# CAD: Computer-Aided Design Tools

*"If it wasn't hard, they wouldn't call it hardware."*

Many digital designers with twenty years of experience consider this statement to be indisputable. Yet more and more, digital design is being carried out using *software*, and it's getting easier as a result.

The terms *computer-aided design (CAD)* and *computer-aided engineering (CAE)* are used to refer to software tools that aid the development of circuits, systems, and many other things. "CAD" is the more general term and applies to tools both inside and outside the electronics area, including architectural and mechanical design tools, for example. Within electronics, "CAD" often refers to *physical-design* tools, such as IC- and PCB-layout programs. "CAE" is used more often to refer to *conceptual-design* tools, such as schematic editors, circuit simulators, and HDL compilers. However, a lot of people in electronics (including the author) tend to use the two terms interchangeably. In this section, we'll discuss some CAD/CAE tools used by digital designers.

**IS HARDWARE NOW EASY-WARE?**

Since more and more hardware design and debugging is being carried out using software tools, is it really getting easier? Not necessarily.

In the author's experience, the increasing use of CAD means that instead of spending time fighting with soldering irons and test clips, many designers can now spend their time fighting buggy programs running in a buggy software environment.

## CAD.1 Hardware Description Languages

In previous decades, most logic design was performed graphically, using block diagrams and schematics. However, the rise of synthesizable HDLs coupled with programmable logic devices and very-large-scale ASIC technology in the 1990s has radically changed the way that the largest digital designs are done.

In traditional software design, high-level programming languages like C, C++, and Java have raised the level of abstraction so that programmers can design larger, more complex systems with some sacrifice in performance compared to hand-tuned assembly-language programs. The situation for hardware design is similar. The circuit produced by a VHDL or Verilog synthesis tool may not be as small or fast as one designed and tweaked by hand by an experienced designer, but in the right hands these tools can support much larger system designs. This is, of course, a requirement if we're ever to take advantage of the tens of millions of gates offered by the most advanced CPLD, FPGA, and ASIC technologies.

An introduction to HDL-based digital design and related tools appears in Section 5.1. I urge you to read this section if you haven't read it already. And you should learn either VHDL or Verilog in Section 5.3 or 5.4, and study the corresponding HDL examples in the rest of this book. To get the best understanding, nothing beats hands-on experience; fot that reason, a student version of popular HDL CAD software is included in many editions of this book.

## CAD.2 Schematic Capture

Except in homework assignments, your first step in designing a circuit usually is to convince someone that your proposed approach is the right one. That means preparing block diagrams and presentation slides, and discussing your ideas with managers and peers in a preliminary *design review*. Once your project has been approved, you can safely start on all the "fun stuff," such as drawing the schematic.

*design review*

Schematics used to be drawn by hand, but most are now prepared using *schematic editors,* CAD programs that run on engineering workstations. The process of creating a schematic on a computer is often called *schematic capture*. (Huh? How did it get loose?) This term is used because the schematic editor captures more information than just a drawing. Most information in the schematic has a "type" associated with it, so that selected information can be retrieved automatically as required later in the design process.

*schematic editor*
*schematic capture*

In order to simplify both entry and retrieval of information in a schematic, a schematic editor typically provides *fields* for each information type, either hidden or drawn automatically nearby the associated schematic element. Some typical fields and their uses are described below.

*fields*

- *Component type*. By specifying a component type (e.g., resistor, capacitor, 74FCT374, GAL16V8), the designer can call out a predrawn symbol from a *component library.* Some components have user-selectable alternate symbols (e.g., DeMorgan equivalents for gates). The component type is used both for manufacturing documentation and for simulation.

*component type*
*component library*

- *Component value*. Most analog components have a value that must be specified by this field (e.g., 2.7 kΩ). Additional distinguishing information may appear in a "rating" or "tolerance" field (e.g., 1/4 W, 1%). Digital components may give a speed rating here (e.g., -10 for a 10-ns PLD).

