
ASSERTION-BASED DESIGN

SECOND EDITION

ASSERTION-BASED DESIGN

SECOND EDITION

Harry Foster

Jasper Design Automation, Inc.

Adam Krolnik

LSI Logic Corporation.

David Lacey

Hewlett-Packard Company

KLUWER ACADEMIC PUBLISHERS

NEW YORK, BOSTON, DORDRECHT, LONDON, MOSCOW

eBook ISBN: 1-4020-8028-X
Print ISBN: 1-4020-8027-1

©2005 Springer Science + Business Media, Inc.

Print ©2004 Kluwer Academic Publishers
Boston

All rights reserved

No part of this eBook may be reproduced or transmitted in any form or by any means, electronic, mechanical, recording, or otherwise, without written consent from the Publisher

Created in the United States of America

Visit Springer's eBookstore at:
and the Springer Global Website Online at:

<http://ebooks.kluweronline.com>
<http://www.springeronline.com>

Dedicated to:

*Jeanne—the most wonderful person in my life.
And to my children—Elliott, Lance, and Hannah.
Always remember, when I was your age I used to
have to walk across the room to change the channel.*

-Harry

Cindy, Seth, Nicholas, Sarah and Jesus the Christ.

-Adam

*To my loving wife, Deborah, for her patience and
support while this book was written.*

-David

TABLE OF CONTENTS

Chapter 1 Introduction	1
1.1 Property checking	1
1.2 Verification techniques	2
1.3 What is an assertion?	3
1.3.1 A historical perspective	4
1.3.2 Do assertions really work?	6
1.3.3 What are the benefits of assertions?	7
1.3.4 Why are assertions not used?	11
1.4 Phases of the design process	14
1.4.1 Ensuring requirements are satisfied	16
1.4.2 Techniques for ensuring consistency	18
1.4.3 Roles and ownership	19
1.5 Summary	20
Chapter 2 Assertion Methodology	21
2.1 Design methodology	21
2.1.1 Project planning	22
2.1.2 Design requirements	27
2.1.3 Design documents	28
2.1.4 Design reviews	29
2.1.5 Design validation	30
2.2 Assertion methodology for new designs	30
2.2.1 Key learnings	31
2.2.2 Best practices	33
2.2.3 Assertion density	37
2.2.4 Process for adding assertions	39
2.2.5 When not to add assertions	39
2.3 Assertion methodology for existing designs	40
2.4 Assertions and simulation	42
2.5 Assertions and formal verification	44
2.5.1 Formal verification framework	44
2.5.2 Formal methodology	48
2.5.3 ECC example	53
2.5.4 Gradual exhaustive formal verification	56

2.6 Summary	59
Chapter 3 Specifying RTL Properties	61
3.1 Definitions and concepts	62
3.1.1 Property	62
3.1.2 Events	65
3.2 Property classification	65
3.2.1 Safety versus liveness	66
3.2.2 Constraint versus assertion	67
3.2.3 Declarative versus procedural	67
3.3 RTL assertion specification techniques	68
3.3.1 RTL invariant assertions	69
3.3.2 Declaring properties with PSL	72
3.3.3 RTL cycle related assertions	73
3.3.4 PSL and default clock declaration	74
3.3.5 Specifying sequences	75
3.3.6 Specifying eventualities	80
3.3.7 PSL built-in functions	82
3.4 Pragma-based assertions	82
3.5 SystemVerilog assertions	84
3.5.1 Immediate assertions	84
3.5.2 Concurrent assertions	86
3.5.3 System functions	95
3.6 PCI property specification example	96
3.6.1 PCI overview	96
3.7 Summary	102
Chapter 4 PLI-Based Assertions	103
4.1 Procedural assertions	104
4.1.1 A simple PLI assertion	105
4.1.2 Assertions within a simulation time slot	108
4.1.3 Assertions across simulation time slots	111
4.1.4 False firing across multiple time slots	116
4.2 PLI-based assertion library	118
4.2.1 Assert quiescent state	119
4.3 Summary	123
Chapter 5 Functional Coverage	125
5.1 Verification approaches	126
5.2 Understanding coverage	127
5.2.1 Controllability versus observability	128
5.2.2 Types of traditional coverage metrics	128
5.2.3 What is functional coverage?	130
5.2.4 Building functional coverage models	132
5.2.5 Sources of functional coverage	133

