
Preface

When I need to learn a new piece of software I invent a little problem for myself that is within the domain of the application and then set out to solve it using the new tool. When the software package is a word processor, I'll use it to write a paper or article I'm working on; when the software is a drawing tool, I'll use it to draw some block diagrams of my latest creation. In the course of solving the problem, I learn how to use the tool and gain a practical perspective on which features of the tool are useful and which are not.

When the new software is a programming environment or a new programming language, the problem is a little different. I can't just apply the new language or environment to an existing problem. Unless I'm already familiar with the language, I don't want to commit to using it in a new development project. On the other hand, I may have an inkling that it would be best to use the new language. Otherwise, why would I be interested in it in the first place? I need a small program to help me understand the fundamental features and get a feel for how the language works. The program must be small and succinct, something I can write quickly and debug easily. Yet, it must use interesting language features I wish to learn.

Brian Kernighan and Dennis Ritchie solved this problem for all of us when they wrote the famous "Hello World" program. In their classic book *The C Programming Language*, they started off with a program that is arguably the most trivial program you could write in C that still does something. The beauty of Hello World is in its combination of simplicity and completeness. The program quoted here in its entirety, is not only simple, it also contains all of the constructs of a complete C program.

```
#include <stdio.h>

main()
{
    printf("hello, world\n");
}
```

All those years ago, I typed the program into my text editor, ran `cc` and `ld`, and a few seconds later saw my green CRT screen flicker with:

```
hello, world
```

Getting that simple program working gave me confidence that C was something I could conquer. I haven't counted how much C/C++ code I've written since, but it's probably many hundreds of thousands of lines. I've written all manner of software, from mundane database programs to exotic multi-threaded programs. It all started with Hello World.

The Open Verification Methodology (OVM) is a programming environment built upon SystemVerilog. It is designed to enable the development of complex testbenches. Like C (or SystemVerilog or SystemC), it will take some time and effort to study the OVM and understand how to apply all the concepts effectively. The goal of this book is to give you the confidence that running Hello World gave me all those years ago. If I, the author of this book, have done my job reasonably well, then somewhere along the way, as you read this book and exercise the examples, you should experience an aha! The metaphorical light bulb in your brain will turn on, and you will grasp the overall structure of the OVM and see how to apply it.

The premise of this book is that most engineers, like me, want to jump right into a new technology. They want to put their hands on it, try it out and see how it feels, learn the boundaries of what kinds of problems it addresses, and develop some practical experience. This is why quickstart guides and online help systems are popular. Generally, we do not want to read a lengthy manual and study the theory of operation first. We would rather plunge in, and later, refer to the manual only when and if we get stuck. In the meantime, as we experiment, we develop a general understanding of what the technology is and how to perform basic operations. Later, when we do crack open the manual, the details become much more meaningful.

This book takes a practical approach to learning about testbench construction. It provides a series of examples, each of which solves a particular verification problem. The examples are thoroughly documented and complete and delivered with build and run scripts that allow you to execute them in a simulator and observe their behavior. The examples are small and focused so you don't have to wade through a lot of ancillary material to get to the heart of an example.

This book presents the examples in a linear progression—from the most basic testbench, with just a pin-level stimulus generator, monitor, and DUT, to fairly sophisticated uses that involve stacked protocols, coverage, and automated testbench control. Each example in the progression introduces new concepts and shows you how to implement those concepts in a straightforward manner. Start by examining the first example. When you feel comfortable with it, move on to the second one. Continue in this manner, mastering each example and moving to the next.

The examples in the cookbook are there for you to explore. After you run an example, study the code to really understand its construction. The documentation provided with each example serves as a guidepost to point you to the salient features. Use this as a starting point to study the code organization, style, and other implementation details not explicitly discussed.

Play with the examples, too. Change the total time of simulation to see more results, modify the stimulus, add or remove components, insert print statements, and so on. Each new thing you try will help you more fully understand the examples and how they operate.

Feel free to use any of the code examples as templates for your work. For pieces that you find useful, cut and paste them into your code, or use them as a way to start developing your own verification infrastructures. Mainly, enjoy!

Mark Glasser, January 2009

Organization of This Book

Chapter 1. Describes some general principles of verification and establishes a framework for designing testbenches based on two questions—Does it work? and Are we done?

Chapter 2. This chapter provides an introduction to object-oriented programming and how OO techniques are applied to functional verification.

Chapter 3. Here, I introduce transaction-level modeling (TLM). The foundation of OVM is based on TLM. I illustrate basic put, get, and transport interfaces with examples.

Chapter 4. This chapter explains the mechanics of OVM, illustrating how to build hierarchies of class-based verification components and connect them with transaction-level interfaces. It also explains the essentials of using the OVM reporting facility.

Chapter 5. This chapter introduces the essential components of testbenches, such as drivers and monitors, and illustrates their construction with examples.

Chapter 6. This chapter discusses the essential topic of reuse—how to build components so that you have to do so only once and can apply what you have built in multiple situations.

