# Things to Remember

CHAPTER 16

*"There goes my tail again." —Eeyore*
*Paraphrased from Winnie-the-Pooh, by A.A. Milne*

An ending is, by definition, a new beginning. This, the last chapter, provides a good opportunity to review some of the handbook's main points. The authors sincerely hope that this is also a beginning for you to benefit from using some of the techniques presented in the preceding chapters.

This chapter is the 30,000-foot view of what we have covered. A wise, experienced manager once told the authors, "If you want your team to remember something, tell them at most three things." We take that advice—sort of—and present the three most important ideas of each part in the book.

We hope that this handbook, and its accompanying code, was and will continue to be useful. In the end, however, it is your job to verify the chip.

## Part I: Use C++ and Layers!

In the first part of the book we introduced verification, C++, object oriented programming, and what a layered verification looked like. Here are the important points:

- C++ is a good language for verification.
- Use OOP techniques for verification, but not to excess.
- Layering is the main technique for a verification system.

C++ is what the majority of the software industry uses. As a result there are lots of books on C++, tools to support a C++-based development flow, and lots of open-source code. C++ has a rich and well-polished feature set. To learn C++, start slowly and add language features carefully.

The verification world is a bit enamored with OOP. We are probably in the early stages of settling down and using it, or not, where appropriate. By using OOP techniques we can communicate our architectural intent clearly.

The concept of layering, formally described as abstraction, roles, and responsibilities, is perhaps the single most important technique we can use. We presented terms for layers that we later implemented as classes and conventions.

## Part II: An Open-Source Approach

In this part of the handbook we presented some code that has proved useful to us a nd those at other companies. That code may not have everything you want, but it should be flexible enough for you to adapt it to your needs. We noted specifically the following:

- Teal is a minimal, yet sufficient, interface to the HDL. It is the enabler when one uses C++ for functional verification.
- Truss provides a flexible, yet well-defined, verification application framework.
- A simple, but complete, example can be useful.