



# ADVANCED FPGA DESIGN

Architecture, Implementation, and Optimization

STEVE KILTS

# Advanced FPGA Design Architecture, Implementation, and Optimization

**Steve Kilts**

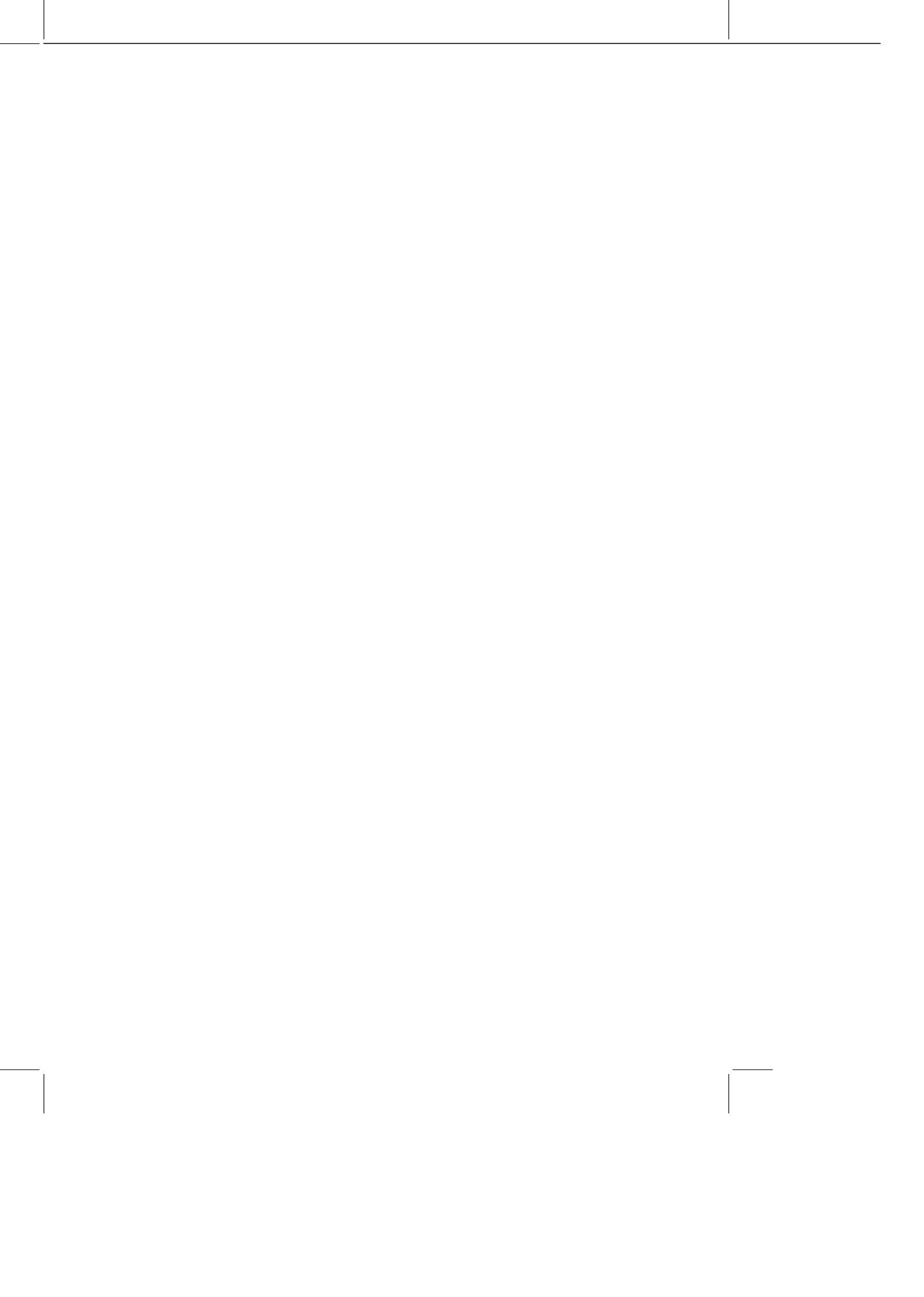
*Spectrum Design Solutions  
Minneapolis, Minnesota*



The Institute of Electrical and Electronics Engineers, Inc., New York



**WILEY-INTERSCIENCE**  
**A JOHN WILEY & SONS, INC., PUBLICATION**



# **Advanced FPGA Design**



---

**THE WILEY BICENTENNIAL—KNOWLEDGE FOR GENERATIONS**

---

Each generation has its unique needs and aspirations. When Charles Wiley first opened his small printing shop in lower Manhattan in 1807, it was a generation of boundless potential searching for an identity. And we were there, helping to define a new American literary tradition. Over half a century later, in the midst of the Second Industrial Revolution, it was a generation focused on building the future. Once again, we were there, supplying the critical scientific, technical, and engineering knowledge that helped frame the world. Throughout the 20th Century, and into the new millennium, nations began to reach out beyond their own borders and a new international community was born. Wiley was there, expanding its operations around the world to enable a global exchange of ideas, opinions, and know-how.