Chapter 7. This chapter presents complete testbenches that use the types of components discussed so far and new ones, such as coverage collectors and scoreboards.

Chapter 8. OVM provides a facility called sequences for building complex stimulus generators. Sequences are discussed in this chapter, including how to construct sequences and how to use them to form a test API.

Chapter 9. It is important to reuse block-level testbenches when testing subassemblies or complete systems. This chapter illustrates some techniques for taking advantage of existing testbench components when constructing a system from separate blocks.

Chapter 10. SystemVerilog and OVM motivate new coding conventions. This chapter discusses some ways of constructing code to ensure that it is efficient, readable, and of course, reusable.

Obtaining the OVM Kit

You can get the open source OVM kit from www.ovmworld.com. The OVM kit contains complete source code and documentation.

Obtaining the Example Kit

The code used to illustrate concepts in this text is derived from the OVM cookbook kit available from Mentor Graphics. You can download the kit from www.mentor.com. Many of the snippets throughout the text have line numbers associated with them and, in some cases, a file name. The file names and line numbers are from the files in the Mentor OVM example kit.

Using the OVM Libraries

The OVM SystemVerilog libraries are encapsulated in a package called `ovm_pkg`. To use the package, you must import it into any file that uses any of the OVM facilities. The OVM library also contains a collection of macros that are useful in some places. You will need to include those as well as import the package

```
import ovm_pkg::*;  
'include "ovm_macros.svh"
```

To make the OVM libraries available to your SystemVerilog testbench code, you must compile it into the work library. This requires two command line options when you compile your testbench with Verilog:

```
+incdir+<location-of-OVM-libraries>/src  
<location-of-OVM-libraries>/src/ovm_pkg.sv
```

The first option directs the compiler to search the OVM source directory for include files. The second option identifies the OVM package to be compiled.

Building and Running the Examples

Installing the cookbook kit is a matter of unpacking the kit in a convenient location. No additional installation scripts or processes are required. You will have to set the `OVN_HOME` environment variable to point to your installation of OVM:

```
% setenv OVM_HOME <ovm-location>
```

Each example directory contains a `run_questa` script and one or more `compile_*` scripts. The `run_questa` script runs the example in its entirety. The compile script is a file that is supplied as an argument to the `-f` option on the compiler command line. Each example is also supplied with a `vsim.do` file that contains the simulator commands needed to run each example.

The simplest way to run an example is to execute its `run_questa` script:

```
% ./run_questa
```

This script compiles, links, and runs the example. You can also run the steps manually with the following series of commands:

```
% vlib work  
% vlog -f compile_sv.f  
% vsim -c top -do vsim.do
```

You must have the proper simulator license available to run the examples.

Who Should Read This Book?

This book is intended for electronic design engineers and verification engineers who are looking for ways to improve their efficiency and productivity in building testbenches and completing the verification portion of their projects. A familiarity with hardware description languages (HDL) in general, and specifically SystemVerilog, is assumed. It is also assumed that you know how to write programs in SystemVerilog, but it is not necessary to be an expert. Familiarity with object-oriented programming or OO terminology is helpful to fully understand the OVM. If you are not yet

familiar with OO terminology, not to worry, the book introduces you to the fundamental concepts and terms.

Acknowledgements

The author wishes to acknowledge the people who contributed their time, expertise, wisdom, and in some cases, material to this project. This book would never have come to completion without their dedication to this project.

Without Adam Rose's simple, yet brilliant observation that construction of hierarchies of class-based components in SystemVerilog can be done in the same manner as SystemC, OVM and its predecessor AVM would not exist. Adam was also a key participant in the development of the TLM-1.0 standard which has greatly influenced the nature of OVM. Tom Fitzpatrick, who has been involved in the project since the earliest days of AVM, provided some material and helped refine the text. Rich Edelman, with humor, good grace, and a keen eye for detail, and Andy Meyer, with his amazingly deep reservoir of verification knowledge, allowed me to bounce ideas around with them and helped me crystallize the concepts and flow of the material. Adam Erickson, who is a true code wizard and an expert in object-oriented patterns, always keeps me honest.

Todd Burkholder taught me about narrative flow and loaned me some of his English language skills to smooth out awkward sentences. Jeanne Foster did the detailed copy editing and an insightful job of producing an index.

Thanks to Harry Foster, who inspired the HFPB protocol and encouraged me to write this book. Hans VanderSchoot did a detailed review of the text and suggested many good ideas for improving the text. Also thanks to Kurt Schwartz of WHDL who reviewed an early draft. Cliff Cummings provided excellent advice on construction of the RTL examples.

A special thanks to Jan Johnson, who sponsored and supported the OVM Cookbook project from its inception.



<http://www.springer.com/978-1-4419-0967-1>

Open Verification Methodology Cookbook

Glasser, M.

2009, XX, 235 p., Hardcover

ISBN: 978-1-4419-0967-1