FPGA Simulation

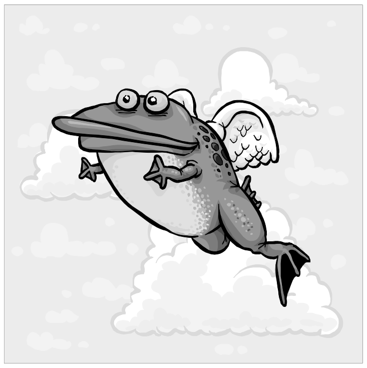
A Complete Step-by-Step Guide

Ray Salemi

1 The Boiled Frog

IN THIS CHAPTER:

- *A Boiled Story*
- *The "Verification Complete" Milestone*
- *Taking it Step by Step*
- *The Seven Steps to the Ultimate Test Bench*



There's a famous story about how best to boil a frog. As the story goes, if you throw the frog into boiling water it will jump right out, but if you put the frog in cool water and heat it slowly, then the frog will stay there until it dies.

Personally, I think that a real frog thrown into a real boiling pot of water would suffer a horrible, watery death, though I agree with the lesson of the story. We are willing to put up with increasing difficulties as long as they increase slowly. Then one day, we look around and realize that we have been cooked.

As FPGA designers, we are the frogs and we are cooked. Our projects usually have a schedule milestone called "In the Lab" and a three-week task called "Lab Bring-up." Three weeks is laughable. We should replace the task "Lab Bring-up" with a task called "Death March," and "Death March" should have no end date.

Once upon a time, getting into the lab really meant that you could expect to ship your FPGA three weeks later. That was back when we were aggregating board logic onto an FPGA whose most complex logic was an address MUX.

We didn't simulate our designs back then because there was little point. The designs were simple and it would have taken longer to create a test bench than to fix the design in the lab. Besides, being in the lab was more fun.

As time went on, however, Moore's law began to turn up the heat. The FPGAs got larger and our lab bring-up time got longer. We began putting more complex logic into designs, but we continued to neglect simulation. Today, we have huge designs and almost no simulation.

This is obscenely expensive. The most obvious cost is debug time. Three weeks in the lab costs about $12,000 per engineer in fully loaded costs. Three months in the lab costs about $50,000 per engineer. That's an incremental difference of $38,000 per engineer over regular debug time.

Then, of course, there is the risk. If we knew that it would take three months to bring up a fully loaded FPGA, we could just write that into our schedule. It would be expensive, but at least it would be predictable. However, that's not the case. Three months of lab time is an average with a high standard deviation; it could easily be longer. The only thing worse than knowing your project will take three months is not knowing how long it will take at all.

A missed schedule is only one cost of neglecting simulation. A product recall is another, much more concrete cost. We know two things about lab debug:

1. We never see the bug that causes a recall in the lab, because if we had seen it we would have fixed it.
2. We can never know whether we've fully tested our RTL.[1]

These two problems make it very likely that a complex, unsimulated, lab-debugged FPGA will see problems in the field.

Finally, there is the human factor. Debug death marches have high — and completely unmeasurable — costs, in stress, missed family events, lack of weekends, burnout, closed market windows, and short-circuited careers.

In short, we are boiled frogs, and it's time to heave our bloated carcasses out of the pot before its too late.

1. "RTL" in this book means synthesizable Verilog or VHDL code.

1.1 A Boiled Story

The scene is a status meeting. A group of engineers sits in a conference room while their engineering director takes the status over a speaker phone that looks like an alien space ship. The engineering lead sits next to the phone:

“So, is the design in the lab yet?” asks the director.

“No. Not yet,” says the lead.

“The schedule says you’re supposed to be in the lab next week. Will you be there?”

“Absolutely.”

A week passes. We are back in the conference room with the engineering lead sitting next to the phone.

“Are you in the lab yet?” asks the director.

“Yup. We got in yesterday.”

“Excellent! Good work everyone. The schedule says we’ll be done with bring-up in three weeks. Let’s hit the milestone and ship!”