For 200 years, Wiley has been an integral part of each generation's journey, enabling the flow of information and understanding necessary to meet their needs and fulfill their aspirations. Today, bold new technologies are changing the way we live and learn. Wiley will be there, providing you the must-have knowledge you need to imagine new worlds, new possibilities, and new opportunities.

Generations come and go, but you can always count on Wiley to provide you the knowledge you need, when and where you need it!

**WILLIAM J. PESCE**  
PRESIDENT AND CHIEF EXECUTIVE OFFICER

**PETER BOOTH WILEY**  
CHAIRMAN OF THE BOARD

# Advanced FPGA Design Architecture, Implementation, and Optimization

**Steve Kilts**

*Spectrum Design Solutions  
Minneapolis, Minnesota*



The Institute of Electrical and Electronics Engineers, Inc., New York



**WILEY-INTERSCIENCE**  
**A JOHN WILEY & SONS, INC., PUBLICATION**

Copyright © 2007 by John Wiley & Sons, Inc. All rights reserved.

Published by John Wiley & Sons, Inc., Hoboken, New Jersey.  
Published simultaneously in Canada.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning, or otherwise, except as permitted under Section 107 or 108 of the 1976 United States Copyright Act, without either the prior written permission of the Publisher, or authorization through payment of the appropriate per-copy fee to the Copyright Clearance Center, Inc., 222 Rosewood Drive, Danvers, MA 01923, 978-750-8400, fax 978-646-8600, or on the web at [www.copyright.com](http://www.copyright.com).

Requests to the Publisher for permission should be addressed to the Permissions Department, John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030, (201) 748-6011, fax (201) 748-6008.

**Limit of Liability/Disclaimer of Warranty:** While the publisher and author have used their best efforts in preparing this book, they make no representations or warranties with respect to the accuracy or completeness of the contents of this book and specifically disclaim any implied warranties of merchantability or fitness for a particular purpose. No warranty may be created or extended by sales representatives or written sales materials. The advice and strategies contained herein may not be suitable for your situation. You should consult with a professional where appropriate. Neither the publisher nor author shall be liable for any loss of profit or any other commercial damages, including but not limited to special, incidental, consequential, or other damages.

For general information on our other products and services please contact our Customer Care Department within the U.S. at 877-762-2974, outside the U.S. at 317-572-3993 or fax 317-572-4002.

Wiley also publishes its books in a variety of electronic formats. Some content that appears in print, however, may not be available in electronic format.

***Library of Congress Cataloging-in-Publication Data***

Kilts, Steve, 1978-  
Advanced FPGA design: Architecture, Implementation, and Optimization/  
by Steve Kilts.  
p. cm.  
Includes index.  
ISBN 978-0-470-05437-6 (cloth)  
1. Field programmable gate arrays--Design and construction.  
I. Title.  
TK7895.G36K55 2007  
621.39'5--dc22  
2006033573