*component value*

**WHAT'S A WORKSTATION?**

An "engineering workstation" nowadays can be anything from a $500 PC to a $50,000 parallel processor. The "canonical" engineering workstation in terms of cost (what a typical company will spend per engineer) and performance (among the best) is a $5,000 dual-CPU workstation, whatever model you get for that price this year.

- *Approved-part number*. Using all of the distinguishing information from the previous fields, the CAD software may automatically select a part number from the company's "approved parts list" (e.g., 126-10117-0272 for a 2.7-kΩ, 1/4-W, 1% metal-film resistor), or a warning if no such part is available. (No, in large companies you can't use just any part in the catalog; all parts must be approved through a *parts-qualification* process.)

*approved-part number*

*parts qualification*

- *Reference designator*. This alphanumeric label distinguishes among multiple instances of similar parts and is used just about everywhere.

*reference designator*

- *Component location*. Using a coordinate system, this field may indicate the physical position of the component on an assembled PCB, simplifying the task of finding it during debugging. Of course, the final component location generally is not known during the initial circuit design. However, CAD software with *back-annotation* capability can insert this information into the schematic when the PCB layout is completed.

*component location*

*back annotation*

- *Pin numbers*. These specify the assignment of the signal pins on each component and are usually prespecified in the symbol that is called up from the component library. However, for SSI and other parts that have multiple identical sections in a single package, pin numbers (and even reference designators) might not be specified until the PCB layout is completed; this is another case where back annotation is handy.

*pin number*

- *Wire type*. Usually the only wire types are "ordinary" and "bus," where a bus is just a set of ordinary wires that are grouped for drawing purposes. However, other wire types, such as extra-wide PCB traces for analog signals that carry high current, may be indicated in some systems; such information is passed to the PCB-layout program.

*wire type*

- *Signal name*. The user may (and should) associate a name with each signal. The name is especially useful in simulation and debugging, but it does not affect the physical PCB layout.

*signal name*

- *Connection flag*. This drawing element indicates the connection of one signal line to another one on the same or a different page. Software can ensure that all outgoing connections are matched by incoming ones. In a hierarchical schematic, software can use the connection flags to generate signal names in a "logic symbol" that represents the entire page.

*connection flag*

**THE PRICE OF SOFTWARE**

The workstation price quoted in the previous box does *not* include the price of the CAD software that runs on the workstation. The price of CAD software can substantial, often exceeding the cost of the workstations that run it.

As a student, you may not be used to paying much for software, but you better get used to the fact that software has substantial economic value and requires large investment by its developers, who deserve a fair return on investment.

Once you've "captured" a circuit's schematic in a CAD system, there are lots of things you can do with it besides printing it. At least two documents can be generated for manufacturing purposes:

- *Parts list*. This is a list of components and their reference designators. *parts list*
- *Net list*. A *net* is a set of pins that are all connected to the same electrical node or signal. A *net list* specifies all such connections that are required by the schematic, typically using an alphabetically sorted list of signal names. For each signal name, it lists the device pins (identified by reference designator and pin number) where that signal is connected. *net* *net list*

  Alternatively, the net list may be sorted by reference designator and pin number, and show the signal name connected to each pin; sometimes this is called a *pin list*. *pin list*

The parts list and the net list are the main inputs to the PCB-layout process. The designer may also use the parts list to estimate the cost and reliability of the circuit, and the manufacturing department obviously can use it to order the parts.

## CAD.3 Timing Drawings and Specifications

Block diagrams and schematics are not the only drawings used in digital system design. Timing diagrams are an essential part of almost any documentation package.

Larger systems are decomposed into smaller subsystems that communicate through well-defined interfaces. The definition of an interface typically contains not only signal names and functions, but also specifications of expected timing behavior. The starting point is typically a specification of maximum and sometimes minimum clock frequencies. Inputs to a subsystem have their setup- and hold-time requirements with respect to a clock edge specified. Outputs of a subsystem have their minimum and maximum delays from a clock edge specified.

