Chapter 1

PLAN, PLAN, PLAN

Introduction to Verification and Verification Plans

1 VERIFICATION

If engineering is about creating and managing change via problem solving, and if this problem solving takes the form of wrapping your arms around a set of problems and dividing them up into smaller problems and solving them, then how does this apply to verification? What problem or set of problems is verification trying to solve?

Verification is the process of confirming that a design entity (gate, sub-block, block, multi-block, chip, system, etc.) functionally performs as intended. Someone (an architect) has defined what is intended, someone else (a designer) has implemented the circuitry to perform what is intended, and now a third someone (a verification engineer) does a check to see if the circuitry is functionally correct. Functional verification is an attempt to get ones arms around the specific problems of seeing if a chip is performing its functionality.

2 GENERAL SPECIFICATIONS

A company has decided to make a certain design. Architects have come up with an approach, and typically with a few of the top designers, they create a specification document or, as in most cases, several documents. Good or bad, these specifications are the starting point for the circuit designers. Armed with the documents, designers interpret and implement
the specifications. Designers divide and conquer; they create the intended functionality. The path is somewhat straightforward, but of course in the real world things often change midstream. Specification changes are common; sometimes the customer decides the world needs a slightly or grossly different set of features than was originally planned. Other times the architecture is faulty and a complete re-work is needed. So the specification typically gets changed; a new revision is created. Functional verification not only has to get its arms around the original set of problems, but also it has to ebb and flow with any changes just like the designers do.

At any rate, the design engineers implement the intended functionality. Verification engineers make sure they did it correctly. Verification engineers are a second-set-of-eyes, so to speak; they enable a two-heads-are-better-than-one approach.

This approach is not a new concept. The software engineering community has been using this kind of checks and balance system principle for years. It is called extreme programming. It is useful and yields good results. This system uses a concurrent track of HDL engineers and verification engineers independently interpreting the specification documents. Mistakes are found: mistakes in the specification, mistakes in the HDL, and mistakes in the verification code.

Sometimes designers verify their own designs and forgo the second-set-of-eyes principle. The problem is that the specification said \(2+2=4\), but they, via a simple human error, read the specification as \(2+2=5\) and implemented the hardware as \(2+2=5\). Then, the self-verifying designers who have the \(2+2=5\) mistake stuck in their brain write a little directed testcase to check that \(2+2=5\), and they accidentally cover up the simple interpretation mistake. With the two-sets-of-eyes approach, there is a much better chance that the testcase will check for \(2+2=4\) and the oversight will be found.

Of course, the mistake might have just as well have been made by a verification engineer and not by the RTL designer, but this is a basic example of the functional specification interpretation problem that verification engineering is trying to solve.

Verification engineers are looking for mistakes, or bugs as we like to call them. Bugs come in many forms. They could be human mistakes like interpretation bugs, or just plain typos, or they could be technical bugs like something in the specification being flat out undoable, or they could be system bugs like the simulator failing or a file being corrupted. So let's look at the nature of where and how bugs occur.