Habits of Deficient ASIC Design

Don Mills

LCDM Engineering

ABSTRACT

This paper will discuss many of my observations of habits that companies and engineers follow that cause ASIC schedule slips and cost overruns. I will be addressing issues such as getting specifications set in place, using the right tools (hard and soft), getting trained, and engineer distractions.

1.0 Introduction

Any engineer who works in the industry for any amount of time encounters a variety of problems that go along with the job. You may or may not have seen the situations discussed in this paper. These are some of the experiences I have dealt with or observed while doing ASIC design over many years.

2.0 You spend more time reading news groups, reading email, and checking your stock accounts than you do designing your ASIC

The advent of the Internet has created awesome capabilities for the engineering world. Data, reports, email, news groups, and web sites are now resources that contribute to successful engineering. Unfortunately, with all this information comes the need to deal with information overload. It can be easily justified in one's mind that spending time keeping up on numerous engineering news groups is part of the job. However, there are just too many groups to keep up with on a regular basis. If you want to stay on top of your current design project, you will not be able to follow many news groups with anything more than a cursory glance.

Email is another form of information diarrhea. It is important to keep up on your email to be aware of all the company reorganizations, and who is now over what department. After a trip away from the plant, what do you spend your first day back doing? Catching up on email. Fifteen years ago what would you have done? You would have filled out an expense report, responded to accumulated messages, and then gotten back to the project. I am not saying that email is bad. Occasionally, you will receive a message that really needs your attention! It just seems that there are lots of extraneous mailings, and if you never read them, you will be no worse off. The tough part is knowing which ones need to be read now, and which ones can be put off indefinitely.

As I work among many companies now, it seems that one of the most favorite pastimes of engineers is to see how the stock market is doing. I fall into this category as well. I admit I like to see how my picks are doing throughout the day. However, there is a difference between checking, and studying out the next move. VHDL does not stand for Verify How Dell Looks. Take a look, and then move on.

3.0 You start the design with the promise that you will soon be given a real spec.

OR

I know what it needs to do. Why do I need to write a spec?

I am sure that this situation has never happened to anyone but me. Yeah, right!!! It is completely amazing to see companies come up with an idea, and then set the schedule before they have a specification of what they are really going to design. Maybe I am naive, but this seems a little backwards to me. We are told to just start with the ideas in hand and the spec will be along soon. Many designs are completed and sent to fab. The engineers then use the time between fab sign off and prototypes received to write the spec for the chip they just completed. No wonder 90% of the ASICs work as designed, but only 50% work in the system.

On the other hand, take the engineer who uses the napkin sketches and the white board drawings as his spec. This situation is similar to the previous one. It creates an environment prone to failure, with interface mismatches caused by failure to formally communicate via the spec.

4.0 You have designers on your team that are not convinced that designing with HDL (either VHDL or Verilog) is efficient, and they are determined to prove their opinion with your design.

This is becoming less of an issue today than it has been in the past few years, but that does not mean it has gone away. There are those designers who still look at the results of synthesis and state, "I could have designed that circuit to be faster/smaller". This is often true. Given enough time, any one of us could take a design, study out optimal alternatives, and work on bettering the design. The key word here is TIME. When designing one, two, four, eight (or more) million gate designs, we don't have the time to tweak every part of the design. Set the constraints, and if the block meets the constraints, then move on . I cringe with terror when I get assigned to a project with an engineer whose first line of circuit design is schematic capture (yes, some of those engineers still exist). It's like trying to swim with an anchor attached. To those designers who still live in the schematic world, I say, "Why stop at the schematic level? If you can make the design better in schematic, then you should be able to further improve it if you go down to the transistor level. Or better still, go straight to the layout level. That's where you will see the real optimization. Sure, build your 2 million gate design at the transistor level."

5.0 Your management hasn't upgraded your workstations in years, and now wants you to design a 2 million gate design with an HP700 or a Sun SPARC10.

This happens way too much. Not only is management asking us to do the impossible with an ancient workstation, but wants it accomplished in the six to nine month schedule he/she

reads in the trade magazines is now the standard. I heard a quote once (DAC 1997 Keynote Address) that said something like..."an average engineer can design pretty good on an up-to-date workstation, but it takes a top-notch engineer to accomplish the same task on an old, slow, low-memory workstation." Think of the brain power lost trying to work the old systems into submission. Time is spent working scripts, breaking designs into smaller blocks, and waiting for runs to finish. That same time would bring much more ROI if the engineer had up-to-date workstations.

