

Practical FPGA Programming in C

By David Pellerin, Scott Thibault

Donwload Read Online

Practical FPGA Programming in C By David Pellerin, Scott Thibault

High-performance FPGA-accelerated software applications are a growing demand in fields ranging from communications and image processing to biomedical and scientific computing. This text introduces powerful, C-based parallel-programming techniques for creating these applications, verifying them, and moving them into FPGA hardware.

Download Practical FPGA Programming in C ...pdf

Read Online Practical FPGA Programming in C ...pdf

Practical FPGA Programming in C

By David Pellerin, Scott Thibault

Practical FPGA Programming in C By David Pellerin, Scott Thibault

High-performance FPGA-accelerated software applications are a growing demand in fields ranging from communications and image processing to biomedical and scientific computing. This text introduces powerful, C-based parallel-programming techniques for creating these applications, verifying them, and moving them into FPGA hardware.

Practical FPGA Programming in C By David Pellerin, Scott Thibault Bibliography

- Sales Rank: #521913 in Books
- Published on: 2005-05-02
- Original language: English
- Number of items: 1
- Dimensions: 9.40" h x 1.10" w x 7.20" l, 1.75 pounds
- Binding: Paperback
- 464 pages

Download Practical FPGA Programming in C ...pdf

Read Online Practical FPGA Programming in C ...pdf

Preface

This is a book about software programming for FPGAs. To be more specific, this book is about using parallel programming techniques in combination with the C language to create FPGA-accelerated software applications.

We have written this book to help bridge the gap—the great chasm, in fact—that exists between software development methods and philosophies, and the methods and philosophies of FPGA-based digital design. We understand that as a result we may be writing for two quite different audiences: software application developers and digital hardware designers.

For software engineers, our goal is to present FPGAs as software-programmable computing resources. We hope to show that these devices, when programmed using appropriate methods, are not so different from other non-traditional computing platforms, such as DSPs. We will show, through example, that software development methods and software languages can be used in a practical way to create FPGA-based, high-performance computing applications, without a deep knowledge of hardware design.

For hardware designers our intent is similar, but with a caveat: we are not trying to replace your existing methods of design or suggest that the methods and tools described in this book represent a "next wave" of hardware engineering. After all, if you are an FPGA designer using VHDL or Verilog to create complex electronic systems, the title of this book, *Practical FPGA Programming in C*, may sound like an oxymoron. How can C, a software programming language, be a practical way to describe digital electronic hardware? The truth is, sometimes the explicit and precise descriptions offered by hardware description languages (HDLs) are essential to achieve designs goals. But, as you'll see, this explicit control over hardware is not always necessary. In the same way that you might first write software in C and then recode key portions in assembler, hardware designers can benefit from tools that allow them to mix high-level and low-level descriptions as needed to meet design goals as quickly as possible. Even when the entire hardware design will be eventually be recoded with a lower-level HDL, high-level design languages allow hardware engineers to rapidly explore the design space and create working prototypes.

So for you, the experienced hardware engineer, we'll state right up front that we agree with you. We do not believe that C and C++ (as used by legions of programmers worldwide) are practical replacements for VHDL, Verilog, or any other HDL. And we agree with you that C and C++ may not play a leading, or perhaps even significant, role as a design entry language for general-purpose ASIC and FPGA design, at least not as we know such design today. Nevertheless, we believe there is a place for C-based design in a hardware design flow. Still not convinced? Stay with us for a moment, and we'll explain.

C Language for FPGA-Based Hardware Design?

Let's think a bit more about the role of C—or lack of a role, as the case may be—in hardware design. Why is standard C not appropriate as a replacement for existing hardware design languages? Because any good programming language provides one important thing: a useful abstraction of its target. VHDL and Verilog (or, more precisely, the synthesizable subsets of these languages) succeed very well because they provide a rational, efficient abstraction of a certain class of hardware: level- and edge-sensitive registers with reset and clock logic, arbitrary logic gates, and somewhat larger clocked logic elements arranged in a highly parallel manner. All of today's FPGAs fit this pattern, and it is the pattern also found in the vast majority of today's ASIC designs, no matter how complex.

