Oral History of Philip Raymond “Phil” Moorby

Interviewed by:
Steve Golson

Recorded: April 22, 2013
South Hampton, New Hampshire

Phil Moorby, April 22, 2013

Steve Golson: My name is Steve Golson. Today is Monday, April 22nd, 2013 and we're at the home of Phil Moorby to do his oral history. So Phil, why don't we get started? Where did you grow up?

Phil Moorby: I grew up in Birmingham, England. And that's a pretty industrial city. Went to school there and then left there to go to University in Southampton which is on the south coast of England.

Golson: What was your family like? You said it was an industrial city.

Moorby: Yes pretty much what would be called working class. My father was a fibrous plasterer, worked with his hands in creating molds for very old buildings, manor houses, and churches and so on. And so yes, he's very much working with his hands all the time.

Golson: And siblings? How big was your family?

Moorby: Two sisters, one brother, four of us, all older than myself. I was the baby of the family.

Golson: So when you went off to university was that a real change for you from this working class background and you ended up studying mathematics? How big a change was that for you?

Moorby: Pretty big change, yes. I was eager to get out of the home <*laughs*> quite naturally. And yes, it was a pretty drastic transition from being at home all the time to being at university all the time.

Golson: Did you have a technical interest when you were growing up? Did you have hobbies that foreshadowed your technical work later on?

Moorby: I think you could say that, other than the sports. At age around about 11 or 12 and younger it was all swimming, and then after that I transitioned into basketball so that was the sports side of things but on the academic side, it was all mathematics. I always considered math as my hobby, puzzles, having puzzle books, I always had my head in doing that and reading anything really that I could understand obviously in math.

Golson: When you went off to university then, your initial degree was in mathematics so that was an easy step for you.

Moorby: Except for a period of about I would say two months. In England the final exam system in the grammar school is what's called “A Levels” and they're very intense. That was all mathematics. There were two maths courses and a physics course and I felt like I'd had enough of mathematics. That amount of saturation just doing mathematics so I actually looked to do psychology instead as a reaction until towards the end of the summer I realized that would be a big mistake. I went back to mathematics. Actually just being away from it for two months made me realize actually I did have a deep interest in continuing with mathematics so I went back. It was probably the best choice of my whole life perhaps to continue to do that.

Golson: We all appreciate that you made that choice.

Moorby: <*laughs*>

Golson: At what point did you start studying computer science? You started with pure mathematics and how did that change?

Moorby: Yes it was all pure mathematics all the way through undergraduate. I did one course on computer science which was sufficient to turn anybody off computer science. It was so badly taught. And it was actually details about the IBM MVS operating system. You know, go through all the details of all the zeroes and ones you have to set to create certain options. It was a terrible course, nothing to do with real computer science. So it wasn't until after the undergraduate that I actually realized I was able to do some computer programming and realized that I could stay with that almost day and night. You know those programmers that stay up all night drinking coffee?

Golson: So that was as an undergrad?

Moorby: That was an undergraduate course, yes. It was a side shoot of mathematics, yes.

Golson: Aside from the MVS job control details, what [programming] languages were you using?

Moorby: FORTRAN.

Golson: It was all FORTRAN.

Moorby: There was a PDP-11 in the department that again it was so heavily used the best time to use it was at night time so you usually get into that working all night long kind of habit, but that was just working on a PDP-11 in FORTRAN.

Golson: Was that class work you were doing or was it just fun?

Moorby: A bit of a mix but there was a course work to do that. It was just enough to tell me to have a future in mathematics you either you have probably one of the only ways to continue with mathematics to become a professor, and teach it. And I knew that that would be too much for me. I wouldn't be able to stay with it because it's a very introverted, insular lifestyle to be able to stay with that and to become—you had to become the best of the best for people to want you to actually do that for a living. So I thought computer science I knew I could stay with it day and night so it was actually a very good choice. And I didn't know that much about computer science so I did a master’s at Manchester University. It was considered a conversion course into computer science.

Golson: So conversion as converting from…?

Moorby: Mathematics. Outside of computer science into computer science. So there were six months of courses and six months of writing a dissertation.

Golson: At this point how much hardware experience did you have? Or was this all very much computer science, writing software?

