You've developed a verification environment, hooked up the DUT, written a bunch of tests and alas! Simulations start to fail So just before you dive in, Think Verification's tips department recommends the following:

Prepare a Quick Regression Suite: Yeah, you got your fancy random one test does it all test. That's really wonderful, but more often than not, a set of directed tests, each stimulating a different area of your design may come in handy when you need a status real quick. As a matter of fact, we've published an entire article about creating the right mix of directed and random tests.

Print all useful information in a log-friendly style: I've seen many kinds of messages that people send out to log files. Some are good, some give useless information and some actually cause confusion. So a certain level of messaging uniformity across the elements of the environment can make a big difference when it comes to analyzing a log file and trying to figure out what the heck had happened there

Always start with one packet: First time you're running your SVE (Simulation & Verification Environment), a single packet/transaction will usually catch most of the zero-order bugs. In fact, whenever my design or environment undergoes a major change, I go back to the 1-packet test, just to see that everything's ok.

Save heavy ammo for later: Once again, the principle here is that debug takes time so if you have in advance a sense of what's supposed to happen in a test, and what the expected behavior should be you're saving yourself a lot of debug time. So try to enhance the complexity of your test scenarios gradually by releasing constraints as you go one step at a time.

Be creative: tkdiff or equivalent applications are sometimes not less a debug tool than anything else When a test fails unexpectedly you might want to try diffing your current file version with the last good one (dut or env for that matter). Double check yourself regularly, like someone once said - The question is not whether I am paranoid, but whether I am paranoid enough!

So debug efforts tend to quickly become bottlenecks, and to deal with that you've got to come up with as many debug tools and methods as you can before you instinctively bring on the heavy machinery " the interactive debugger, the waveform viewer, the back-annotated source code, etc. I remember this one time I got called in to help out with a very complex generation issue: *Hey Yaron, can you please come here for a minute and find out why this constraint here is never satisfied by the generator? I got the generation debugger already set up for you* , the engineer said.
Sure bro
, I said,
but just before we do that would you do me a favor and insert a syntax error in this file here?
Are you out of your mind, Yaron?
he frowned at me.
Just do it, please,
I said. You can probably guess the ending - wrong file was loaded