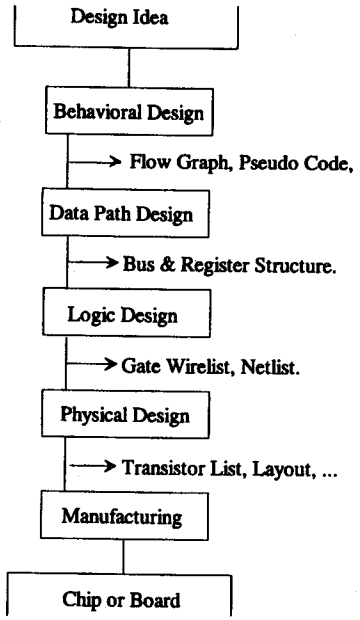


VHDL
ANALYSIS & MODELING
OF DIGITAL SYSTEMS
Z. NAVABI

Figure 1.1 A digital system design process. (From M. R. Barbacci, *The ISPS Computer Description Language*, Carnegie- Mellon University, 1981, p. 70.)

pseudocode. The designer specifies the overall functionality and an input to output mapping without giving architectural or hardware details of the system under design.

The next phase in the design process is the design of the system data path. In this phase, the designer specifies the registers and logic units necessary for implementation of the system. These components may be interconnected using either bidirectional or unidirectional busses. Based on the intended behavior of the system, the procedure for controlling the movement of data between registers and logic units through busses is then developed. Figure 1.2 shows a possible result of the data path design phase. Data components in the data part of a circuit communicate via system busses, and the control procedure controls flow of data between these components. As shown, this design phase results in the architectural design of a system with specification of the control flow. No information about the implementation of the controller—e.g., hardwired, encoding technique, or microprogrammed—is given in this phase.

Logic design is the next step in the design process and involves the use of primitive gates and flip-flops for the implementation of data registers, busses, logic units, and their controlling hardware. The result of this design stage is a netlist of gates and flip-flops. Components

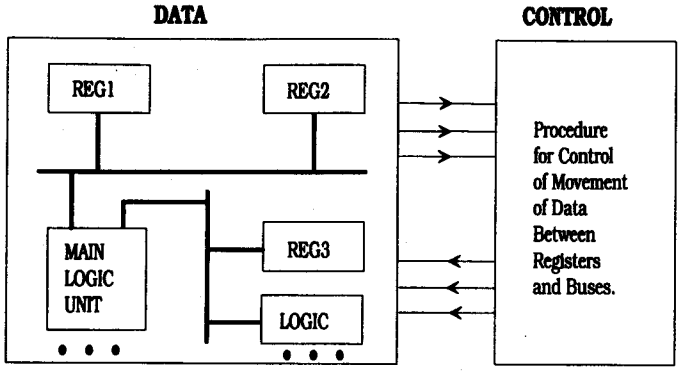


Figure 1.2 Result of the data path design phase.

used and their interconnections are specified in this netlist. Gate technology and even-gate-level details of flip-flops are not included in this netlist.

The next design stage transforms the netlist of the previous stage into a transistor list or layout. This involves the replacement of gates and flip-flops with their transistor equivalents or library cells. This stage considers loading and timing requirements in its cell or transistor selection process.

The final step in the design is manufacturing, which uses the transistor list or layout specification to burn fuses of a field-programmable device or to generate masks for integrated-circuit (IC) fabrication.

1.1.1 Design automation

In the design process, much of the work of transforming a design from one form to another is tedious and repetitive. From the point of view of a digital system designer, the design phase is complete when an idea is transformed into an architecture or a data path description. The rest is routine work and involves tasks that a machine can do much faster than a talented engineer. The same can be said about the verification process; that is, a machine can be programmed to verify functionality or timing of a designed circuit much easier than any human can.

Activities such as transforming one form of a design into another, verifying a design stage output, or generating test data can be done at least in part by computers. This process is referred to as design automation (DA).