Moorby: The Manchester course, the master’s was everything. I mean Manchester University had a very good reputation for building computers so in that department they did everything with respect to building computers, from the actual building all the hardware modules and putting the whole thing together and of course all the software and operating systems for it. They drove pretty much the British computer industry so there was one main company that Britain had kind of fostered, a company called ICL [International Computers Limited] not very well known outside of Britain. And many of those ICL designs that became industrialized came out of Manchester University. So in the department I worked in they developed a thing called MU5. It goes back to some of the original computers that were ever built. So that was a fun place to work in the center of excellence.

Golson: While you were there, were you aware of this past history of Manchester?

Moorby: A little bit. I probably found out a lot more in later years—reading books on Alan Turing, because he was there for a short time.

Golson: What was the first simulator that you worked on? Was that at Manchester?

Moorby: Yes, maybe I would call it a simulator. I did in the master's course, the six months of doing some research and writing the report, I chose a project that spanned both hardware and software because I couldn't make a decision which way to go. “Okay,” I thought, “okay, well I like hard problems, big problems,” so I did both. And that was to build a piece of hardware that would test the MU5 little hardware modules. So you had to build an interface, it was actually an interface to a PDP-8, which was close to being on its knees in terms of it actually working. And the frustrating part with that, after working day and night and trying to get this hardware to work, [was] realizing that the reason it didn't work is that any kind of power glitch in the building would cause it to stop working. So, [I] brought the professor in one day—"Look, the problem is, why it doesn't work is, go turn the light on." Turn the light on. "You see? It stops working." And that would cause a glitch through the PDP-8. Anyway. So that was the hardware experience I had.

And the software side was to build a test generator so I dived into what's called the D-Algorithm which is the classic algorithm for test generation, wrote a program for that to automatically generate test patterns for these. They were fairly simple—well, [by] today's standard of course they were very simple logic that was on the module cards that went into the MU5.

Golson: So you had this very early experience with debugging at both the hardware and the software level.

Moorby: Absolutely, yes. <*laughs*> I guess that's appreciation of making sure that there aren't bugs in the software. It probably was started there.

Golson: Yes, and back to something you mentioned earlier about in your boyhood of enjoying puzzles and solving a puzzle.

Moorby: Oh yes, I always enjoyed the hardest of the puzzles.

Golson: So after Manchester you ended up on the HILO team. Walk us through how did that happen that you came to join the HILO team.

Moorby: Well I enjoyed that work. It wasn't called EDA [Electronic Design Automation] then of course. They used the term “Computer Aided Design” or “CAD” software. I knew I had a strong interest in that so I was looking around all of Britain to see where I could go. I had my eye on doing a PhD.

Golson: Back up a moment. On your master’s work at Manchester, how were you doing your development work for your software? What sort of machine were you running on? Do you remember?

Moorby: This PDP-8.

Golson: On the PDP-8.

Moorby: Yes.

Golson: That you were interfacing with.

Moorby: With paper tape.

Golson: Oh my.

Moorby: Oh yeah, it was awful. Well looking back. I mean then you didn't know better. You just worked with whatever you had. Looking back you think just how primitive it was. You either could handle punch cards or paper tape and both had their pros and cons. Paper tape of course you didn't have individual cards. It was a continuous piece of paper tape. And if ever it got nicked or cut in any way that was it. Done. You wouldn't be able to read it back in even. It was extremely primitive. And quite often you would lose your program so you'd have to type it all in again. You learned methods of backup to safeguard the work that you had done. You learned a lot of tricks very quickly. But I think I wrote in the PDP-8 assembler language pretty much. There was no compiler on that machine. I was thinking back how to write the D-Algorithm in that. I have no idea how I achieved that.

Golson: Dare I ask how much memory that PDP-8 had?

Moorby: I couldn't even begin to think. Probably four kilobytes. <*laughs*>

Golson: That's what I'm thinking too. Moving ahead—you wanted to continue similar work, hardware, software simulation, and so you were looking about for where you might go.

Moorby: I looked around. There were very few places in Britain I could do the work. There was Edinburgh was a good center of excellence, and Brunel University which is just outside London. And the professor there, Gerry Musgrave, I had an interview with him. I said I wrote the D-Algorithm and got that working, he said "When do you want to start?" <*laughs*> So I started a PhD there

And the person I was working with on a day-to-day basis was Peter Flake and he had done several years of good research and development of HILO-1. Actually they were in the middle of developing HILO-1 at that point in time. And I joined to do the PhD on timing analysis. They knew that this was one of the big serious problems that verification of hardware required so I jumped into dynamic timing analysis that would go with the simulator. They had a contract with the Ministry of Defence in England through a company called Smiths Industries. So they had that some money coming in for doing the HILO-1 project.