Programs such as TimingDesigner (`www.chronology.com`) can automate the tedious task of creating complicated timing diagrams and specifications, where changes in one timing parameter may propagate to many others.

## CAD.4 Circuit Analysis and Simulation

The component library of a sophisticated CAD system contains more than just a symbol for each component. For ICs, the library may contain a *component model* describing the device's logical and electrical operation. In fact, HDLs like VHDL and Verilog were originally invented strictly for the purpose of modeling the logical behavior of components and their interconnection in systems. Using these models, logical and timing errors can be found in simulation. *component model*

As a minimum, the model for an IC indicates whether each pin is an input or output. With just this information, a *design-rule checker* can detect some of the more common "stupid mistakes" in a design, such as shorted outputs and *design-rule checker*

floating inputs. With the addition of input loading and output driving characteristics for each pin, the program can also determine whether any output's fanout capability is exceeded.

*timing verification*

The next step is *timing verification*. Even without a detailed model of an IC's logical behavior, the component library can provide a worst-case delay value for each input-to-output path, and setup and hold times for clocked devices. Using this information, a *timing verifier* can find the worst-case delay paths in the overall circuit, so the designer can determine whether the timing margins are adequate.

*timing verifier*

*simulation*

*simulator*

Finally, the library may contain a detailed model of each component's logical behavior, in which case *simulation* may be used to predict the overall circuit's behavior for any given sequence of inputs. The designer provides an input sequence, and a *simulator* program determines how the circuit would respond to that sequence. The simulator output is usually displayed graphically, in the form of timing waveforms that the designer could see on an oscilloscope or logic analyzer if the same inputs were applied to a real circuit. In such an environment, it is possible to debug an entire circuit without "breadboarding" it, and to assemble a PCB that works on the very first try.

Simulators use different models for different types of components. Analog simulators like SPICE may use mathematical models with just one parameter for a resistor and with anywhere from a few to dozens of parameters for a bipolar transistor. Based on these models and the circuit's structure, the simulator develops and solves equations to determine the circuit's behavior.

Digital simulators normally use one of the two following models for each logic element:

*behavioral model*

- *Behavioral model*. In such a model, the software "understands" the basic function of the logic element. For example, an AND gate is understood to produce a 1 output with delay $t_{pLH}$ after all inputs are 1, and a 0 output with delay $t_{pHL}$ after any input is 0.

*structural model*

- *Structural model*. Larger elements may be modeled by connecting groups of smaller elements that have behavioral models.

We saw that VHDL and Verilog support both types of model. For example, Table 6-14 on page 399 gave a VHDL behavioral model for a 3-to-8 decoder. Both VHDL and Verilog have mechanisms (not covered in this book) for specifying the delay from input to output in such a model. Alternatively, Table 6-13 on page 398 gave a structural model of a 2-to-4 decoder as a set of gates connected according to the internal logic diagram of Figure 6-32 on page 385. Assuming that timing information already exists for the individual gates in the structure (`inv` and `and3`), the structural model can predict circuit timing more accurately, since it can follow the actual circuit path taken by each different input transition.

*software model*

Both of the foregoing models are called *software models* because the chip's operation is simulated entirely by software. Some chips, including microprocessors and other LSI parts, require huge software models. In many cases, the chip manufacturers don't provide such models (they're considered to be proprietary information), and the models would take months or years for a user to develop. To get around this problem, innovative CAD companies have developed a third type of model:

*physical model*
*hardware model*

- *Physical model*, sometimes called a *hardware model*. A real, working copy of the chip is connected to the computer that's running the simulation. The simulator applies inputs to the real chip and observes its outputs, which are then used as inputs to the other models in the simulation.

*uninitialized sequential circuits*

A digital designer's input to a logic simulator is a schematic and a sequence of input vectors to be applied to the circuit in simulated time. The first output that the designer usually sees is "`xxxxx`", meaning that the simulator can't figure out what the output will be. This usually occurs because the circuit contains sequential circuits (flip-flops and latches) that are not initialized. To obtain meaningful simulator output, the designer must ensure that all circuits are explicitly reset to a known value, even if reset is not required for correct operation in the real system. For example, even a very simple circuit like the free-running modulo-16 counter in Figure 8-29 on page 715 can't be simulated, because the simulator has no way of knowing its starting state.

*simulation inefficiency*

Once the simulated circuit has been initialized, the designer's ability to probe its operation is limited only by the simulator's speed. Simulation of complex circuits can be very slow, compared to the speed of the real thing. Depending on the level of detail being simulated, the simulation time can be $10^3$ to $10^8$ times longer than the interval being simulated. A strictly functional simulation, in which all components are assumed to have zero delay, is fastest. A much more realistic simulation, one that calculates and considers a worst-case range of delays for all signal paths, is the slowest.

## CAD.5 PCB Layout

Printed-circuit-board layout can be carried out by a program or by a person (a PCB designer) or by a combination of the two. PCB designers are usually people who like to solve puzzles. Their job is to fit all of the components in the schematic onto a board of a given size, and then to hook up all the connections as required by the schematic—and usually to do so with the smallest possible PCB size and number of interconnection layers.

PCB design can be done automatically by software, but the results aren't always so good. Both the PCB designer and the circuit designer normally must guide the initial component-placement steps, taking into account many practical concerns, including the following:

- *Mechanical constraints.* Certain connectors, indicators, and other components may have to be placed at a particular location on the PCB because of a predetermined mechanical requirement.

  *mechanical constraints*

- *Critical signal paths.* An automatic placement program can try to place components on the board in a way that minimizes the average length of interconnections, but designers usually have a more intuitive feel for which areas of the circuit and which specific signals have the most critical requirements. In fact, the circuit designer normally provides the PCB designer with a *floorplan,* a suggested placement for groups of components that reflects the designer's ideas about which things need to be close to each other.

  *critical signal paths*

  *floorplan*

- *Thermal concerns.* Some areas of the PCB may have better cooling than others when the board is installed in a system, and certainly some components run hotter than others. The designers must ensure that the operating temperature range of the components is not exceeded.

  *thermal concerns*

  The temperature anywhere on a well-designed PCB in a properly cooled system typically will not exceed 10°C above ambient, but a power-hungry chip (such as a microprocessor) placed in an area with no air flow could create a temperature rise of 30°C or more. If the ambient temperature were 40°C, the actual temperature at the microprocessor would be 70°C, which is at the limit of the commercial operating range for most chips.

- *Radio-frequency emissions.* Electronic equipment radiates radio-frequency energy as a side effect. Government agencies in the U.S. and other countries specify the maximum permissible level of emissions; in the U.S. the limits are spelled out in *FCC Part 15* regulations.

  *radio-frequency emissions*

  *FCC Part 15*

  A circuit's emissions levels usually depend not only on component performance and circuit configuration, but also on the physical layout of the PCB. During layout, special attention must therefore be paid to component placement, grounding, isolation, and decoupling, in order to minimize unwanted (and illegal!) radio-frequency emissions.

- *Stupid mistakes.* Perhaps the most important step in PCB design, especially when using new or unfamiliar components, is checking for stupid mistakes. Almost every digital designer, even successful ones, can tell you stories of laying out a PCB using the mirror image of a new part's pinout, putting mounting holes or connectors in the wrong place, using the wrong version of an IC package, using the wrong sex of a connector, leaving out the inner connection layers when the board is fabricated, and so on. Remember, "Computers don't make mistakes, humans do!"

  *stupid mistakes*

Once component placement is done, all of the signals must be hooked up; this is called *routing*. This phase of PCB layout is most easily automated; in fact, a decent "auto-router" can route a moderate-sized PCB overnight. Still, design engineers may have to give special attention to the routing of critical-path traces and to other issues such as placement of test points and signal terminations, which are discussed in Section DFT and Section Zo at DDPPonline.

*routing*