exhaustive and, therefore, a guarantee of conformance to specification is impossible. Such a guarantee is possible with a *formal verification method* [380], which mathematically proves the correctness of the design. A restricted form of formal verification, known as *model checking* [168, 453], verifies finite state concurrent systems by an exhaustive search of the state-space. It verifies whether a given specification is true. According to Clarke *et al.* [168], an efficiently implemented model checking procedure will always terminate with a yes/no answer and can be run on moderate-sized machines, though not on an average desktop computer. Thus, the high complexity of formal methods allows their use only at the higher behavior level. In spite of the incompleteness, simulation provides a better check on the manufacturability of the design. An ideal system of design verification should combine the behavior-level formal verification with the logic and circuit-level simulation.

Example 5.1 *Simulating an adder circuit: Consider a combinational logic circuit designed to add two 32-bit binary integers. This circuit has 64 binary inputs and 33 outputs. To completely verify the correctness of the implemented logic, we must simulate* 2^{64} *input vectors and check that each produces the correct sum output. This circuit may have about 200 gates and a fast logic simulator may require* $1\mu s$ *to simulate one vector. The time required to complete the simulation is:*

$$\frac{2^{64} \times 10^{-6}}{3600 \times 24 \times 365} \approx 584,942 \; years$$

This is clearly impractical. So, the designer must simulate some selected vectors. For example, one may add pairs of integers where both are non-zero, one is zero, and both are zero. Then add a large number of (say, 10^6*) randomly generated integer-pairs. Such heuristics, though they seem arbitrary, can effectively find many possible errors in the designed logic. The next example illustrates a rather simple heuristic.*

Example 5.2 *Design verification heuristic for a ripple-carry adder: Figure 5.2 shows the logic design of an adder circuit. The basic building block in this design is a full-adder that adds two data bits,* A_n *and* B_n*, and one carry bit,* C_n*, to produce sum and carry outputs,* S_n *and* C_{n+1}*, respectively. For logic verification, one possible strategy is to select a set of vectors that will apply all possible inputs to each full-adder block. For example, if we set* $A_0 = B_0 = 0$ *and apply all four combinations (00, 01, 10, 11) to* A_1 *and* B_1*, and then set* $A_0 = B_0 = 1$ *and again apply the four combinations to* A_1 *and* B_1*, these eight vectors include all eight inputs for the* $FA1$ *block. Logic simulation of these vectors will thoroughly check* $FA1$*, since the states of both outputs* S_1 *and* C_2 *are provided by the simulator. One significant advantage of simulation is that all internal signals of the circuit can be examined. This reduces the complexity of verification. Table 5.1 gives a set of eight vectors for verifying a 4-bit ripple-carry adder. Interestingly, the regular pattern in each vector allows us to expand the width of the vector, without increasing the number of vectors, for applying this heuristic to an adder of arbitrarily large size.*