Chapter 3

Procedural Statements and Routines

3.1 Introduction

As you verify your design, you need to write a great deal of code, most of which is in tasks and functions. SystemVerilog introduces many incremental improvements to make this easier by making the language look more like C, especially around argument passing. If you have a background in software engineering, these additions should be very familiar.

3.2 Procedural Statements

SystemVerilog adopts many operators and statements from C and C++. You can declare a loop variable inside a **`for`** loop that then restricts the scope of the loop variable and can prevent some coding bugs. The increment **`++`** and decrement **`--`** operators are available in both pre- and post- form. If you have a label on a **`begin`** or **`fork`** statement, you can put the same label on the matching **`end`** or **`join`** statement. This makes it easier to match the start and finish of a block. You can also put a label on other SystemVerilog end statements such as **`endmodule`**, **`endtask`**, **`endfunction`**, and others that you will learn in this book. Example 3-1 demonstrates some of the new constructs.

Example 3-1 New procedural statements and operators

```
initial
  begin : example
  integer array[10], sum, j;

  // Declare i in for statement
  for (int i=0; i<10; i++)       // Increment i
    array[i] = i;

  // Add up values in the array
  sum = array[9];
  j=8;
  do                             // do...while loop
    sum += array[j];             // Accumulate
  while (j--);                   // Test if j=0
  $display("Sum=%4d", sum);      // %4d - specify width
end : example                    // End label
```