Golson: Had you been familiar with HILO before you joined there?

Moorby: No, I hadn't no.

Golson: The idea of dynamic timing, was that something you had any background on? Or you could see the benefit of it from your previous work, perhaps?

Moorby: Doing the test generator, an additional part of that [is] to do what almost could be considered a simulator. In other words, once you generated a test you then have to see how many faults you would be covering. So you take the patterns and then forward propagate through the circuitry to work out the logic values and the effects of the faults. So that was a kind of a simulator but didn't have any timing on it. So when I got to Brunel University I very quickly dived on programming a simulator. I ended up writing quite a few of them. The dynamic timing analysis was something in addition to that, where you couldn't just develop a simulator to do that work. You actually had to do what was considered analysis work that would be not time-based with an event wheel, but you'd actually be having to analyze the circuitry like today in static timing analysis. It's a lot more towards formal analysis-like work as opposed to an event-driven logic simulator. But part of that work is, I programmed a logic simulator fairly quickly just to test the ideas.

Golson: Were you able to leverage the earlier HILO work? You say you had to write a logic simulator. I mean they already had something like that.

Moorby: In HILO-1, yes. But that was written in assembly language on an ICL machine for the Ministry of Defence. And they realized that it wasn't very portable so they wanted to move on to the HILO-2 project and redo it. And it was HILO-2 that I pretty much had joined to start— [I] had my first programming experience on HILO-2. Actually I never touched HILO-1.

Golson: So what was the development environment then? If they were trying to get away from ICL assembler, what was the goal for HILO-2? How was it to be implemented?

Moorby: To be portable mainly so they could move it around from machine to machine. The Ministry of Defence finally woke up that there were a large number of computers in the world. <*laughs*> They were not all the British ICL machines. So they absolutely needed to get off that machine and make it portable and also to have the code run on a large array of different computers. And so that's when, not due to me but Peter Flake, mainly, I believe that he researched the best language of the day. Thank goodness it was not FORTRAN <*laughs*> but it was a language called BCPL [Basic Combined Programming Language] which if you know—one of them, there were several others but one of the inspirations of the C language. Fairly low level. It was much, much better than assembler language but nowhere near as powerful as C. But it was very, very portable. BCPL was defined to be a 32-bit language where everything was a 32-bit integer. But there was a lot of developers had developed interpreters for it so we actually put the interpreter on a PDP-11 machine which is a 16-bit machine in order to develop all the software for it. And eventually we got onto a real 32-bit machine so all the code would run very easily on that machine and be compiled so it would run very fast.

Golson: So you're starting on a PDP-11 with an interpreter pretending to be a 32-bit machine.

Moorby: That's right.

Golson: Eventually what was the 32-bit machine that you ported to?

Moorby: The university at that time had bought a Honeywell Multics machine so that was a true 32-bit machine and there happened to be a BCPL compiler for it. So when we got that working that was like night and day, the speedup was probably the order of a couple of thousand times.

Golson: On the Multics machine, is it starting to become what we would perhaps recognize as software development today? Are you using a glass terminal? Or are you still—

Moorby: Not yet, still a terminal so there's all—

Golson: Paper.

Moorby: Paper. At least you had a typewriter so you could type your program and then run it and then print it out on the paper. Didn't have access to it all the time, I mean you didn't have free access. You could have perhaps an hour or two a day in that mode. It was a big advance over having to deal with punch cards. So with punch cards of course you have to keep everything in order. If they get dropped your program is kind of thrown away so it was far advanced from that but still on paper so still quite limited in being able to debug programs and write them and run many times a day. You had to keep on working through bugs. And it wasn't until the department had bought a VAX machine from DEC [Digital Equipment Corporation] that we actually got a real screen with a decent editor so you could actually edit and debug much, much faster.

Golson: So working on Multics, if you only had access to the computer for an hour or two a day, how did that affect your development work? Did you spend a lot of thinking time or…?

Moorby: Yeah, the rest of the hours of the day you'd go through the code by hand instead so we became pretty good compiler writers that way. <*laughs*> That was a big leap in productivity from just the year before where you would be lucky to get like two runs a day. In fact you'd have to run down to the computer center, get your previous run out. Go through the code. See what happened—usually it had a compiler error so you'd work on the next bug in your code. It usually meant having to retype a few of the punch cards, and put it back in again so you would hope that you would get two runs in a day. So from that backdrop it was a vast increase in productivity if you could interactively work on your code for a couple of hours in a day. And the rest of the time of course you would be writing more code and just checking by hand through the code.

