testbench, it will be in an initial block inside a program module. Its function is to check the condition given and report if an error has occurred. When completed, and assuming there is no error, execution continues with the next procedural statement.

- A ***concurrent assertion*** is an assertion that is always active. It acts as a separate (concurrent) element whose sole purpose is to check a property that is specified in the assertion. Once started it continues to execute.

Immediate assertions will be discussed in this section along with other preliminary information to provide a quick understanding of what an assertion might check, and to show how they fit into the simulation kernel. Concurrent assertions, a much larger set of topics, is left for later sections of the Chapter.

```
1  module adder;
2    logic  [3:0]  a, b, s;
3    logic  [3:0]  co;
4    logic        cIN;
5
6    full_add b0 (s[0], co[0], a[0], b[0], cIN);
7    full_add b1 (s[1], co[1], a[1], b[1], co[0]);
8    full_add b2 (s[2], co[2], a[2], b[2], co[1]);
9    full_add b3 (s[3], co[3], a[3], b[3], co[2]);
10
11   initial begin: testbench
12     a = 3; b = 2; cIN = 0;
13     #1 $display
14       ("time=%d, a=%b, b=%b, cIN=%b, s=%b, co=%b",
15          $stime, a, b, cIN, s, co);
16     checkadd0: assert (s === a + b + cIN)
17       $display ("%m works! a=%b, b=%b, cIN=%b, s=%b,\
18             co=%b", a, b, cIN, s, co);
19       else
20         $error ("%m says no cigar! a=%b, b=%b, cIN=%b,\
21             s=%b, co=%b", a, b, cIN, s, co);
22 ...  // other code omitted
23   end
24 endmodule: adder
25
26 module full_add
27   (output    sum, co,
28    input     a, b, cin);
29
30    xor (sum, a, b, cin);
31    assign co = a&b | a&cin | b&cin;
32 endmodule: full_add
33
34 time=        1, a=0011, b=0010, cIN=0, s=0101, co=0010
35 adder.testbench.checkadd0 works! a=0011, b=0010, cIN=0,
                                    s=0101, co=0010
```

Example 9.1 — Immediate Assertion and Printout

9.1.1 An Immediate Assertion Example

An example of an immediate assertion is shown in Example 9.1. The example is of a 4-bit adder made out of four instantiated 1-bit full adders. The module definition of the full adder is shown on lines 26-32; the full adder is then instantiated 4 times in the adder module to create a structural model of the 4-bit adder (lines 6-9). On lines 11-23, an initial block that sets the inputs to the adder and observes the output is shown. As a simple illustration, the initial block sets the inputs to a=3, b=2, and cIN=0. The initial block delays one time unit, allowing the values to propagate, and then $displays the results. The simulation output (line 34) of the example shows that the results $displayed are correct: 3+2 equals 5.

An immediate assertion is then executed (line 16) to check that the result is correct. The form of the immediate assertion is:

```
1  label: assert (expression)
2        pass_Statement else fail_Statement;
```

The label and pass_Statement are optional. The immediate assert acts as an "if" statement. If the expression is TRUE, the pass_Statement is executed; otherwise the fail_Statement is executed. Since most verification engineers only want to know if there was an error, the pass_Statement is normally omitted.

The simulation output of the assertion statement printing its pass_Statement is shown on line 35.