Printed in the United States of America

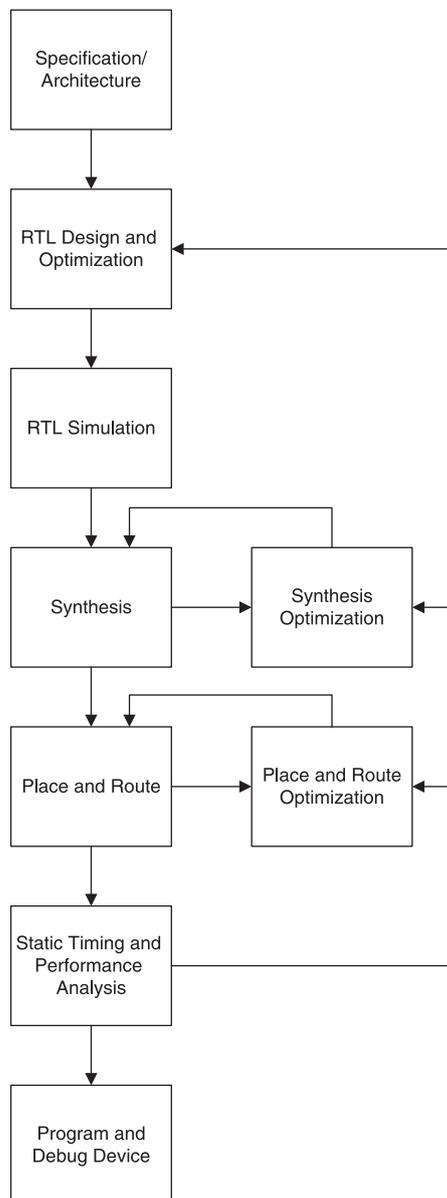
10 9 8 7 6 5 4 3 2 1

*To my wife, Teri, who felt that the  
subject matter was rather dry*

# Flowchart of Contents

---

- Chapter 1: Architecting Speed
- Chapter 2: Architecting Area
- Chapter 3: Architecting Power
- Chapter 4: Example Design: The Advanced Encryption Standard
- Chapter 5: High-Level Design
- Chapter 6: Clock Domains
- Chapter 7: Example Design: I2S versus SPDIF
- Chapter 8: Implementing Math Functions
- Chapter 9: Example Design: Floating-Point Unit
- Chapter 10: Reset Circuits
- Chapter 11: Advanced Simulation
- Chapter 12: Coding for Synthesis
- Chapter 13: Example Design: The Secure Hash Algorithm
- Chapter 14: Synthesis Optimization
- Chapter 15: Floorplanning
- Chapter 16: Place and Route Optimization
- Chapter 17: Example Design: Microprocessor
- Chapter 18: Static Timing Analysis
- Chapter 19: PCB Issues



# Contents

---

<b>Preface</b>	<b>xiii</b>
----------------	-------------

---

<b>Acknowledgments</b>	<b>xv</b>
------------------------	-----------

---

<b>1. Architecting Speed</b>	<b>1</b>
------------------------------	----------

---

1.1 High Throughput	2
1.2 Low Latency	4
1.3 Timing	6
1.3.1 Add Register Layers	6
1.3.2 Parallel Structures	8
1.3.3 Flatten Logic Structures	10
1.3.4 Register Balancing	12
1.3.5 Reorder Paths	14
1.4 Summary of Key Points	16

<b>2. Architecting Area</b>	<b>17</b>
-----------------------------	-----------

---

2.1 Rolling Up the Pipeline	18
2.2 Control-Based Logic Reuse	20
2.3 Resource Sharing	23
2.4 Impact of Reset on Area	25
2.4.1 Resources Without Reset	25
2.4.2 Resources Without Set	26
2.4.3 Resources Without Asynchronous Reset	27
2.4.4 Resetting RAM	29
2.4.5 Utilizing Set/Reset Flip-Flop Pins	31
2.5 Summary of Key Points	34

<b>3. Architecting Power</b>	<b>37</b>
------------------------------	-----------

---

3.1 Clock Control	38
3.1.1 Clock Skew	39
3.1.2 Managing Skew	40

**viii** Contents

- 3.2 Input Control 42
- 3.3 Reducing the Voltage Supply 44
- 3.4 Dual-Edge Triggered Flip-Flops 44
- 3.5 Modifying Terminations 45
- 3.6 Summary of Key Points 46

---

**4. Example Design: The Advanced Encryption Standard 47**

- 4.1 AES Architectures 47
  - 4.1.1 One Stage for Sub-bytes 51
  - 4.1.2 Zero Stages for Shift Rows 51
  - 4.1.3 Two Pipeline Stages for Mix-Column 52
  - 4.1.4 One Stage for Add Round Key 52
  - 4.1.5 Compact Architecture 53
  - 4.1.6 Partially Pipelined Architecture 57
  - 4.1.7 Fully Pipelined Architecture 60
- 4.2 Performance Versus Area 66
- 4.3 Other Optimizations 67

---

**5. High-Level Design 69**

- 5.1 Abstract Design Techniques 69
- 5.2 Graphical State Machines 70
- 5.3 DSP Design 75
- 5.4 Software/Hardware Codesign 80
- 5.5 Summary of Key Points 81

---

**6. Clock Domains 83**

- 6.1 Crossing Clock Domains 84
  - 6.1.1 Metastability 86
  - 6.1.2 Solution 1: Phase Control 88
  - 6.1.3 Solution 2: Double Flopping 89
  - 6.1.4 Solution 3: FIFO Structure 92
  - 6.1.5 Partitioning Synchronizer Blocks 97
- 6.2 Gated Clocks in ASIC Prototypes 97
  - 6.2.1 Clocks Module 98
  - 6.2.2 Gating Removal 99
- 6.3 Summary of Key Points 100