5.3 Does functional coverage really work?	134
5.3.1 Benefits of functional coverage	135
5.3.2 Success stories	135
5.3.3 Why is functional coverage not used	136
5.4 Functional coverage methodology	137
5.4.1 Steps to functional coverage	138
5.4.2 Correct coverage density	139
5.4.3 Incorrect coverage density	141
5.4.4 Coverage analysis	142
5.4.5 Coverage best practices	145
5.4.6 Coverage-driven test generation	149
5.5 Specifying functional coverage	150
5.5.1 Embedded in the RTL	150
5.5.2 Functional coverage libraries	151
5.5.3 Assertion-based methods	152
5.5.4 Post processing	154
5.5.5 PLI logging and reporting	154
5.5.6 Simulation control	155
5.6 Functional coverage examples	156
5.7 AHB example	158
5.8 Summary	160

Chapter 6 Assertion Patterns 161

6.1 Introduction to patterns	161
6.1.1 What are assertion patterns?	162
6.1.2 Elements of an assertion pattern	163
6.2 Signal patterns	164
6.2.1 X detection pattern	164
6.2.2 Valid range pattern	167
6.2.3 One-hot pattern	169
6.2.4 Gray-code pattern	172
6.3 Set patterns	173
6.3.1 Valid opcode pattern	173
6.3.2 Valid signal combination pattern	175
6.3.3 Invalid signal combination pattern	177
6.4 Conditional patterns	179
6.4.1 Conditional expression pattern	179
6.4.2 Sequence implication pattern	181
6.5 Past and future event patterns	185
6.5.1 Past event pattern	185
6.5.2 Future event pattern	187
6.6 Window patterns	189
6.6.1 Time-bounded window patterns	189
6.6.2 Event-bounded window patterns	192
6.7 Sequence patterns	194
6.7.1 Forbidden sequence patterns	194
6.7.2 Buffered data validity pattern	195

6.7.3 Tagged transaction pattern	196
6.7.4 Pipelined protocol pattern	199
6.8 Applying patterns to a real example	202
6.8.1 Intra-interface assertions	204
6.8.2 Inter-interface assertions	208
6.9 Summary	210
Chapter 7 Assertion Cookbook	211
7.1 Queue—FIFO	213
7.2 Fixed depth pipeline register	219
7.3 Stack—LIFO	222
7.4 Caches—direct mapped	225
7.5 Cache—set associative	231
7.6 FSM	236
7.7 Counters	240
7.8 Multiplexers	244
7.8.1 Encoded multiplexer	244
7.8.2 Decoded one-hot multiplexer	245
7.8.3 Priority multiplexer	246
7.8.4 Complex multiplexer	248
7.9 Encoder	249
7.10 Priority encoder	251
7.11 Simple single request protocol	252
7.12 In-order multiple request protocol	254
7.13 Out-of-order request protocol	257
7.14 Memories	259
7.15 Arbiter	262
7.16 Summary	266
Chapter 8 Specifying Correct Behavior	267
8.1 Natural language interpretation	267
8.1.1 Temporal ambiguity	268
8.1.2 Active ambiguity	271
8.1.3 Boundary ambiguity	273
8.1.4 Too strong interpretation	274
8.1.5 Implicit assumption	276
8.1.6 Partial specification	277
8.2 Property specification guidelines	278
8.2.2 Syntax ambiguity	280
8.3 Clarity in higher-level specification	281
8.3.1 Implementation assertions	283
8.3.2 Higher-level requirements	285
8.3.3 Modeling high-level requirements	287
8.4 Summary	288

Appendix A Open Verification Library	291
A.1 OVL methodology advantages	291
A.2 OVL standard definition	292
A.2.1 OVL runtime macro controls	293
A.2.2 Customizing OVL messages	294
A.3 Firing OVL monitors	296
A.4 Using OVL assertion monitors	297
A.5 Checking invariant properties	298
A.5.1 assert_always	298
A.5.2 assert_never	300
A.5.3 assert_zero_one_hot	302
A.5.4 assert_range	303
A.6 Checking cycle relationships	305
A.6.1 assert_next	305
A.6.2 assert_frame	307
A.6.3 assert_cycle_sequence	309
A.7 Checking event bounded windows	311
A.7.1 assert_win_change	311
A.7.2 assert_win_unchange	313
A.8 Checking time bounded windows	314
A.8.1 assert_change	315
A.8.2 assert_unchange	316
A.9 Checking state transitions	318
A.9.1 assert_no_transition	318
A.9.2 assert_transition	319
 Appendix B PSL Property Specification Language	 321
B.1 Introduction to PSL	321
B.2 Operators and keywords	322
B.3 PSL Boolean layer	323
B.4 PSL Temporal Layer	324
B.4.1 SERE	324
B.4.2 Sequence	325
B.4.3 Braced SERE	325
B.4.4 SERE concatenation (;) operator	325
B.4.5 Consecutive repetition ([*]) operator	325
B.4.6 Nonconsecutive repetition ([=]) operator	327
B.4.7 Goto repetition ([->]) operator	328
B.4.8 Sequence fusion (:) operator	329
B.4.9 Sequence non-length-matching (&) operator	329
B.4.10 Sequence length-matching (&&) operator	329
B.4.11 Sequence or () operator	329
B.4.12 until* sequence operators	330
B.4.13 within sequence operators	330
B.4.14 next operator	330
B.4.15 eventually! operator	331
B.4.16 before* operators	331