Design automation tools can help the designer with design entry, hardware generation, test sequence generation, documentation, veri-

fication, and design management. Such tools perform their specific tasks on the output of each of the design stages of Fig. 1.1. For example, to verify the outcome of the data path design stage, the bussing and register structure is fed into a simulation program. Also, to generate tests for register transfer faults, a design automation tool can be used for processing this level of system description and producing tests that can be used by a test engineer. Other DA tools include a synthesizer that can automatically generate a netlist from the register and bus structure of the system under design.

HDLs provide formats for representing the outputs of various design stages. An HDL-based DA tool for the analysis of a circuit uses this format for its input description, and a synthesis tool transforms its HDL input into an HDL which contains more hardware information. In the sections that follow, we discuss HDLs, digital system simulation, and hardware synthesis.

1.2 The Art of Modeling

The dictionary[1] defines *model* and *modeling* as follows:

mod-el (mŏd′l) *noun.* **1.** A small object, usually built to scale, that represents in detail another, often larger object. **2.a.** A preliminary work or construction that serves as a plan from which a final product is to be made. **b.** Such a work or construction used in testing or perfecting a final product. **3.** A schematic description of a system, theory, or phenomenon that accounts for its known or inferred properties and may be used for further study of its characteristics.

mod-el-ing (mŏd′l-ĭng) *noun.* **1.** The act or art of sculpturing or forming in a pliable material, such as clay or wax. **2.a.** Representation of depth and solidity in painting, drawing, or photography.

We can even go further in defining model and modeling from the point of view of a hardware designer and digital system design environments. Modeling is an art, and a modeler is one who uses certain modeling tools for representing an event, an object, or an idea as best as possible. The modeling tools, however, may be different from one art to another. For a painter, the modeling tools are the paintbrush, easel, paint, and the paint palette. In addition to being different in the way things are represented by various modelings, the level of details of representation in a model may be different from one model to another. For example one painting may represent a mountain at the

[1]*The American Heritage® Dictionary of the English Language, Third Edition,* copyright © 1992 by Houghton Mifflin Company.

detailed level of rocks and plantation, while another painting of the same mountain may represent it from a distant view showing its peak and the hills around it.

Models are used for different purposes. A painting may be used for decorating a room, while another type of a model, for instance a piece of music, may be used to express a political view or an event. Shostakovich wrote his Symphony No. 7, the Leningrad Symphony, about the siege of his native city of Leningrad (St. Petersburg) by German troops. The symphony models the spirit of this city during the war, life in this city under siege, and the eventual victory over the invaders. In this model, he employs a certain level of details and uses his general modeling schemes for doing so.

A hardware designer models a circuit using any available tools. The level of abstraction for this modeling depends on the purpose for which the model is intended. If the model is to be used for documenting the functionality of a circuit at a very abstract behavioral level, a relatively simple abstract model is all that is necessary. On the other hand, if the model is to be used for verification of the timing of the circuit, a more detailed description is needed. A hardware engineer models his or her circuit such that it imitates the actual hardware component as closely and accurately as possible for its intended purpose. A good modeler uses available hardware modeling tools to produce an elegant and artistic model of the hardware part.

Available modeling tools to a hardware engineer include paper and pencil, schematic capture programs, breadboarding facilities, and hardware description languages. The newest and the most promising of all such tools are hardware description languages. These modeling tools enable a hardware designer to model his or her circuit at many levels of abstractions for various design, analysis, and documentation purposes. Although all hardware description languages may be regarded as such, the level of model elegance and artistic representation of hardware may be different from one language to another.

1.3 Hardware Description Languages

HDLs are used to describe hardware for the purpose of simulation, modeling, testing, design, and documentation. These languages provide a convenient and compact format for the hierarchical representation of functional and wiring details of digital systems. Some HDLs consist of a simple set of symbols and notations that replace schematic diagrams of digital circuits, while others are more formally defined and may present the hardware at one or more levels of abstraction. Available software for HDLs include simulators and hardware synthe-