---

**7. Example Design: I2S Versus SPDIF 101**

- 7.1 I2S 101
  - 7.1.1 Protocol 102
  - 7.1.2 Hardware Architecture 102

7.1.3	Analysis	105
7.2	SPDIF	107
7.2.1	Protocol	107
7.2.2	Hardware Architecture	108
7.2.3	Analysis	114

---

## 8. Implementing Math Functions 117

8.1	Hardware Division	117
8.1.1	Multiply and Shift	118
8.1.2	Iterative Division	119
8.1.3	The Goldschmidt Method	120
8.2	Taylor and Maclaurin Series Expansion	122
8.3	The CORDIC Algorithm	124
8.4	Summary of Key Points	126

---

## 9. Example Design: Floating-Point Unit 127

9.1	Floating-Point Formats	127
9.2	Pipelined Architecture	128
9.2.1	Verilog Implementation	131
9.2.2	Resources and Performance	137

---

## 10. Reset Circuits 139

10.1	Asynchronous Versus Synchronous	140
10.1.1	Problems with Fully Asynchronous Resets	140
10.1.2	Fully Synchronized Resets	142
10.1.3	Asynchronous Assertion, Synchronous Deassertion	144
10.2	Mixing Reset Types	145
10.2.1	Nonresetable Flip-Flops	145
10.2.2	Internally Generated Resets	146
10.3	Multiple Clock Domains	148
10.4	Summary of Key Points	149

---

## 11. Advanced Simulation 151

11.1	Testbench Architecture	152
11.1.1	Testbench Components	152
11.1.2	Testbench Flow	153
11.1.2.1	Main Thread	153
11.1.2.2	Clocks and Resets	154
11.1.2.3	Test Cases	155

**x** Contents

11.2	System Stimulus	157
11.2.1	MATLAB	157
11.2.2	Bus-Functional Models	158
11.3	Code Coverage	159
11.4	Gate-Level Simulations	159
11.5	Toggle Coverage	162
11.6	Run-Time Traps	165
11.6.1	Timescale	165
11.6.2	Glitch Rejection	165
11.6.3	Combinatorial Delay Modeling	166
11.7	Summary of Key Points	169

---

**12. Coding for Synthesis** **171**

12.1	Decision Trees	172
12.1.1	Priority Versus Parallel	172
12.1.2	Full Conditions	176
12.1.3	Multiple Control Branches	179
12.2	Traps	180
12.2.1	Blocking Versus Nonblocking	180
12.2.2	For-Loops	183
12.2.3	Combinatorial Loops	185
12.2.4	Inferred Latches	187
12.3	Design Organization	188
12.3.1	Partitioning	188
12.3.1.1	Data Path Versus Control	188
12.3.1.2	Clock and Reset Structures	189
12.3.1.3	Multiple Instantiations	190
12.3.2	Parameterization	191
12.3.2.1	Definitions	191
12.3.2.2	Parameters	192
12.3.2.3	Parameters in Verilog-2001	194
12.4	Summary of Key Points	195

---

**13. Example Design: The Secure Hash Algorithm** **197**

13.1	SHA-1 Architecture	197
13.2	Implementation Results	204

---

**14. Synthesis Optimization** **205**

14.1	Speed Versus Area	206
14.2	Resource Sharing	208

14.3	Pipelining, Retiming, and Register Balancing	211
14.3.1	The Effect of Reset on Register Balancing	213
14.3.2	Resynchronization Registers	215
14.4	FSM Compilation	216
14.4.1	Removal of Unreachable States	219
14.5	Black Boxes	220
14.6	Physical Synthesis	223
14.6.1	Forward Annotation Versus Back-Annotation	224
14.6.2	Graph-Based Physical Synthesis	225
14.7	Summary of Key Points	226
<b>15. Floorplanning</b>		<b>229</b>
15.1	Design Partitioning	229
15.2	Critical-Path Floorplanning	232
15.3	Floorplanning Dangers	233
15.4	Optimal Floorplanning	234
15.4.1	Data Path	234
15.4.2	High Fan-Out	234
15.4.3	Device Structure	235
15.4.4	Reusability	238
15.5	Reducing Power Dissipation	238
15.6	Summary of Key Points	240
<b>16. Place and Route Optimization</b>		<b>241</b>
16.1	Optimal Constraints	241
16.2	Relationship between Placement and Routing	244
16.3	Logic Replication	246
16.4	Optimization across Hierarchy	247
16.5	I/O Registers	248
16.6	Pack Factor	250
16.7	Mapping Logic into RAM	251
16.8	Register Ordering	251
16.9	Placement Seed	252
16.10	Guided Place and Route	254
16.11	Summary of Key Points	254
<b>17. Example Design: Microprocessor</b>		<b>257</b>
17.1	SRC Architecture	257
17.2	Synthesis Optimizations	259
17.2.1	Speed Versus Area	260