B.4.17 abort operator	332
B.4.18 Endpoint declaration	332
B.4.19 Suffix implication operators	332
B.4.20 Logical implication operator	333
B.4.21 always temporal operator	333
B.4.22 never temporal operator	334
B.5 PSL properties	334
B.5.1 Property declaration	334
B.5.2 Named properties	334
B.5.3 Property clocking	334
B.5.4 forall property replication	335
B.6 The verification layer	336
B.6.1 assert directive	336
B.6.2 assume directive	336
B.6.3 cover directive	336
B.7 The modeling layer	337
B.7.1 prev()	337
B.7.2 next()	337
B.7.3 stable()	338
B.7.4 rose()	338
B.7.5 fell()	339
B.7.6 isunknown()	339
B.7.7 countones()	339
B.7.8 onehot(), onehot0()	340
B.8 BNF	340

Appendix C SystemVerilog Assertions 353

C.1 Introduction to SystemVerilog	353
C.2 Operator and keywords	353
C.3 Sequence and property operations	355
C.3.1 Temporal delay	356
C.3.2 Consecutive repetition	357
C.3.3 Goto repetition	357
C.3.4 Nonconsecutive repetition	358
C.3.5 Sequence and Property AND	359
C.3.6 Sequence intersection	360
C.3.7 Sequence and Property OR	360
C.3.8 Boolean until (throughout)	361
C.3.9 Within sequence	362
C.3.10 Ended	362
C.3.11 Matched	363
C.3.12 First match	363
C.3.13 Property Implication	364
C.3.14 Conditional property selection	365
C.4 Property declarations	366
C.4.1 Sequence composition	368

C.5 Assert, Assume and Cover statements.	369
C.6 Dynamic data within sequences	370
C.7 System Functions	371
C.7.1 New operators	372
C.8 SystemTasks	373
C.9 BNF	374

FOREWORD

There is much excitement in the design and verification community about assertion-based design. The question is, who should study assertion-based design? The emphatic answer is, both design and verification engineers.

What may be unintuitive to many design engineers is that adding assertions to RTL code will actually reduce design time, while better documenting design intent.

Every design engineer should read this book! Design engineers that add assertions to their design will not only reduce the time needed to complete a design, they will also reduce the number of interruptions from verification engineers to answer questions about design intent and to address verification suite mistakes. With design assertions in place, the majority of the interruptions from verification engineers will be related to actual design problems and the error feedback provided will be more useful to help identify design flaws. A design engineer who does not add assertions to the RTL code will spend more time with verification engineers explaining the design functionality and intended interface requirements, knowledge that is needed by the verification engineer to complete the job of testing the design.

Every verification engineer should read this book! The smart verification engineer will assist the design engineer to add assertions to the RTL-design code because the sooner a design engineer understands the usage and benefits of inserting assertions into the design, the more valuable that design engineer will be to the verification effort. A smart verification engineer is someone who can help a designer to catch the vision and understand the ease and value of assertion-based design. This is the first book to comprehensively address and explain HDL assertion-based design.

My colleague Harry Foster is the best-known name in the Verilog verification and assertion-based methodology community. Along with Lionel Bening, Harry pioneered the Verilog Open Verification Library (OVL), a freely available set of verification-focused Verilog modules that have been used in advanced design and verification environments ever since they were introduced.

My colleague Adam Krolnik was the verification champion of the Verilog-2001 Standards Group. I counted on Adam to promote

and propose verification enhancements to the IEEE Verilog language.

David Lacey, Harry and Adam are key participants on the Accellera SystemVerilog Standards Group. Their practical verification experience has contributed to the value of the assertion enhancements added to the SystemVerilog standard.