As the director hangs up the ten engineers look at each other around the table. It’s true that they are in the lab. The FPGA is on the board on a lab bench. The milestone is complete. Of course, the FPGA doesn’t work. It might as well be a lump of coal. The death march has begun.

Six weeks pass. The engineers have been working round-the-clock shifts to try to debug the FPGA. They’ve been knocking bugs off at the rate of one per four days, but they are three weeks late.

As a result of these failures, the engineering director has flown in to sit in the room during the status meeting. He wants to be there with his engineers as they give their reports because he wants to look into their eyes.

He fixes the lead with a stare and asks, “Are we out of the lab yet?”

“No,” the lead looks down at his notes as he answers.

"What's the problem?"

"We're having trouble with a bug."

"Is this the same one as last week?"

"No, we fixed that one. There's another one."

There's silence.

"When will you be out of the lab?" asks the engineering director quietly.

The lead doesn't like the quiet tone but he answers truthfully, "I don't know."

The lead and the engineering director look at each other. They had this same conversation last week. They'll have this same conversation next week. Odds are they'll be having this conversation for the next two months.

Shift the scene to an upper-level management meeting. They are discussing the engineering director. One VP looks at the other and says, "I don't know about him. He just doesn't seem able to deliver."

Another promising career has been boiled away by the lab.

1.1.1 Root Cause Analysis

What destroyed this project and this engineering director's career? Was it improper design techniques? Perhaps. Was it the tools? Probably not. Was it lack of dedication? No. Was it a lack of proper verification? Surprisingly, no. While this was the immediate cause, it was not the root cause.

The root cause analysis shows that the engineering director's career was rightfully stalled because the problem was his fault.

His schedule destroyed the project. Not the dates, but the milestones. The engineering director had created a schedule with a milestone that said "FPGA in Lab," followed by a "Lab Bring-up" task. He had neglected a critical step.

The schedule was missing the one milestone that would have ensured success: "Verification Complete."

1.2 The “Verification Complete” Milestone

We need to achieve complete functional verification before we get to the lab, and we need a milestone on the schedule called “Verification Complete” to make that happen. What gets measured gets done. If the Director of Engineering in our story had been asking, “Are you done with Verification?,” he’d still have a career.[2]

1.2.1 What do you mean by “Verification Complete?”

Schedule milestones need to be clearly defined. We need to be able to say, for a fact, that the milestone has been completed before we check it off.

Verification has a slippery history in this regard. There are many ways to measure verification. For example, you could measure how many tests you wrote, or how long you ran the simulation. You could even measure whether you’ve run out of time and are forced to go into the lab. None of these are real measures of complete verification.

There are two questions whose answers tell us whether we’ve completed verification:

- Have we simulated the entire design?
- Are we still finding bugs?

If we’ve simulated the entire design and we aren’t finding any bugs then the verification process is complete.

Many engineers tell me that the goal of simulating an entire design is unrealistic. Also, it’s true that you can’t simulate every possible set of inputs to an ALU, and you can’t simulate every memory location.

Though you can’t simulate everything, you can find the important aspects of your design and simulate those. You can simulate all the addressing modes, or commands, or combinations of the two. You can make sure that your router can pass traffic among all its ports.

2. Lest you think this story is exaggerated, I am on the web right now looking at a job opening to replace a Director of Engineering who was done in by verification woes.

This type of simulation is not new. The chip design industry has been doing successful verification in the ASIC world for twenty years. It's possible. It's being done today. Fortunately, as FPGA designers we don't need to go to the lengths of ASIC designers, but we still need to verify all the important functions in our design before we touch it with a logic analyzer.

1.3 Taking it Step by Step

Some books preach radical revolution. They say that if you don't change everything today, you will be left behind in a morass of your own regret. This is not one of those books.

Engineering is a tricky business. It's difficult to get billions of bits to line up and march in order. Frankly, given the complexity of what we do, it's amazing that technology works at all. Despite the complexity, however, we make our projects work over and over again.

We don't do it by radical innovation. Instead we take something that works and add something to it. This way, step by step, we improve our design skills and deliver more-complex projects.