**xii** Contents

17.2.2	Pipelining	261	
17.2.3	Physical Synthesis	262	
17.3	Floorplan Optimizations	262	
17.3.1	Partitioned Floorplan	263	
17.3.2	Critical-Path Floorplan: Abstraction 1	264	
17.3.3	Critical-Path Floorplan: Abstraction 2	265	
<b>18. Static Timing Analysis</b>			<b>269</b>
<hr/>			
18.1	Standard Analysis	269	
18.2	Latches	273	
18.3	Asynchronous Circuits	276	
18.3.1	Combinatorial Feedback	277	
18.4	Summary of Key Points	278	
<b>19. PCB Issues</b>			<b>279</b>
<hr/>			
19.1	Power Supply	279	
19.1.1	Supply Requirements	279	
19.1.2	Regulation	283	
19.2	Decoupling Capacitors	283	
19.2.1	Concept	283	
19.2.2	Calculating Values	285	
19.2.3	Capacitor Placement	286	
19.3	Summary of Key Points	288	
<b>Appendix A</b>			<b>289</b>
<hr/>			
<b>Appendix B</b>			<b>303</b>
<hr/>			
<b>Bibliography</b>			<b>319</b>
<hr/>			
<b>Index</b>			<b>321</b>
<hr/>			

# Preface

---

In the design-consulting business, I have been exposed to countless FPGA (Field Programmable Gate Array) designs, methodologies, and design techniques. Whether my client is on the Fortune 100 list or is just a start-up company, they will inevitably do some things right and many things wrong. After having been exposed to a wide variety of designs in a wide range of industries, I began developing my own arsenal of techniques and heuristics from the combined knowledge of these experiences. When mentoring new FPGA design engineers, I draw my suggestions and recommendations from this experience. Up until now, many of these recommendations have referenced specific white papers and application notes (appnotes) that discuss specific practical aspects of FPGA design. The purpose of this book is to condense years of experience spread across numerous companies and teams of engineers, as well as much of the wisdom gathered from technology-specific white papers and appnotes, into a single book that can be used to refine a designer's knowledge and aid in becoming an advanced FPGA designer.

There are a number of books on FPGA design, but few of these truly address advanced real-world topics in detail. This book attempts to cut out the fat of unnecessary theory, speculation on future technologies, and the details of outdated technologies. It is written in a terse, concise format that addresses the various topics without wasting the reader's time. Many sections in this book assume that certain fundamentals are understood, and for the sake of brevity, background information and/or theoretical frameworks are not always covered in detail. Instead, this book covers in-depth topics that have been encountered in real-world designs. In some ways, this book replaces a limited amount of industry experience and access to an experienced mentor and will hopefully prevent the reader from learning a few things the hard way. It is the advanced, practical approach that makes this book unique.

One thing to note about this book is that it will not flow from cover to cover like a novel. For a set of advanced topics that are not intrinsically tied to one another, this type of flow is impossible without blatantly filling it with fluff. Instead, to organize this book, I have ordered the chapters in such a way that they follow a typical design flow. The first chapters discuss architecture, then simulation, then synthesis, then floorplanning, and so on. This is illustrated in the Flowchart of Contents provided at the beginning of the book. To provide

accessibility for future reference, the chapters are listed side-by-side with the relevant block in the flow diagram.

The remaining chapters in this book are heavy with examples. For brevity, I have selected Verilog as the default HDL (Hardware Description Language). Xilinx as the default FPGA vendor, and Synplicity as the default synthesis and floorplanning tool. Most of the topics covered in this book can easily be mapped to VHDL, Altera, Mentor Graphics, and so forth, but to include all of these for completeness would only serve to cloud the important points. Even if the reader of this book uses these other technologies, this book will still deliver its value. If you have any feedback, good or bad, feel free to email me at [steve.kilts@spectrumdsi.com](mailto:steve.kilts@spectrumdsi.com)