These three verification specialists have written a book that will endow the reader with an understanding of the fundamental and important topics needed to comprehend and implement assertion-based design.

Included in Chapter 7 of this book is a valuable set of commonly used assertion examples to help the reader become familiar with the capabilities of assertion-based design. This book is a must for all design and verification engineers.

Clifford E. Cummings

Verilog Guru & President, Sunburst Design, Inc.

Member IEEE 1364-1995 Verilog Standards Group

Member IEEE 1364-2001 Verilog Standards Group

Member IEEE 1364-2002 Verilog RTL Synthesis Standards Group

Member Accellera SystemVerilog 3.0 Standards Group

Member Accellera SystemVerilog 3.1 Standards Group

PREFACE

You may have heard that this is a book about verification and now you're wondering why it's called *Assertion-Based Design*, and not *Assertion-Based Verification*. The answer to that is one of the driving forces in this book: Verification doesn't happen in a vacuum. Specification has to occur before any form of verification, and as you know, specification occurs very early in the design cycle. Thus, our contention is that assertion specification is one of the integral pieces of a contemporary design cycle.

Within that context then, the focus of this book is three-fold:

- How to specify assertions
- How to create and adopt a methodology that supports assertion-based design (predominately for RTL design)
- What to do with the assertions and methodology once you have them

To support these three over-arching goals, we showcase multiple forms of assertion specification: Accellera Open Verification Library (OVL), Accellera Property Specification Language (PSL), and Accellera SystemVerilog.

The recommendations and claims we make in this book are based on our combined actual experiences in applying an assertion-based methodology to real design and verification as well as our work in developing industry assertion standards.

Real-world experience. In *Assertion-Based Design*, we have pooled our combined experiences to share our understanding and provide a reality-based picture of our chosen topic. The following is a summary of our background related to this topic:

- Harry Foster—Chairs the Accellera Formal Verification Technical Committee; which is developing the PSL standard; created the Open Verification Library; member of the SystemVerilog Assertion Committee; and previously developed assertion-based methodologies at Hewlett-Packard Company.

-
- Adam Krolnik—Accellera SystemVerilog Assertion Committee member and major contributor in the development of the SystemVerilog assertion constructs; created assertion-based methodologies at Cyrix and LSI Logic Corporation.
 - David Lacey—Chairs the Accellera Open Verification Library committee; member of the SystemVerilog Assertion Committee; created functional coverage and assertion-based methodologies at Hewlett-Packard Company.

Fundamentals. Property specification is fundamental to an assertion-based verification platform (that is, assertions, constraints, and functional coverage). Once specified, properties enable the following components, which may be included in your assertion-based verification platform:

- *verifiable testplans* through property specification (for example, executable *functional coverage models*, which help answer the question “*what functionality has not been exercised?*”)
- *exhaustive* and *semi-exhaustive* formal property checking technology (for example, model checking and bounded-model checking)
- *dynamic property checking* technology (for example, monitoring assertions in simulation) for improved observability to reduce the time involved in debug
- *hardware verification languages* (HVLs) for testbench generation that leverage property specification to define expected input (constraints) and output (assertions) behavior
- *constraint-driven stimulus generation* based on interface properties targeting block-level designs
- *assertion property synthesis* to address silicon observability challenges during chip bring-up in the lab, as well as operational error detection required for high availability (HA) class systems

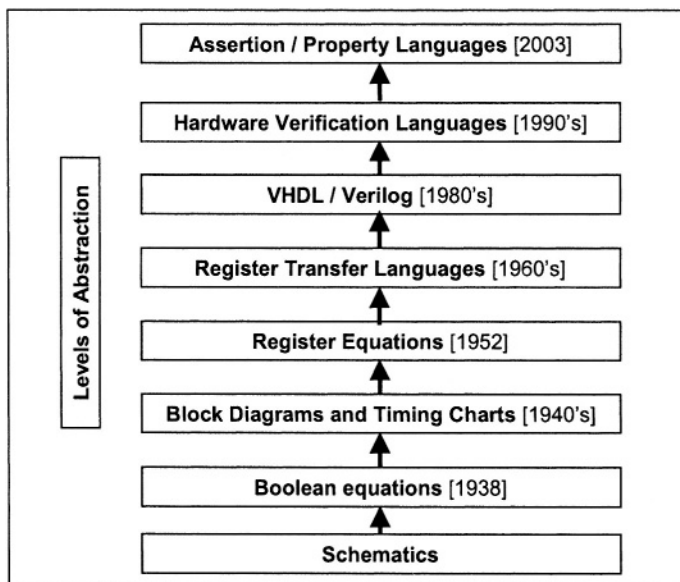
In this book, we discuss the important role that property specification plays in an assertion-based verification flow.