6.0 You assume that because it simulates correctly in RTL that the design will work correctly after synthesis.

Please, please, please don't make this assumption! The paper I presented at SNUG99 (co-authored with Cliff Cummings) documented a number of coding styles where the simulation results differ before and after synthesis. Most of these differences can be avoided by understanding the coding constructs that cause the differences, and then following a strict coding standard that avoids these constructs. Additionally, verify your synthesis results. This must be done with a variety of tools. No one tool can perform your full post-synthesis verification. Your post-synthesis verification consists of two parts: timing verification and functional verification. You will begin with static timing analysis, followed by formal verification, augmented with gate-level simulation. I am not convinced yet that formal verification is ready to stand alone as the complete back end functional verification tool. (Many engineers disagree with me.) But I am ready to say that it should be a major part of the back end verification process.

7.0 TRAINING? What's that? If we train our employees, they will just leave the company.

I have been involved with companies that offer outstanding employee training programs. I have also worked with companies in which employees were given such training limitations that they would overrun the training budget just to take the company mandatory training courses consisting of ethics training, security (anti-espionage) training, ESD training, time card training, diversity training, ISO 9000 training, etc. The list goes on and on. Yet if an engineer petitioned to obtain training regarding tools, methodology, or some other project-related training, the answer was always the same: there was no budget for the training, nor could the projects afford the loss of time while the engineer was away. How can some companies have such limited foresight as to not provide a way for engineers to remain at the top of their skill and ability? As to the loss of project development time while an engineer is away on training, if the training is directly job related, the time gained by having a trained engineer will far exceed the time lost for not training the engineer. Those not being trained are left to muddle through the tool documentation, and then use trial and error to come up to speed. They often spend enormous amounts of time working with the tool's AE (AC at Synopsys) trying to figure out all the nuances. They weary the AE with questions and problems that would have been answered in training. Eventually the AE becomes reluctant to help. And then when they really need help, it's not there.

I was at one company that was working to become ISO certified. After attending the ISO training class, I was given a stack of documents 2 feet high (no kidding) that I was required to understand in order to meet the ISO requirements for my job. For labor, I was told that I could charge up to two hours to the ISO training charge number to perform this task. Any additional time required to prepare for the ISO audit was to be on my own time. These kinds of company policies and practices become nothing but distractions for the job of engineering.

Granted, companies are not obligated to train you. Yet there are companies that provide training opportunities, and even some companies that strongly encourage annual employee technical training. On the other hand, some managers really fear that their trained employees will move on. So what keeps an employee at his job? If employees are challenged by their assignments, they are usually happy. If they feel comfortable with their co-workers and their environment, they'll be more likely to stay. A competitive salary is a big factor in employee retention (are you always below the IEEE average?). And don't forget the Golden Handcuffs. Would you want to leave all your stock options behind?

Now, one recommendation: you are in charge of your own career. If you feel that you need training or want it in a specific area, don't wait for the company to come to you. Search out opportunities on your own. Don't sit around and complain about what the company is not doing. Be proactive, not just reactive. When I got out of school and started work at my first engineering job, I was assigned to work with a seasoned and very wise senior engineer, who became my mentor. Among the things he taught me was that I had gone to engineering school "to learn how to learn". The engineer that stops learning new design processes and tools will become obsolete.

8.0 It's only software. Why can't any software engineer design ASICs with HDL?

When a project is short on hardware HDL engineers, managers sometimes try to fill the empty slots with software engineers. There are some software engineers who focused on the hardware side of their field, and who are capable of coding HDL successfully. But generally speaking, the methods used to optimize software are the most inefficient coding styles for HDL coding. Take the following code shown in example 1:

```
if (cnt < 32) then
  cnt = cnt + 1
else
  <perform some function>
```

Example 1: inefficient HDL comparitor

From a software perspective, this approach is acceptable. From a hardware perspective, this creates a much larger comparitor than is required. To code HDL, you've got to think HARDWARE. If you think in software algorithms, you are destined to fail as an HDL designer. A more efficient hardware approach is to compare just the significant bit, as shown in example 2:

```
if (cnt[5] = 1) then
  <perform some function>
else
  cnt = cnt + 1
```

Example 2: efficient HDL comparitor

This coding style must include comments to describe the algorithm being implemented.