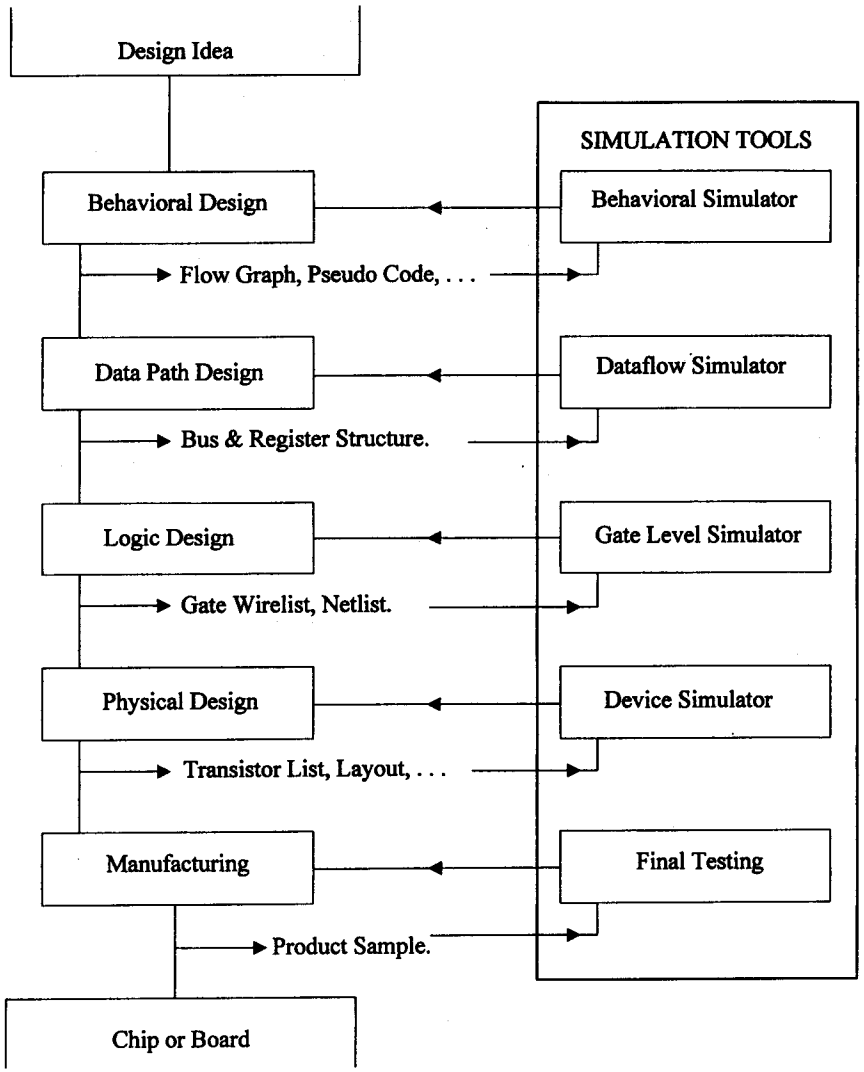


Figure 1.7 Verifying each design stage by simulating its output.

level or device simulation, runs much more slowly but provides more detailed information about the timing and functionality of the circuit. To avoid the high cost of low-level simulation runs, simulators should be used to detect design flaws as early in the design process as possible.

Regardless of the level of design to which a simulation program is applied, digital system simulators have generally been classified into *oblivious* and *event-driven* simulators. In oblivious simulation, each

circuit component is evaluated at fixed time points, while in event-driven simulation, a component is evaluated only when one of its inputs changes.

1.4.1 Oblivious simulation

As an illustration of the oblivious simulation method, consider the gate network of Fig. 1.8*a*. This is an exclusive-OR circuit that uses AND, OR, and NOT primitive gates and is to be simulated with the data provided in Fig. 1.8*b*.

The first phase of an oblivious simulation program converts the input circuit description to a machine-readable tabular form. A simple example of such a table is shown in Fig. 1.9. This table contains information regarding the circuit components and their interconnections, as well as the initial values for all nodes of the circuit.

