SystemVerilog adds several new operators and procedural statements to the Verilog language that allow modeling at a more abstract, C-like level. Additional enhancements convey the designer's intent, helping to ensure that all software tools interpret the procedural statements in the same way. This chapter covers these operators and procedural statements, and offers guidelines on how to properly use these new constructs.

This SystemVerilog features presented in this chapter include:

- New operators
- Enhanced for loop
- New bottom testing loop
- New jump statements
- Enhanced block names
- Statement labels
- Unique and priority decisions
6.1 New operators

6.1.1 Increment and decrement operators

++ and -- operators

SystemVerilog adds the ++ increment operator and the -- decrement operator to the Verilog language. These operators are used in the same way as in C. For example:

```verbatim
for (i = 0; i <= 31; i++)
    @(posedge shift_clock) serial_out <= data[i];
```

Post-increment and pre-increment

As in C, the increment and decrement operators can be used to either pre-increment/pre-decrement a variable, or to post-increment/post-decrement a variable. Table 6-1 shows the four ways in which the increment and decrement operators can be used.

Table 6-1: Increment and decrement operations

<table>
<thead>
<tr>
<th>Statement</th>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>j = i++;</td>
<td>post-increment</td>
<td>j is assigned the value of i, and then i is incremented by 1</td>
</tr>
<tr>
<td>j = ++i;</td>
<td>pre-increment</td>
<td>i is incremented by 1, and j is assigned the value of i</td>
</tr>
<tr>
<td>j = i--;</td>
<td>post-decrement</td>
<td>j is assigned the value of i, and then i is decremented by 1</td>
</tr>
<tr>
<td>j = --i;</td>
<td>pre-decrement</td>
<td>i is decremented by 1, and j is assigned the value of i</td>
</tr>
</tbody>
</table>

The following code fragments show how pre-increment versus post increment can affect the termination value of a loop.

```verbatim
while (i++ < LIMIT) begin: loop1
    ... // last value of i will be LIMIT
end

while (++j < LIMIT) begin: loop2
    ... // last value of j will be LIMIT-1
end
```

In loop1, the current value of i will first be compared to LIMIT, and then i will be incremented. Therefore, the last value of i within the loop will be equal to LIMIT.