STEVE KILTS

*Minneapolis, Minnesota  
March 2007*

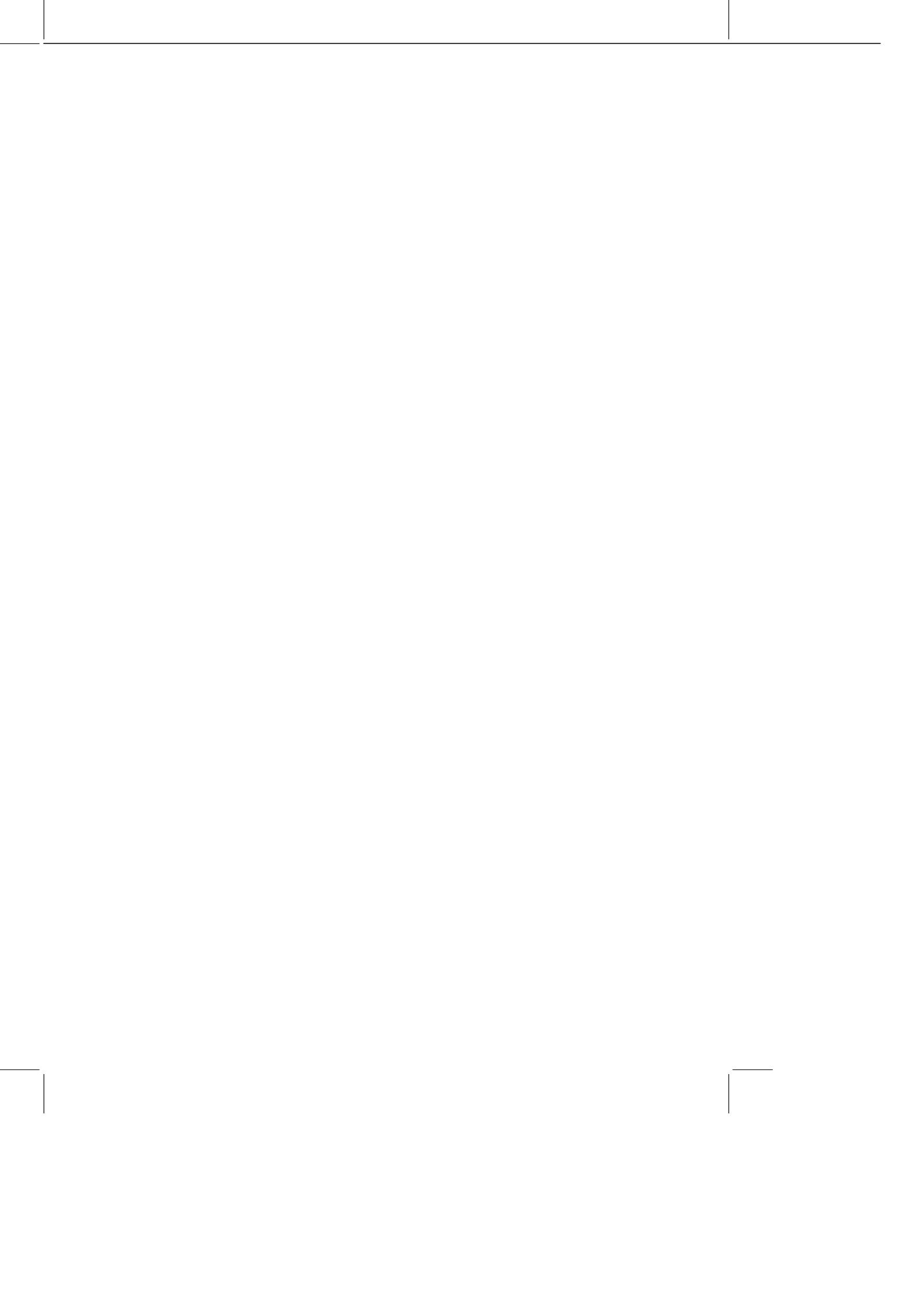
# Acknowledgments

---

During the course of my career, I have had the privilege to work with many excellent digital design engineers. My exposure to these talented engineers began at Medtronic and continued over the years through my work as a consultant for companies such as Honeywell, Guidant, Teradyne, Telex, Unisys, AMD, ADC, and a number of smaller/start-up companies involved with a wide variety of FPGA applications. I also owe much of my knowledge to the appnotes and white papers published by the major FPGA vendors. These resources contain invaluable real-world heuristics that are not included in a standard engineering curriculum.