After the initialization of the circuit, the simulation phase of an oblivious simulation method reads input values at fixed time intervals, applying them to the internal tabular representation of the circuit. At time t_i, input values of a and b are read from an input file. These values replace the old values of a and b in the value column of the table of Fig. 1.9. Using these new values, the output values of *all*

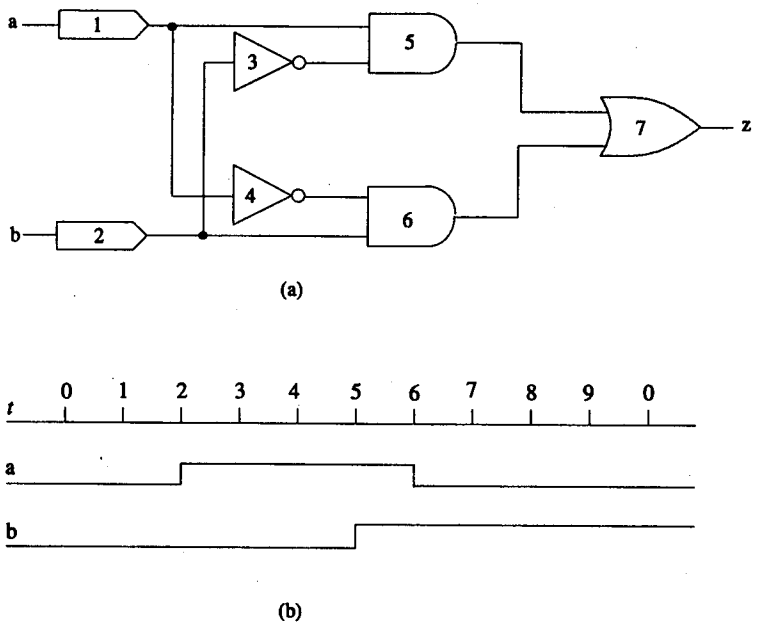


Figure 1.8 An exclusive-OR function in terms of AND, OR, and NOT gates. (*a*) Logical diagram; (*b*) test data.