Golson: What year was this? When did you move to Brunel?

Moorby: 1975.

Golson: ‘75.

Moorby: Yes. Yes, '75. This would have been '76, '77.

Golson: I understand you uncovered quite a problem with the Multics BCPL compiler. Do you want to tell us that story?

Moorby: Sure. So Peter Flake and myself, whenever we look back in those days this one always comes up as a really good story to tell over a drink. So you got this new machine coming in. There was a BCPL compiler that somebody wrote, obviously wrote it really well, obviously a very good compiler developer. And we were getting some libraries working so that our codes would port to the Multics machine and it started to turn out—when you're doing that, obviously you've got quite a few of your own bugs so you're working through things. But it was this weird situation where all of a sudden the program wouldn't work in the same way that you were expecting it to and it wasn't every time. I think it was something like if I remember, one in five or one in ten kind of rate, but when you have what is generally called an intermittent fault like that, which is usually hardware going wrong because of some of the electronics are not very good. But in software you don't normally expect that. And it was a weird bug that took us forever because most of the time it never manifested itself.

So we had cleaned up all the other bugs, we had developed the library we needed, we had ported all the code, and we started to realize that—just a few times a day of course, you get to compile and run—that there was this weird pattern that would be coming out but every now and again the program—I can't remember whether it crashed or did something weird. I think it was a combination of the two. And then you recompile and it would be fine again. So if you ever experienced that with either a piece of hardware or a piece of software when you have run it so many times you start developing a statistical pattern as to “Oh it's about every one in five” or something like that. So we had to get to the innards of the compiler because we [wanted to] track it down. It's got to be some kind of a compiler bug or even a hardware fault which was very unlikely because all the hardware was new.

So we examined the compiler and I think it was Peter who had encountered a situation where we froze one of the compilation version and we had—I can't remember how we froze it now—but we were able to analyze a version of the compilation output that caused the crash. So we said okay, let's start going through this code carefully. I mean there were no real debugging tools so to speak unlike today with the very detailed debuggers you have. So all you could do is to put print statements in. Of course to put print statements in you had to recompile which meant the bug came and went intermittently. We were able to freeze it and we found that there was some sort of a problem with I believe it was an instruction that was being generated that was an offset to a global variable and that offset had to be packed into the instruction in a certain way and the calculation for that was wrong, so that it would depend on where the function sat in memory. So what we finally got to realize is that when we looked at the BCPL spec it actually said that it was undefined the order in which the functions would be laid down in memory and the compiler writer thought of a very clever idea to use a random number generator to define the order of the functions. I mean not only a pseudo random number but a random number based on the time of day. So every time you ran, the order of the functions would be in a different order and the offset to the global would sometimes throw out this problem. So we spent probably weeks on trying to track this problem down, so the moral of the story is how somebody who is obviously very good at his work, obviously a really good compiler developer, how he read the spec as something said undefined and that translated into a true random number generator to define the order of the functions.

Golson: That must have been very frustrating.

Moorby: Is that intelligence or what? *<laughs>* So those sort of stories Peter and I look back and just laugh at it.

Golson: You can laugh at it but then you can also see how that affects your future work.

Moorby: Oh yes, yes.

Golson: In ease of debugging and—

Moorby: Absolutely. It was a first-class experience of telling you how long some bugs can take to find and if you just think straight in the design phase, to get it right in the first place, or put together test cases to make sure you that you don't have those bugs, it teaches you just how much time you potentially save yourself later.

Golson: I have a quote attributed to you: "People value your product based on the quality of the bugs they find." I think this is a good example. Do you want to elaborate on that?

Moorby: Well I guess it was a reaction to people coming back, customers would say that Verilog was very, very bug free and say "Well, how did you do it?" Just put together lots and lots of really good test cases. And if you do that, the bugs that they do eventually find—which they always will—are what I started almost jokingly said "They are good bugs to have." Or they are okay bugs to have. The bad bugs are those where the customer will say… I mean the customers are very, very smart. They encounter a bug which—remember, this is the center of their frustration to have a bug because it's time wasting for them. If they can see that the bug was due to you not simply testing something when you should have tested, they get even more frustrated and they get angry with you. But if there's a bug that they almost sort of take pleasure over somewhat, like “I did this, I did this, and I twisted around like this and this didn't work.” “Oh, I never thought of that one before.” *<laughs>* And so that in the good extreme situation of that you can get away with saying “Oh yeah? What were you thinking? Why did you try that?” So you can say okay that's a good bug. And the customers almost feel pleased with himself that he found this bug for you. But the other bugs where they should never have existed because you obviously didn't generate any particular test for it for that particular area. They'd get very frustrated with that and then they would get angry with it.

