

Book Reviews

Was it worth the wait? Yes!

Brian Bailey

Brian Bailey Consulting

■ **A BOOK** that follows from the *Reuse Methodology Manual (RMM)* by Michael Keating and Pierre Bricaud (Springer, 2002) has a lot to live up to. Does the *Verification Methodology Manual for SystemVerilog (VMM)* succeed? We should first look at the similarities and differences between the two situations. With the *RMM*, its need was clear: Engineers involved with design reuse had tried many things, and the *RMM* solidified those best practices into one book. It brought order to the chaos of design reuse at RTL. With the *VMM*, its need is equally clear, but the book deals with methodologies that are still somewhat experimental and still evolving, and a language to implement them—SystemVerilog—which is barely out of the starting gate, having been standardized by the IEEE just last year.

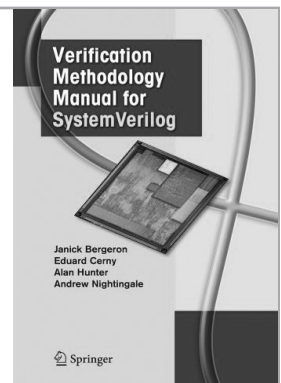
But that's not to say there isn't a wealth of experience between the covers of this book. It draws on the talents of Janick Bergeron of Synopsys, who wrote one of the first books in the functional verification field to have impact on the practices many companies use today. The *VMM*'s other three authors, Eduard Cerny (also with Synopsys), and Alan Hunter and Andrew Nightingale of ARM, bring to the book their own years of experience in solving these problems.

Furthermore, the methodology outlined in this book is not the first advanced verification methodology to have been put together. Synopsys had one—the Reference Verification Methodology (RVM), based on the Vera verification language; and Verisity, now Cadence, had their e Reuse Methodology (eRM). However, this book specifically talks about how to put together a verification methodology using the capabilities of SystemVerilog with the aim of showing users how to create both well-structured and reusable testbench components.

So who should read this book? It is not an introductory text for verification methodologies, assertions, or

Reviewed in this issue

Verification Methodology Manual for SystemVerilog, by Janick Bergeron et al. (Springer, 2005, ISBN 0-387-25538-9, 510 pp., \$129).



the SystemVerilog language. In fact, it assumes you are already an expert in all of the underlying techniques. It is not a standard because it has not been through an industry review process, and the object classes that it draws upon are not freely available to the public. It is not purely a marketing book—it really does provide significant technical value to the reader who wants to create reusable testbenches (although it's very clear that marketing had a hand in its creation, and, unfortunately, the book draws on some specific features of the Synopsys tool implementation). It is also not a book that explains why the rules are set; it just expects you to accept its teaching. Although some or all of these traits might be a negative for some people, they should try to look beyond these issues to extract the piles of good stuff the book contains.

I was surprised at the results I got from a straw poll I conducted among other verification luminaries and companies involved in verification IP development. I expected the respondents to have been eagerly awaiting the book's publication, which people had long anticipated since an announcement in early 2004 that it would be ready by the time of the 2004 Design Automation Conference. Instead, I found very few who had any plans to use it, and most had not even read it

yet. This might be due to the newness of SystemVerilog.

The book comprises numerous chapters that each deal with a particular aspect of a verification environment, such as assertions, testbench infrastructure, and coverage. Some later chapters also deal with constrained environments, such as those for formal verification and hardware-assisted verification. The last quarter of the book describes the class library that forms the methodology's underpinning.

Chapter 1, the introduction, is a bit of a disaster. It is badly written, flighty in nature, contains numerous errors, and has many poorly worded definitions. If you are a person who tends to read a book from beginning to end, you might want to skip this chapter, because it contains little useful information and will put you in a bad frame of mind for the rest of the book.

Thankfully, by Chapter 2, and throughout the rest of the book, the style settles down and becomes much better structured. Each chapter and each subsection within a chapter provides a quick description of its scope, and then provides rules, recommendations, and suggestions to go along with that. Some sections are thick with rules—in areas where the choice of language constructs can have a big impact on the block's reusability—while others are very sparse. For example, the chapter on coverage-driven verification has a total of only eight rules, whereas the much smaller subsection within the assertions chapter, called simple checkers, manages to include 17 rules.

Although I am sure the authors agonized over what should be a rule and what could be relaxed, there are some areas where it has resulted in strange priority inversions. In these cases, the recommendations are more actionable and useful than the rules, with the rules only doing hand waving and providing motherhood and apple pie statements. For a block of verification IP, for example, the book states that the IP must follow all rules to be considered compliant. Establishing compliance will clearly be an art and not a science.

Given that the book provides neither a lot of the background explanation (about, for example, why it recommends doing things in a particular way) nor a summary of basic techniques (such as object-oriented programming), it would have been very helpful for the authors to list, at the beginning of each chapter, other sources from which readers could learn the concepts. Unfortunately, the book does not offer such a resource; it also does not provide a bibliography.

I hope that ARM and Synopsys will commission the authors to write a second edition, based on feedback from readers and other valuable input from users and

the industry. It can be more actionable, bring in the wisdom of more people who have been struggling with these issues in industry, and include an open-source object class library. Then it will become a standard that the industry can really embrace.

ALTHOUGH it might seem like I have many issues with this book, they are all minor compared with the value the book provides. People who can look past these problems will find much wisdom that can help them develop methodologies in any verification language they choose. Even though better ways to do the same things might be possible in other languages, almost all of the concepts conveyed in this book are applicable to most verification environments. Everyone waited a long time for this book, but I think people will use it for years to come. It was worth the wait. ■

Brian Bailey is a functional-verification and business consultant to EDA and systems companies. Contact him at brian_bailey@acm.org or at <http://brianbailey.us>.

■ Direct questions and comments about this department to Grant Martin, 2424 Raven Road, Pleasanton, CA 94566; gmartin@ieee.org.




**Sign Up Today
for the IEEE
Computer
Society's
e-News**

Be alerted to

- articles and special issues
- conference news
- registration deadlines

Available for FREE to members.



computer.org/e-News