The standard C language does not provide that level of abstraction (which we call register transfer level, or RTL), so "C-based" languages for hardware could add RTL-like constructs in the form of syntax decorations,

extra functions or keywords, compiler hints, and more to create some meaningful way of expressing parallelism and describing low-level hardware structures such as clocks and resets. But in this case we would just have another HDL with new syntax. On the other hand, without the benefit of RTL constructs such as these, the developers of C compilers for FPGAs and other nontraditional targets would face a nearly impossible problem: how to efficiently map algorithms and applications written for one class of processing target (the traditional microprocessor) to something entirely different (arbitrary logic gates and registers combined with somewhat higher-level logic structures). Nobody has yet figured out how to do that mapping from a pure software application with a reasonable level of efficiency, although we are getting better at it.

So why use C at all for FPGA design? There are significant advantages, including the potential for hardware-software codesign, for the creation of test benches written in C, and (if the modified C language supports it) the ability to compile and debug an FPGA application using a standard C development environment. And if a mid-level approach to hardware abstraction is taken—one that does not require that the programmer understand all the details of the hardware target, and yet is guided by the programming model toward more appropriate methods of coding—you can strike a balance between software design productivity and hardware design results, as measured in system performance and size.

Compelling Platforms for Software Acceleration

Here's where software developers come in. On the applications side there is an increasing trend toward using FPGAs as hardware accelerators for high-performance computing. Applications that demand such performance exist in many domains, including communications, image processing, streaming media, and other general-purpose signal processing. Many of these applications are in the embedded software space, while others represent scientific, biomedical, financial, and other larger-scale computing solutions.

When acceleration is required, these applications typically begin their lives as software models, often in C language, and are then manually rewritten and implemented in hardware using VHDL or Verilog. This manual conversion of software algorithms to hardware is a process that can be long and tedious in the extreme; hence, there is a strong demand for more rapid paths to working hardware. There is also a strong desire to avoid later redesigns of that hardware to reflect software algorithm updates. Automating the process of software-to-hardware conversion—at the very least for the purpose of creating hardware prototypes—is therefore highly compelling. This automation also brings hardware acceleration within reach of design teams and developers who do not have vast hardware design expertise, or where hardware design might be cost-prohibitive using traditional approaches. C-to-hardware compilation is an enabling technology that has the potential to open up hardware acceleration for a whole new class of applications and developers.

On the hardware side, there have been recent advances in FPGA-based programmable platforms that make FPGAs even more practical for use as application accelerators. In fact, FPGAs with embedded processor cores have now become cost-effective as replacements for entire embedded systems. The concept of building a complete "system on an FPGA" is finally a reality, thanks to the continued efforts of the major FPGA providers and third-party intellectual property (IP) suppliers. What is common about these new FPGA-based platforms is that they combine one or more general-purpose processor cores with a large amount of programmable logic. Just as significantly, they come equipped with substantial amounts of royalty-free, pretested components and an ever-growing collection of licensable third-party and open-source components that can be assembled on-chip in a matter of hours to create an amazingly diverse set of potential hardware/software processing platforms.

From a tools perspective, what is interesting about these new devices—and the new computing platforms they represent—is that they are flexible enough to support the use of methods and tools from the software development world, including traditional software debuggers. Using languages such as C and C++ in

conjunction with software-to-hardware compilers makes it practical to implement software applications directly onto a mixed hardware/software platform.

It is here that FPGA designers who are questioning the use of software design languages and methods for FPGAs may be missing the point. C is not likely to replace HDLs such as VHDL and Verilog for traditional, general-purpose hardware design. C will, however, play a strong role in the kind of mixed hardware/software applications that are emerging today—applications in which the line between what is software and what is hardware is becoming increasingly blurred, and where the ability to rapidly prototype and experiment with implementation alternatives is critical.

In addition, because of the prevalence of embedded FPGA processors, the C language is already being used for FPGAs by thousands of software engineers. The problem is that the vast majority of these software engineers are only able to program for the embedded microprocessor within the FPGA. Our goal in this book is to make C-language programming applicable to the entire FPGA, not just the embedded microprocessor within it.