9.0 Surely the tools from the vendors have been verified. This bug can't be from a buggy tool. The AE assures us that this is really a feature (at least until the next release).

OR

EDA tools and vendor libraries change/update faster than the time it takes to compile and synthesize your design.

This doesn't happen very much, but occasionally I come across some goofy quirk in an EDA tool. When I approach the AE, I am told that the tool is operating as defined. So, I write scripts and modify my flow to deal with the tool, only to see the quirk go away with the next release.

It has always been a big dilemma when, in the thick of a project, you get a tool update from your EDA vendor, or a library update from your foundry. Do you upgrade so that you are working with the latest greatest tools and libraries, or do you stay with the baseline that you have been working with? You are familiar with the pitfalls and workarounds required with your current environment, so why even consider moving to something else? What if there are significant bug fixes in the new environment? What if there are new bugs that need to be found?

10.0 My tool AE said the problem is in the foundry's library. The foundry's rep said the problem is the tool.

This world of EDA and ASIC is really one big love triangle, the players being the engineers/designers, the foundry, and the tools. All three players must play together for success to be achieved. The foundry uses the tools to generate libraries that are sent to their customer. Their customer (engineers) will use the tools with the libraries to generate a finished product to be sent back to the foundry for realization. It becomes imperative to the success of the design that the foundry and the tool company are in sync with each other. One of the most difficult situations to work through is when you are getting inconsistent results from your foundry and your tool. You call your foundry rep and he claims that they have built their library per the spec for the tool. You then call the tool rep and he figures the foundry didn't follow the spec correctly. Now what do you do?

I had this situation occur to me once just before a trip to DAC. It just so happened that at DAC, both the tool company's and the foundry's booths were right across the aisle from each other. I got my contact from the tool company to sit down face to face with the foundry contact (not a trivial feat), and I proposed a solution. At the time, they both agreed that the solution would work and that it would be implemented. However, no change was implemented. I ended up being the one to make the changes in my design and scripts.

11.0 Design reviews! I don't need no stinking design reviews.

"Design reviews just get in the way of getting the project done. I have to spend two or three days, maybe even a week getting all my material put together and distributed in preparation for each review. And they'll just change the design anyway." Don't forget that you're not designing in a vacuum. Your design has to work in a system with correct interfaces. And part of the review is making sure that you have not misinterpreted the spec.

So what happens when a design review is held? There are two sides to consider here: that of the reviewer and that of the reviewee. As a reviewer, don't focus on or nit pick code that works, but is coded with a different style than what you would have used. The review must focus on functionality, timing, power, area, efficiency, system interface, and continuity with the system. As the reviewee, don't take the review as a personal attack. The review is a design review, not a personal review or a personality review. Too many engineers become way too intimate with the code they write, and become very offended when comments or recommendations are made about their code.

12.0 We were scheduled 9 months for design, 2 months for test bench generation, 2 weeks for scan insertion, and 4 weeks for foundry interface (with a foundry that we have never worked with before).

How real is this? It happens over and over again. You start out with a decent amount of time scheduled for implementation of the design. Test bench generation typically takes at least the same amount of time as design implementation. And the automation of the scan insertion makes it easy to underestimate the time needed for implementation. Additionally, if the design was not built with scan insertion checks as part of the initial design flow, the time needed for scan insertion can be months, as you go back to fix scan bugs, re-simulate, re-synthesize, and re-verify. The complexity of the scan insertion process grows exponentially based on the number of unique clock zones in your ASIC.

Finally, four weeks for foundry interface is very reasonable, if you are familiar with your foundry and have worked with them before. However, if you are dealing with a foundry for the first time, expect a learning curve. There will be many unwritten requirements and expectations. And you'll deal with this situation every time you go to a new foundry.

13.0 Conclusion

Another thing I gained from my mentor came at a time when I was having major problems with my foundry. His response to the situation was a simple question back to me. This question has helped me keep a reasonable perspective on the difficult situations that arise from time to time while getting ASICs out the door. The question was "What is an engineer?" He then answered his own question by stating that an engineer is nothing more than a problem solver. He told me that I had problems to be resolved, and rather than complaining about how poor the library or the tool was, or whatever the current external problem was, that I should work to resolve the situation. I have since come to the realization that every tool, library, or design environment comes with its own set of problems. Very seldom have I ever come across a problem that couldn't be resolved in some way.