Evolution in levels of abstraction. The following figure shows an evolution in levels of design notation and specification abstraction. Each time we move up a level of abstraction, we expand possibilities, increase productivity, and improve communication of design intent. Perhaps most importantly, is the growth our field has experienced in conceptualizing new forms of specification, developing new technologies based on these new forms of specification, and then developing standards, which in turn opens new markets. For example, the development of Register Transfer Languages in the mid-1960’s led to the development of synthesis. However, it was the standardization of

VHDL/Verilog in the early 1990's that opened new markets and helped drive synthesis adoption.

The way design and verification has traditionally been performed is changing. In the future, we predict that design and verification will become property-based. Through the standardization of assertion and property languages that are occurring at the time of this publication, we foresee new and exciting EDA markets emerge, once again opening the door for improved productivity. All made possible through assertion-based design practices.

Design notation and specification levels of abstraction



Book organization

Assertion-Based Design is organized into chapters that can be read sequentially for a full understanding of our topic. Or you may wish to focus on a particular area of interest and visit the remaining chapters to supplement your understanding. We have allowed repetitions to support the readers who prefer to browse for information as well as readers who intend to read cover-to-cover and then revisit sections of particular personal relevance. Highlights of the contents follow.

Chapter 1, "Introduction"

Chapter 1 introduces property checking and modern verification techniques. It then introduces assertions and their use in industry, and statistics of their effectiveness. It discusses the benefits of assertion-based verification methodologies and dispels common misconceptions about assertion use within an RTL design flow.

Chapter 2, “Assertion Methodology”	Chapter 2 discusses details for effectively creating and managing an assertion-based methodology, which includes validating assertions using simulation and formal verification. It shows how to best apply assertion-based methodologies to both new and existing designs. It explores how an assertion-based methodology works together with, not separate from, a general design methodology. Finally, it introduces several key learnings that must be accepted by the project team in order to be successful with an assertion-based methodology.
Chapter 3, “Specifying RTL Properties”	Chapter 3 discusses basic property and assertion specification concepts and definitions. It introduces various forms of RTL assertion specifications, which include: the Accellera Open Verification Library (OVL), Accellera PSL formal property language, and Accellera SystemVerilog assertion constructs. Examples are given throughout the chapter to demonstrate how each property concept is used in concert with one of the assertion specification forms to create assertions. Many of these assertion examples can be used today, as they utilize the OVL. Other examples can be used as the EDA vendors move to support PSL and SystemVerilog. Finally, the chapter closes with a PCI property specification example.
Chapter 4, “PLI-Based Assertions”	Chapter 4 discusses PLI-based assertion and procedural assertion techniques. It provides actual code that can be used as an assertion specification form with today's Verilog simulators that support the PLI. It also addresses ways to prevent false firing of procedural assertions.
Chapter 5, “Functional Coverage”	Chapter 5 introduces the verification coverage models and how this topic fits in a book about assertions. It discusses the concepts of black-box and white-box testing, as well as assertion techniques that improve functional coverage. It explores the ideas of controllability and observability and their role in verification. The chapter focuses primarily on functional coverage models, but introduces other traditional coverage metrics (such as programming code coverage) and their role in the overall coverage efforts. Real world examples of successes using functional coverage is provides in this chapter as well as a discussion of the benefits of using functional coverage in your verification efforts. It provides an outline for an effective functional coverage methodology, including guidelines for achieving correct functional coverage density within your design. Finally, it provides a description of several forms of functional coverage specification before concluding with examples of both low-level RTL implementation and high-level architectural functional coverage specification.
Chapter 6, “Assertion Patterns”	Chapter 6 introduces the concept of assertion patterns as a convenient method to document and communicate commonly occurring assertions that are found in today's RTL designs. This provides an easy reference for broad classes of assertions that

allows the reader to more easily apply the concepts to their specific designs. A number of different assertion patterns are presented, including signal, set, conditional, past and future event, window, and sequence. Each assertion pattern is described in great detail and provides examples of applicable assertions and descriptive waveforms for additional clarity, as needed.