The Power to Experiment

Using C language programming as an aid to FPGA-based hardware design gives you the power to experiment—with alternate algorithmic approaches, with alternate hardware/software partitioning, and with alternate target platforms. Using the same design entry language, you now have the ability to evaluate your applications and their constituent algorithms using different hardware and software targets.

In this book we will show you how you can, for example, set up a test in which the same C-language algorithm can, with only minor modifications, be executed in a desktop computing environment (under the control of a C debugger such as found in Visual Studio, gbd, or Eclipse) for the purpose of functional simulation, be executed in-system on an embedded processor, and be compiled directly to an FPGA as dedicated hardware. Using this approach, fundamentally different computing alternatives can be tried, and alternative hardware/software partitioning strategies can be evaluated. As you'll see, this power to quickly generate and experiment with hardware/software prototypes is the key advantage of using software-based methods and tools for FPGAs.

How This Book Is Organized

For software and hardware developers, we present important background information about FPGA devices, their history, and the types of tools available for them, and we also survey some of the available FPGA-based computing platforms.

After setting the stage, we then present a method of programming, using the C language, and a programming model that is appropriate for use with highly parallel programmable hardware platforms. This programming model is somewhat different from traditional, more procedural C-language programming but is easy for experienced software engineers to pick up and use.

From that foundation we then move into a series of examples, all written in C, that demonstrate how to take advantage of the massive levels of parallelism that are available in an FPGA-based platform. We will describe software coding techniques that allow better optimization of C-language statements by automated compiler tools. These techniques are not difficult to understand but can have dramatic impacts on the performance of an FPGA application.

Later chapters present examples of how a streaming programming model can be used to create even higher

levels of performance. Using amultiple-process streaming programming model can result in truly astonishing levels of performance with relatively little effort, but to the traditional C programmer the methods used to achieve such levels of performance may be somewhat new and different.

Where This Book Came From

This book, like many projects in the real world, began its life as a prototype, one that had been taken off the shelf, fiddled with, and put back repeatedly over a number of years. In its initial conception—long before the grueling, months-long process to create the book you now hold—the book was to have a different title and quite a different emphasis—on reconfigurable hardware platforms, both FPGA-based and non-FPGA-based. That book—the one that didn't get written but that survives in bits and pieces in certain chapters of this book—was conceived as a follow-on to an earlier book on programmable logic devices authored by David Pellerin and Michael Holley and titled *Practical Design Using Programmable Logic* (Prentice Hall, 1991).

When the idea for such a book was presented to Bernard Goodwin at Prentice Hall, his initial response was "But of course!" (It is the nature of acquisitions editors, we suppose, to be enthusiastic about every project prior to actually seeing the awful sludge of the first sample chapters.) As we worked on refining the proposal, Bernard suggested that we increase the emphasis on one particular area, an area of which we—due to our current roles in a technology startup—might have a greater-than-usual understanding. Hence the emphasis on C programming for FPGAs and on the design and optimization of parallel applications for FPGA targets. Bernard was, of course, eyeing the bottom line and steering us in a direction that would maximize sales (C is popular and FPGAs are popular; ergo, we will attract more readers by having both terms in the title). What Bernard did not know—what none of us know, really—is how widely C programming for FPGAs can and will be accepted. Time will tell, of course. With this book we hope to convince you that such an approach is indeed a practical one.

David Pellerin
Kirkland, Washington

Scott Thibault
Colchester, Vermont

© Copyright Pearson Education. All rights reserved.

Read Practical FPGA Programming in C By David Pellerin, Scott Thibault for online ebook

Practical FPGA Programming in C By David Pellerin, Scott Thibault Free PDF d0wnl0ad, audio books, books to read, good books to read, cheap books, good books, online books, books online, book reviews epub, read books online, books to read online, online library, greatbooks to read, PDF best books to read, top books to read Practical FPGA Programming in C By David Pellerin, Scott Thibault books to read online.

Online Practical FPGA Programming in C By David Pellerin, Scott Thibault ebook PDF download

Practical FPGA Programming in C By David Pellerin, Scott Thibault Doc

Practical FPGA Programming in C By David Pellerin, Scott Thibault Mobipocket

Practical FPGA Programming in C By David Pellerin, Scott Thibault EPub