with FPGAs and programmable interconnect chips. This is one of the most successful applications of multi-FPGA systems (Chapter 3) because the translation of a single ASIC into FPGA logic necessitates hundreds to thousands of FPGAs to provide adequate logic capacity. Fast mapping tools for such systems are also important (Chapter 20).

In Chapter 23 we discussed methods for eliminating (or at least minimizing) the amount of floating-point computation in FPGA designs by converting floating-point operations to fixed point. However, there are situations where floating point is unavoidable. Scientific computing codes often depend on floating-point values, and many users require that the FPGA-based implementation provide *exactly* the same results as those of a processor-based solution. These situations require full floating-point support. In other cases, the high dynamic range of values might make fixed-point computations untenable. Chapter 31 considers the development of a library of floating-point units and their use in applications such as FFTs.

Chapter 32 covers a complex physical simulation application—the finite difference time domain (FDTD) method, which is a way of modeling electromagnetic signals in complex situations that can be very useful in applications such as antenna design and breast cancer detection. The solution involves a large-scale cellular automata (Chapter 5) representation of the space to be modeled and an iterative solver. The key to achieving a high-performance implementation on FPGAs, however, involves conversion to fixed-point arithmetic (Chapter 23), simplification of complex mathematical equations, and careful consideration of the memory bottlenecks in the system (Chapter 21).

Chapter 33 discusses an alternative to traditional design flow for creating FPGA mappings in which the FPGA is allowed to evolve its own configuration. Because the FPGA is reprogrammable, a genetic optimization system can simply load into it random configurations and see how well they function. Those that show promise are retained; those that do

not are removed. Through mutation and breeding, new configurations are created and evaluated in the same way, slowly evolving better and better computations. The hope is that such a system can support important classes of computation with circuits significantly more efficient than standard design flows. This design strategy exploits special features of the FPGA's reprogrammability and flexibility (Chapter 4).

Some of the chapters in this section focus on streaming digital signal processing (DSP) applications. Such applications often benefit from FPGA logic because of their amenability to pipelining and because of the large amount of data parallelism inherent in the computation. Network processing and routing is another such application domain. Chapter 34 considers packet processing, the application of FPGA logic to network filtering, and related tasks. Heavy pipelining of circuits onto the reconfigurable fabric and optimization of custom boards to network processing (Chapter 3) support very high-bandwidth networking. However, because the system retains the flexibility of FPGA logic, new computations and new filtering techniques can be easily accommodated within the system. This ability to incrementally adjust, tune, and invent new circuits provides a valuable capability even in a field as rapidly evolving as network security.

For many applications, memory access to a large set of state, rather than computational, throughput can be the bottleneck. Chapter 35 explores an object-oriented, data-centric model (Chapter 5) based on adding programmable or reprogrammable logic into DRAM memories. The chapter emphasizes custom-reprogrammable chips (Chapter 2) and explores both FPGA and VLIW implementation for the programmable logic. Nevertheless, much of the analysis and techniques employed can also be applied to modern FPGAs with large, on-chip memories.