Chapter 7,
“Assertion
Cookbook”

Chapter 7 provides concrete examples of the common assertions and functional coverage points for components, interfaces, and general logic, including queues, stacks, finite state machines, encoders, decoders, multiplexers, state table structures, memory, and arbiters. It includes examples that use each form described in Chapter 2 (OVL, SystemVerilog, and PSL). While the list of components is not exhaustive, the chapter gives a broad coverage of components that enables readers to extend the concepts to their specific applications.

Chapter 8,
“Specifying
Correct
Behavior”

Chapter 8 present a set of clarifying ideas that we have found effective when attempting to specify various aspects of a design, at multiple levels of abstraction. We first present a set of common ambiguities that arise when interpreting a natural language specification. We then discuss the limitation of temporal property languages, and the need for additional modeling to overcome these limitations when attempting to specify higher level requirements. Finally, we conclude by presenting a construction guide we have found useful in our own work with assertion and functional coverage specification.

Appendix A,
“Open
Verification
Library”

This appendix provides a detailed discussion of the most commonly used monitors in the Open Verification Library.

Appendix B,
“PSL Property
Specification
Language”

This appendix provides a detailed discussion of the most commonly used keywords and operators in the PSL property specification language.

Appendix C,
“SystemVerilog
Assertions”

This appendix provides a detailed discussion of the most commonly used keywords and operators in the proposed SystemVerilog standard.

New in Second Edition

Differences between the first edition and the second edition include:

- Updates to the manuscript based on newer versions of standards
- Corrections to errata identified during reviewer feedback
- New material that presents techniques on how to avoid common ambiguity errors
- New material that discusses high-level requirements modeling for specification

Since the first edition was published, subtle changes occurred in a few of the standards we originally presented. In this edition, we have updated the manuscript to be in line with the Accellera SystemVerilog 3.1a and the Accellera PSL 1.1 proposed standards.

In addition, we have compiled feedback from multiple reviewers, and made the appropriate corrections. And we simplified the coding of a number of the examples and fixed known errors.

Finally, we added new material, Chapter 8, “Specifying Correct Behavior”. This chapter presents a set of clarifying ideas (that is, tips) about a set of common ambiguities that arise when interpreting a natural language specification and suggest techniques to avoid these common errors. In addition, this chapter discusses the limitations of temporal property languages and the need for additional modeling to overcome these limitations when attempting to specify higher-level requirements.

Acknowledgements

The authors wish to thank the following people who participated in discussions, made suggestions and other contributions to our *Assertion-Based Design* project:

Salim Ahmed, Johan Alfredsson, Tom Anderson, Yann Antonioli, Roy Armoni, Brian Bailey, Lionel Bening, Janick Bergeron, Dino Caporossi, Michael Chang, KC Chen, Carina Chiang, Ashwini Choudhary, Edmond Clarke, Claudionor Coelho, Ben Cohen, Cliff Cummings, Bernard Deadman, Kashyap Doerah, Surrendra Dudani, Cindy Eisner, Jeffrey Elbert, E. Allen Emerson, Limor Fix, Tom Fitzpatrick, Dana Fisman, Peter Flake, Jeanne Foster, Gary Gostin, Faisal Haque, John Havlicek, Bert Hill, Richard Ho,

Ramin Hojati, Alan Hu, Alan Hunter, C. Norris Ip, Tony Jones, Yaron Kashai, Kathryn Kranen, Avner Landver, James Lee, Amir Lehavot, Andy Lin, Lawrence Loh, Joseph Lu, Adriana Maggiore, Erich Marschner, Johan Mårtensson, David Matt, Anthony McIsaac, Steve Meier, Hillel Miller, Prakash Narain, Avigail Orni, Doug Perry, Gary Pimentel, Carl Pixley, Andy Piziali, David Price, Jeff Quigley, Bahman Rabii, Rajeev Ranjan, Joe Richards, Sitvanit Ruah, Vigyan Singhal, Sean Smith, Michal Siwinski, Sandeep Shukla, Bassam Tabbara, Sean Torsney, Andy Tsay, Mike Turpin, David Van Campenhout, Gal Vardi, Moshe Vardi, Paul Vogel, Tony Wilcox, Yaron Wolfsthal, Howard Wong-Toi.

Special thanks to Cliff Cummings, Lionel Bening, Avner Landver, Erich Marschner, Gary Pimentel, Andy Piziali, Joe Richards, Paul Vogel, Tony Wilcox.

Finally, a very special thanks to Jeanne Foster for providing high quality editing advice and services throughout this project.