Golson: Did you interface with any of the HILO customers when you were on the HILO team?

Moorby: We struggled to have customers in the early days. The main one was obviously Prabhu Goel, he was the main character who came in. He bought HILO-2 for Wang in Massachusetts and so he traveled the world taking simulators apart and working with all the teams around the world that had simulators. Because he had the job at Wang to go out and buy the best simulator. And so he came and he grilled us and tested and everything for about two weeks at Brunel University and that's how I got to know Prabhu Goel. He found lots of bugs for us too. <*laughs*>

Golson: Good bugs I hope.

Moorby: That's not what he said. <*laughs*>

Golson: <*laughs*>

Moorby: It was a mix.

Golson: I see.

Moorby: HILO-2, we obviously hadn't finished developing the simulator. It was mainly a logic simulator and a fault simulator and a test generator.

Golson: So the dynamic timing work that you had done? Did that ever—

Moorby: No, the dynamic timing never went in. That was my PhD research on the side. It was not due to go into—it never went into HILO-2.

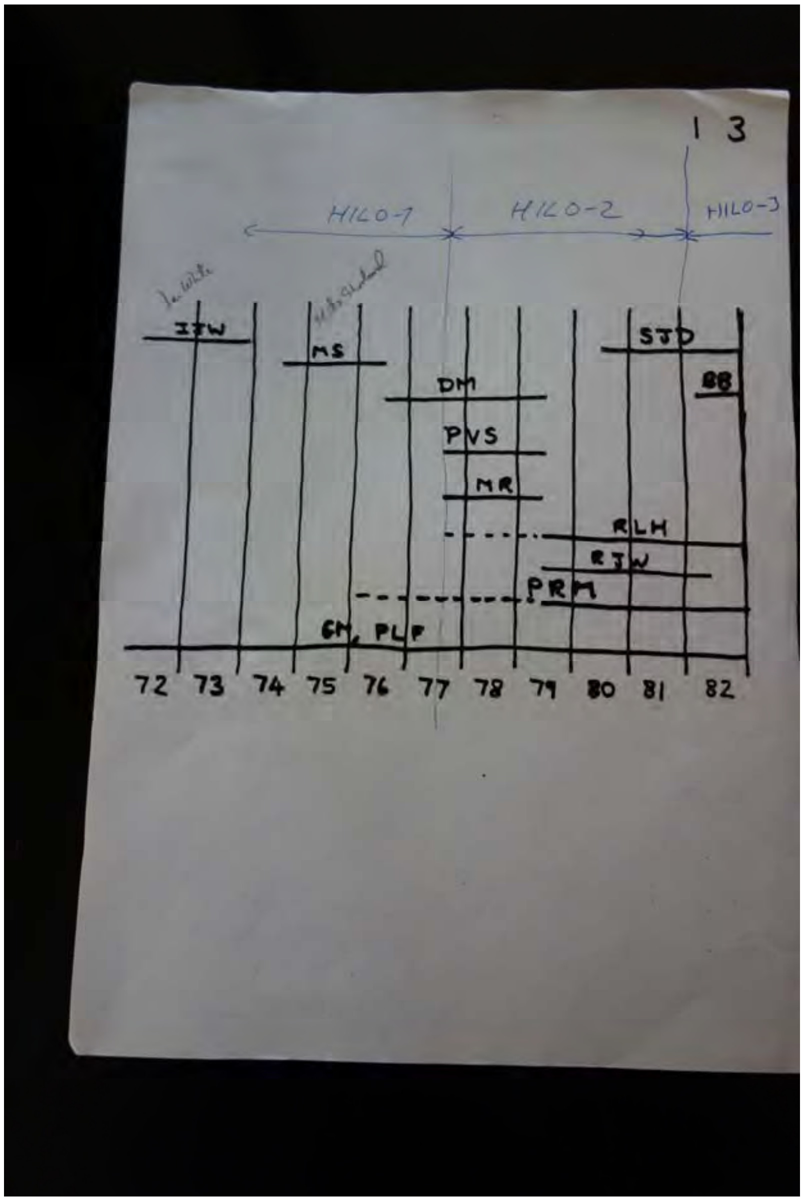
Golson: How long were you working on HILO-2? How long were you at Brunel?