The same holds true of improving verification techniques. While many techniques are available to us, it would be folly to try to implement them all at once. Instead we should adopt a staircase model. We should move from step to step, incrementally adding capability on each project. That way we'll be able to look back down the staircase and be amazed at how far we've come.

1.3.1 The Quest for the Ultimate Test Bench

"If you don't know where you are going, you might wind up someplace else."
– Yogi Berra

Before we start climbing the staircase, we should know what's up there. What can we expect at the top of the verification staircase? In the best of all worlds our test bench will do three things:

- Generate its own stimulus so we don't have to write many tests.
- Check the output of the device under test (DUT) to make sure it is correct.
- Tell us when the DUT has been completely tested.

This is the Ultimate Test Bench, and it is within your grasp because you are holding this book. *FPGA Simulation* will take you, step-by-step, from a simple directed test bench environment to the Ultimate Test Bench.

Each step builds upon the previous steps, so that you don't need to be a radical to get there. You can just add a step or two to each project until you've reached your goal.

What if you decide not to go for the Ultimate Test Bench? What if you get to Step Two and say, "Thanks, Ray. That's good enough for me." Then I say, "Great!" You'll still be going to the lab with more confidence than you have today, and that's the goal of this book.

1.4 The Seven Steps to the Ultimate Test Bench

There are seven steps from flat ground to the top of the verification staircase. Each step builds upon the ones before it and provides a small incremental improvement to your test bench designs. There are examples in all the chapters, and all the example code can be downloaded from www.fpgasimulation.com.

Step Zero is the base of the staircase. It assumes the following:

- You have access to a simulator
- You simulate your code just enough to see if the syntax is correct.

This is all you need to start climbing the steps. From Step Zero, we will walk through seven steps. You can add these steps to your design flow one at a time on a project-by-project basis. Each step gives you a stand-alone improvement over what you are doing today.

Here are the steps:

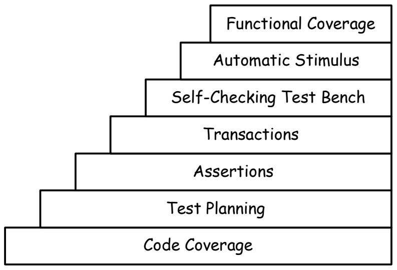


FIGURE 1-1. The Seven Steps to the Ultimate Test Bench

The goal is to add just one step to your next project, get comfortable with it, and add the next one. You can keep doing this until you decide that you are happy with your verification test bench or you reach the top.

Here are the seven steps:

1. **Code Coverage—**In this step you will take advantage of the code coverage features in your simulator to track how much of your RTL has been simulated.
2. **Test Planning—**When you turn on code coverage, you'll see how difficult it can be to achieve complete code coverage with an *ad hoc* testing approach. This step adds a test plan to your process to make verification go faster.
3. **Assertions—**Assertions catch bugs at their source, before they get to the output pins. Study after study has shown that they cut debug time in half. We'll learn how to add assertions easily to our project.
4. **Transaction-Level Simulation—**Transactions make it easy for you to create tests and check results. They encapsulate the complexity of communicating with a device.
5. **Self-Checking Test Bench**—Transactions make it easy to write tests, but we are limited in how many tests we can write if we need to use a manual process to check the results. The Self-Checking Test Bench solves this problem.

6. **Automatic Stimulus—**Once the test bench can check its output, it can start writing its own tests. Automatic Stimulus lets the test bench do the work.
7. **Functional Coverage—**Our test bench can now tell us whether it has tested every function in the design. Functional Coverage goes beyond code coverage to check that we've tested all our interesting data as well as RTL paths.

Your job now is to find where you are on this staircase and then start climbing.

1.5 Summary

In this chapter we looked at the lab debug problems that are raised by FPGAs that grow increasingly more complex. We saw that the solution is to simulate our FPGA designs thoroughly before we go to the lab.

There is no need to implement all the steps of FPGA simulation in a single project. We can take a step-by-step approach that improves our simulation test bench over time. We discussed the seven steps in the verification staircase.

Now its time to take the first step: code coverage.