Specific to this book, I owe a great deal to Xilinx and Synplicity, both of which provided the FPGA design tools used throughout the book, as well as a number of key reviewers. Reviewers of note also include Peter Calabrese of Synplicity, Cliff Cummins of Sunburst Design, Pete Danile of Synplicity, Anders Enggaard of Axcon, Mike Fette of Spectrum Design Solutions, Philip Freidin of Fliptronics, Paul Fuchs of NuHorizons, Don Hodapp of Xilinx, Ashok Kulkarni of Synplicity, Rod Landers of Spectrum Design Solutions, Ryan Link of Logic, Dave Matthews of Verein, Lance Roman of Roman-Jones, B. Joshua Rosen of Polybus, Gary Stevens of iSine, Jim Torgerson, and Larry Weegman of Xilinx.

S.K.



# Chapter 1

---

## Architecting Speed

**S**ophisticated tool optimizations are often not good enough to meet most design constraints if an arbitrary coding style is used. This chapter discusses the first of three primary physical characteristics of a digital design: speed. This chapter also discusses methods for architectural optimization in an FPGA.

There are three primary definitions of speed depending on the context of the problem: throughput, latency, and timing. In the context of processing data in an FPGA, throughput refers to the amount of data that is processed per clock cycle. A common metric for throughput is bits per second. Latency refers to the time between data input and processed data output. The typical metric for latency will be time or clock cycles. Timing refers to the logic delays between sequential elements. When we say a design does not “meet timing,” we mean that the delay of the critical path, that is, the largest delay between flip-flops (composed of combinatorial delay, clk-to-out delay, routing delay, setup timing, clock skew, and so on) is greater than the target clock period. The standard metrics for timing are clock period and frequency.

During the course of this chapter, we will discuss the following topics in detail:

- High-throughput architectures for maximizing the number of bits per second that can be processed by the design.
- Low-latency architectures for minimizing the delay from the input of a module to the output.
- Timing optimizations to reduce the combinatorial delay of the critical path.
  - Adding register layers to divide combinatorial logic structures.
  - Parallel structures for separating sequentially executed operations into parallel operations.
  - Flattening logic structures specific to priority encoded signals.
  - Register balancing to redistribute combinatorial logic around pipelined registers.
  - Reordering paths to divert operations in a critical path to a noncritical path.

## 1.1 HIGH THROUGHPUT

A high-throughput design is one that is concerned with the steady-state data rate but less concerned about the time any specific piece of data requires to propagate through the design (latency). The idea with a high-throughput design is the same idea Ford came up with to manufacture automobiles in great quantities: an assembly line. In the world of digital design where data is processed, we refer to this under a more abstract term: pipeline.

A pipelined design conceptually works very similar to an assembly line in that the raw material or data input enters the front end, is passed through various stages of manipulation and processing, and then exits as a finished product or data output. The beauty of a pipelined design is that new data can begin processing before the prior data has finished, much like cars are processed on an assembly line. Pipelines are used in nearly all very-high-performance devices, and the variety of specific architectures is unlimited. Examples include CPU instruction sets, network protocol stacks, encryption engines, and so on.

From an algorithmic perspective, an important concept in a pipelined design is that of “unrolling the loop.” As an example, consider the following piece of code that would most likely be used in a software implementation for finding the third power of X. Note that the term “software” here refers to code that is targeted at a set of procedural instructions that will be executed on a microprocessor.

```
XPower = 1;
for (i=0; i < 3; i++)
    XPower = X * XPower;
```

Note that the above code is an iterative algorithm. The same variables and addresses are accessed until the computation is complete. There is no use for parallelism because a microprocessor only executes one instruction at a time (for the purpose of argument, just consider a single core processor). A similar implementation can be created in hardware. Consider the following Verilog implementation of the same algorithm (output scaling not considered):

```
module power3(
    output [7:0] XPower,
    output      finished,
    input  [7:0] X,
    input      clk, start); // the duration of start is a
                            // single clock

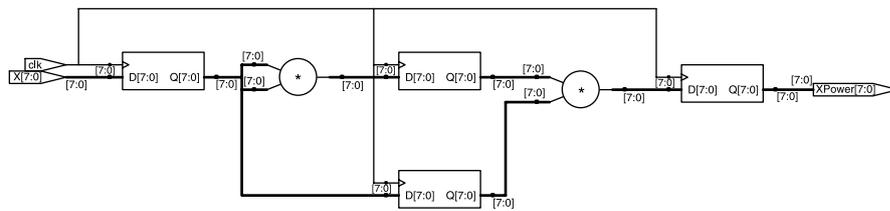
    reg [7:0] ncount;
    reg [7:0] XPower;

    assign finished = (ncount == 0);

    always@(posedge clk)
        if(start) begin
            XPower <= X;
            ncount <= 2;
        end
end
```



## 4 Chapter 1 Architecting Speed



**Figure 1.2** Pipelined implementation.

In the above implementation, the value of  $X$  is passed to both pipeline stages where independent resources compute the corresponding multiply operation. Note that while  $X$  is being used to calculate the final power of 3 in the second pipeline stage, the next value of  $X$  can be sent to the first pipeline stage as shown in Figure 1.2.

