There are several important points to note from this simple example:

- An immediate assertion is exactly like an if statement but it's clear that a check of functionality is being done. There is no mistake that this is not part of the functionality of the design or testbench.
- The detailed functionality of the interconnected modules (lines 6-9) is being compared with a relatively simple statement of their functionality (line 16). This is typical when validating a design. Since all of those instantiated modules and interconnections are supposed to implement a + b + cIN, that's the check that is made. i.e., there's no reason to compare a complex design with a separate complex design — you'll specify one of them incorrectly! Probably both.

Now, let's change one character in module adder's definition: we'll change line 9 to

```
1   full_add b3 (s[3], co[3], a[3], b[3], co[1]);
```

Note that the carry in is specified incorrectly as the carry output of stage 1 (co[1]) rather than (the correct) stage 2. When the testbench is run again, the resulting printout will be

```
1   time=         1, a=0011, b=0010, cIN=0, s=1101, co=0010
2   "ImmAssertBasic.sv", 14: adder.testbench.checkadd0: started at 1s failed at 1s
3         Offending '(s === ((a + b) + cIN))'
4   Error: "ImmAssertBasic.sv", 14: adder.testbench.checkadd0: at time 1
5   adder.testbench.checkadd0 says no cigar! a=0011, b=0010, cIN=0, s=1101, co=0010
```

Line 1 is the $display statement printing but note that s=4'b1101. Clearly this is not the binary result of 2+3! Line 5 shows the immediate assertion's else clause with its $error statement printing. This statement is like a $display statement in that it prints the message given. However, in contrast to the $display, it causes a runtime error. Lines 2-4 are printed by the simulator.

Note that "%m" is used in the print string (lines 17 and 20 of Example 9.1). This prints the hierarchical context of the statement so it is easier to find what caused the printing. In this case the hierarchical context is its label checkadd0 which is inside the begin-end block named testbench which is inside module adder; thus "adder.testbench.checkadd0."

In addition to $error, there are other tasks that can be used to indicate the level of severity of an error. These are available only in an assertion statement.

- ***$fatal*** — A run-time fatal error is generated which also implicitly calls $finish which stops the simulation. It has two arguments. The first argument passed to $fatal is also passed to $finish (by default it is 1). The second argument is the string to be printed.
- ***$error*** — Called to indicate a run-time error. It's argument is a print string.
- ***$warning*** — Called to indicate a run-time warning. It's argument is a print string.
- ***$info*** — Called to indicate an issue with no specific severity. It's argument is a print string.

For all of these, the print string uses the same syntax as a $display task.

Immediate assertions can be used in other contexts. For instance, if a testbench is opening a file to print some results, the code could be: