



BY PALLAB CHATTERJEE, CONTRIBUTING TECHNICAL EDITOR

## Reading skills for IC design

**T**here has been great progress in the worldwide education system to increase both the number of people who can read and their level of reading comprehension. Technology and semiconductors make possible a large part of this rise in reading education. This increased awareness of words and reading has changed the IC-tapeout flow. Current EDA tools allow long alphanumeric-string names, which, in turn, provide a great

deal of flexibility in the area of self-documentation to the design engineer. Most designs today have devices and nets with descriptive names. Instead of traditional labels, such as U10 or net162, companies label parts input-buffer8, and signals have labels such as cache2address6.

This increasing number of descriptive terms greatly enhances an engineer's ability to keep track of components and connections at the device, gate, and runtime level. Using such terms simplifies the design-setup time for simulation, eases the interpretation of the output, and allows for automation of verification and regression testing.

A few guidelines exist for using these terms. In most reading books, lower-, mixed-, and upper-case text all mean the same thing: These cases just have different emphasis. Most EDA-design tools are based on parsing a text string as a variable, so specific alphanumeric sequences must match exactly to be the same. For a flow consisting of multiple EDA tools, the best practice requires designers to use only two options. First, differentiate items that should be different as separate signals with distinctly different name strings. For example, don't use "clock, Clock,

and CLOCK"; instead, use "clock, Clock1, CLOCK2" in all instances. Second, all items with similar labels become the same object—unconditionally and globally—at some point in the flow, such as clock, Clock, and CLOCK all becoming "clock."

With this text-parsing requirement, combining different forms of netlists to describe a complete circuit is a challenge unless uniformity exists in the naming conventions. To achieve this uniformity, a number of companies have instituted company-specific but tool-independent naming conventions. These naming conventions include such characters as \*, \_, [, and !; they also include library names and paths in calls to nonlocal modules, internal-versus-pin object-net names, and power-supply names.

With regard to naming conventions, how design engineers elect global signals and global supplies is the most controversial issue. Regular signals appearing in a netlist generally connect either by position in a list or by name at the module or subcircuit where they reside. For example, the netlist for a NAND gate has pins A, B, and OUT. The signal name A is local to just that subcircuit, and a design engineer may reuse the net name

A on the description of another block without conflict.

A global signal, however, does not appear in a pin list where it would get connected and possibly renamed at the next level of the assembly hierarchy. A global signal is a description of a wire connecting to all levels of a design, as the text string that names it determines. An example is GLOBAL=VDD. In this case, every device, module, or block that has a connection to a VDD net connects to the exact same VDD net. This scenario means only one VDD net exists in the design.

A lot of designers freely use the global declaration at the Verilog gate level, without realizing the implications on the device level and for physical-verification analysis. One of my customers had a design with 450 defined global names and only 26 pins. The LVS (layout-versus-schematic) check's interpretation of this situation was that the devices and ports with global names did not have to attach with wires to pass LVS. Most LVS run sets, unfortunately, use the command "connect by text." As a result, if a global signal exists in a netlist that resides in multiple modules or subcircuits, with the same labeling as that in the layout, then the LVS program will think that all of the text labels connect without actual wires to connect them.

As far as the LVS program knows, no errors exist. But electrons can't read, and you can't teach them how. Thus, they need real wires to go through to get from Point A to Point B. Deciding that it is OK to ignore an error message—that is, deciding that a waiver for an error is OK—is an instance of failing to implement the solution.

Remember three rules: Name different items differently, give the same name to items that are similar, and use hard wires if you want to connect two things. **EDN**

Contact me at [pallabc@siliconmap.net](mailto:pallabc@siliconmap.net).