

## Chapter 4

### Basic OOP

#### 4.1 Introduction

With procedural programming languages such as Verilog and C, there is a strong division between data structures and the code that uses them. The declarations and types of data are often in a different file than the algorithms that manipulate them. As a result, it can be difficult to understand the functionality of a program, as the two halves are separate.

Verilog users have it even worse than C users, as there are no structures in Verilog, only bit vectors and arrays. If you wanted to store information about a bus transaction, you would need multiple arrays: one for the address, one for the data, one for the command, and more. Information about transaction N is spread across all the arrays. Your code to create, transmit, and receive transactions is in a module that may or may not be actually connected to the bus. Worst of all, the arrays are all static, so if your testbench only allocated 100 array entries, and the current test needed 101, you would have to edit the source code to change the size and recompile. As a result, the arrays are sized to hold the greatest conceivable number of transactions, but during a normal test, most of that memory is wasted.

Object Oriented Programming (OOP) lets you create complex data types and tie them together with the routines that work with them. You can create testbenches and system-level models at a more abstract level by calling routines to perform an action rather than toggling bits. When you work with transactions instead of signal transitions, you are more productive. As a bonus, your testbench is decoupled from the design details, making it more robust and easier to maintain and reuse on future projects.

If you already are familiar with OOP, skim this chapter, as SystemVerilog follows OOP guidelines fairly closely. Be sure to read section 4.18 to learn how to build a testbench. Chapter 8 presents advanced OOP concepts such as inheritance and more testbench techniques; it should be read by everyone.

#### 4.2 Think of Nouns, not Verbs

Grouping data and code together helps you in creating and maintaining large testbenches. How should data and code be brought together? You can start by thinking of how you would perform the testbench's job.

The goal of a testbench is to apply stimulus to a design and then check the result to see if it is correct. The data that flows into and out of the design is