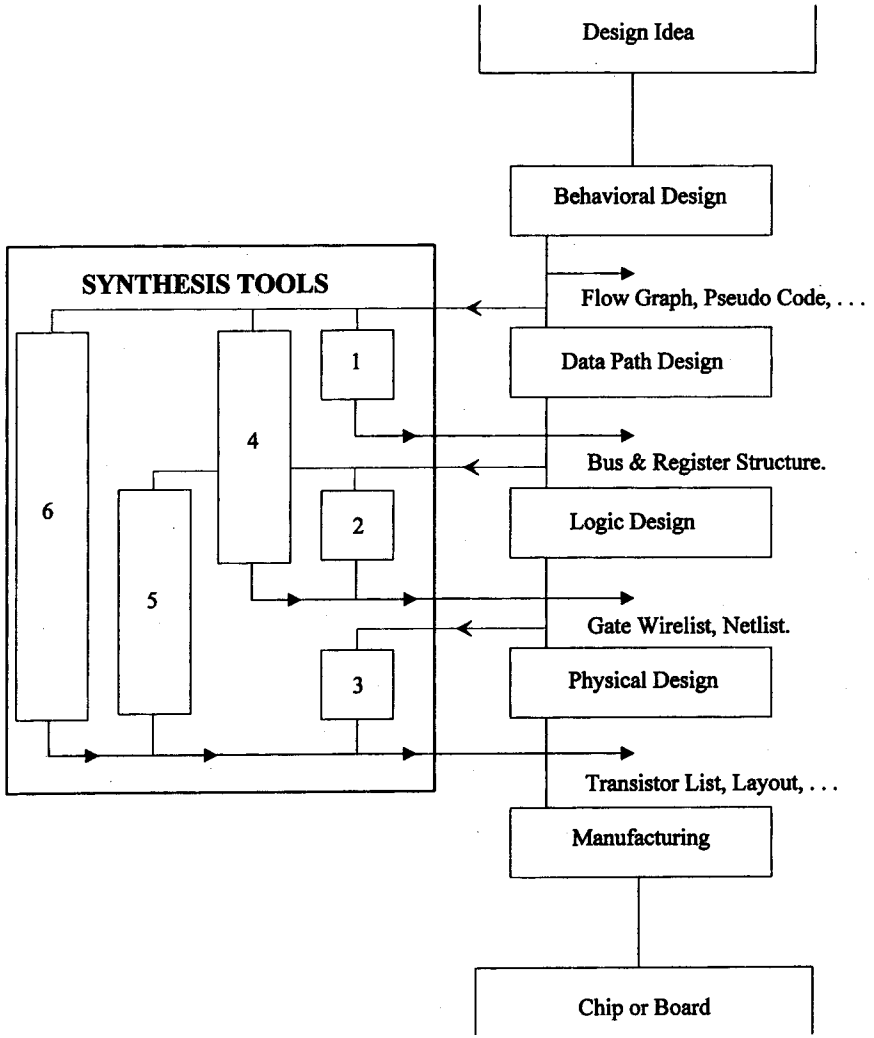


Figure 1.11 Categories of synthesis tools in a design process.

propriate sets of inputs. Resource sharing may be directed by a synthesis tool user through synthesis directives, or may be decided by the tool based on the input description. For example, as shown in Fig. 1.13, an *add* operation on register outputs *a* and *b*, placing the result on bus *c*, in one instance, and another operation adding the same inputs and placing the result on bus *d*, in another instance, can share the same adder unit without any hardware overhead. On the other hand, if the two instances of *add* operations use different inputs (e.g.,

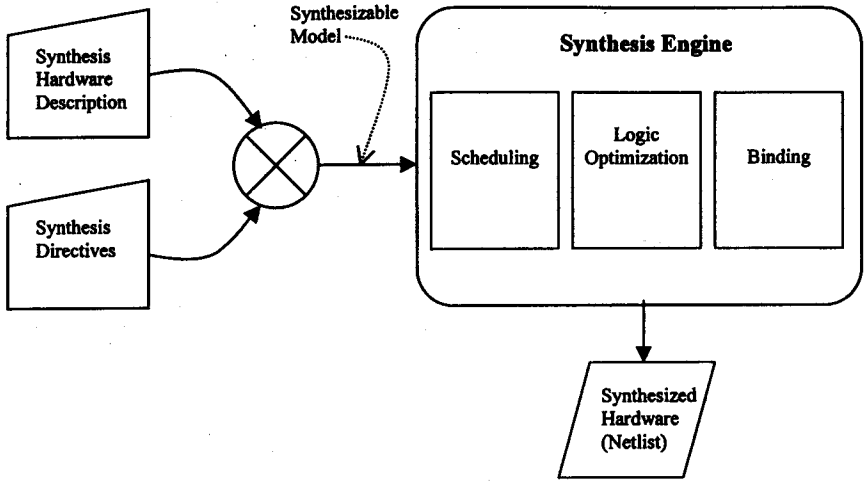
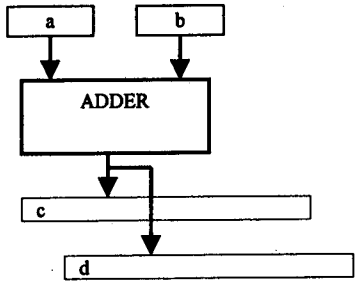


Figure 1.12 Synthesis process.

```
c <= a + b;
d <= a + b;
```




```
c <= a + b;
c <= x + y;
```

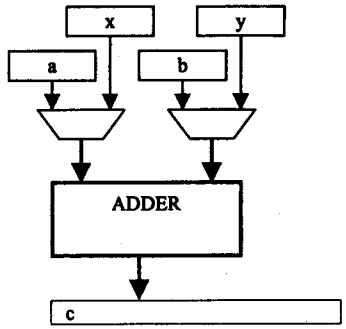


Figure 1.13 Resource sharing.

a and *b* in one instance, and *x* and *y* in another), multiplexers should be used for adder unit input selections.

In the logic optimization pass, boolean expressions are manipulated for better utilization of FPGA cells or chip area. The binding phase in a synthesis tool maps operations of an intermediate data structure, often in the form of a BDD (Binary Decision Diagram), to predefined library cells of a target hardware.

Tools that generate layout from netlists (tool category 3 in Fig. 1.11) have been available and in use for over two decades. These tools are