Both the final calculation of  $X^3$  (XPower3 resources) and the first calculation of the next value of  $X$  (XPower2 resources) occur simultaneously. The performance of this design is

Throughput =  $8/1$ , or 8 bits/clock

Latency = 3 clocks

Timing = One multiplier delay in the critical path

The throughput performance increased by a factor of 3 over the iterative implementation. In general, if an algorithm requiring  $n$  iterative loops is “unrolled,” the pipelined implementation will exhibit a throughput performance increase of a factor of  $n$ . There was no penalty in terms of latency as the pipelined implementation still required 3 clocks to propagate the final computation. Likewise, there was no timing penalty as the critical path still contained only one multiplier.

Unrolling an iterative loop increases throughput.

The penalty to pay for unrolling loops such as this is an increase in area. The iterative implementation required a single register and multiplier (along with some control logic not shown in the diagram), whereas the pipelined implementation required a separate register for both  $X$  and XPower and a separate multiplier for every pipeline stage. Optimizations for area are discussed in the Chapter 2.

The penalty for unrolling an iterative loop is a proportional increase in area.

### 1.2 LOW LATENCY

A low-latency design is one that passes the data from the input to the output as quickly as possible by minimizing the intermediate processing delays. Oftentimes, a low-latency design will require parallelisms, removal of pipelining, and logical short cuts that may reduce the throughput or the max clock speed in a design.

Referring back to our power-of-3 example, there is no obvious latency optimization to be made to the iterative implementation as each successive multiply operation must be registered for the next operation. The pipelined implementation, however, has a clear path to reducing latency. Note that at each pipeline stage, the product of each multiply must wait until the next clock edge before it is propagated to the next stage. By removing the pipeline registers, we can minimize the input to output timing:

```

module power3(
  output [7:0] XPower,
  input  [7:0] X
);
  reg [7:0] XPower1, XPower2;
  reg [7:0] X1, X2;

  assign XPower = XPower2 * X2;

  always @* begin
    X1 = X;
    XPower1 = X;
  end

  always @* begin
    X2 = X1;
    XPower2 = XPower1*X1;
  end
endmodule

```

In the above example, the registers were stripped out of the pipeline. Each stage is a combinatorial expression of the previous as shown in Figure 1.3.

The performance of this design is

Throughput = 8 bits/clock (assuming one new input per clock)

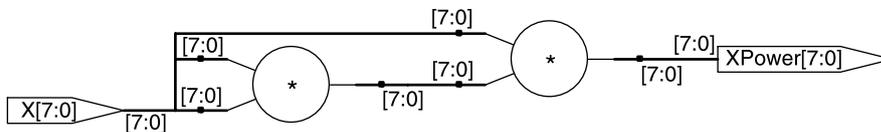
Latency = Between one and two multiplier delays, 0 clocks

Timing = Two multiplier delays in the critical path

By removing the pipeline registers, we have reduced the latency of this design below a single clock cycle.

Latency can be reduced by removing pipeline registers.

The penalty is clearly in the timing. Previous implementations could theoretically run the system clock period close to the delay of a single multiplier, but in the



**Figure 1.3** Low-latency implementation.

- [click Refugee: Force Heretic II \(Star Wars: The New Jedi Order, Book 16\)](#)
- [Practice Makes Perfect Beginning Spanish pdf, azw \(kindle\), epub, doc, mobi](#)
- [download online Shame and Wonder: Essays](#)
- [The Language of Threads: A Novel pdf, azw \(kindle\), epub, doc, mobi](#)
- [click Concise Book of Muscles online](#)
  
- <http://omarnajmi.com/library/Wittgenstein--Empiricism--and-Language.pdf>
- <http://fitnessfatale.com/freebooks/Practice-Makes-Perfect-Beginning-Spanish.pdf>
- <http://thewun.org/?library/Shame-and-Wonder--Essays.pdf>
- <http://drmurphreesnewsletters.com/library/The-Language-of-Threads--A-Novel.pdf>
- <http://sidenoter.com/?ebooks/Concise-Book-of-Muscles.pdf>