Moorby: Can we get that chart out? ...

Gardner Hendrie: Why don't you ask some questions about it?

Golson: So my question was, how long did you work on the HILO project?

Moorby: Yes, see I'm on that horizontal line <*referencing image below*> with PRM so the dotted lines were my PhD research time. So I joined the HILO-2 team full time as a programmer right in the middle of '79.



Golson: And prior to that you were working on your PhD so you were sort of an adjunct to the team?

Moorby: Yes, I was working on a daily basis mainly with Peter Flake, writing simulators and doing the timing analysis on the side.

Golson: I see.

Moorby: As you see there were a number of people.

Golson: Can you walk through the names?

Moorby: I'll do my best to try to remember the names.

Golson: Certainly.

Moorby: Some of the original developers at HILO-1 way back. One guy called Ian White who started it. And then there was a Mike Shorland, Dave Martland. I think that was. <*referring to PVS*> I can't remember his last name, Peter… MR I can't remember. Anyway so when we really got going there was a team of five programmers, and that SJD is Simon Davidmann.

Golson: I see. And Peter Flake at the bottom.

Moorby: Peter Flake on the bottom. Gerry Musgrave, the professor. Myself. Richard [Wilson] and Robert [Harris]. So there’s five of us total. We got going in the middle of ’80, 1980. A good set of people developing HILO-2. So Prabhu would have turned up somewhere around ’81, I would say. So when he came for his two weeks to grill us and find all of our bugs. But he bought HILO-2 for Wang.

Golson: So what other products was he looking at? Who were your competitors at that point?

Moorby: TEGAS was the main other one. I can’t remember there might have been one or two others.

Golson: Endot, perhaps?

Moorby: Possibly. I hadn’t heard of them at that point in time. They came later. TEGAS was the very well-known simulator that he certainly would have focused on and centered on and that was probably the obvious choice. TEGAS was being used all over the world. But we knew it. But it was extremely slow. So we really had the advantage of being much, much faster. And Prabhu got to know that. He could see clearly how much faster we were, but he needed to see the maturity of the code to see whether they could really work with it. And I think we just passed the test. He commented, “There’s still many more bugs you must keep on trying to find and make sure the code becomes very mature.”

Golson: You said you came on fulltime at what point—1980 roughly.

Moorby: It depends on what you mean by—fulltime on the HILO project, it was the middle of ’79.

Golson: So at that point you stopped your PhD?

Moorby: Yes. I did. I did not finish it. So I did all of the good research. Everybody said that I had done enough research. I worked out some mathematics for doing the dynamic timing analysis. But then it comes to writing a whole thesis. And then, of course, I preferred programming rather than writing a thesis. So the thesis got put on the shelf. And there were obviously not enough time because I had a full time programming job. I guess one would argue where HILO led to was okay too. It would have been nice to have finished the PhD, but time was just not there.

Golson: As a fulltime programmer, in 1980 or so, at this point you’re on a VAX by then?

Moorby: Around about then we were still on the PDP-11 interpreted, and turning to the Multics, the Honeywell Multics. I think the VAX came in much later, maybe even all the way up to ’82.

Golson: The project moved from Brunel to Cirrus [Computers Ltd]. Were you still there? How did that work?

Moorby: No, we all stayed at Brunel University but Gerry Musgrave did the deal to join Cirrus but we built an office on the campus.

Golson: So you were physically at Brunel.

Moorby: We were physically at Brunel. And yes, it was actually a pretty good office. So we transitioned from being students—you can imagine being a student on a university campus—to becoming professionals, you know, you’re wearing a tie every day coming into the office, with still the students milling around, looking “Who are they? What are they wearing ties for?” So that was good. And Cirrus Computers, their main office was about 100 miles away. They were on the south coast. So we would go down there once every other week or so. But that was when it started to transition into the HILO-3—this would get into ’82 by now.

Golson: How did it change your working life? Changing from Brunel to Cirrus, now you’re wearing a tie, did it change the direction of the project?

Moorby: Not so much. There was interesting political situations going on because way back the company called GenRad started to get interested in HILO-2. And GenRad had owned I think something of the order of 25 percent of Cirrus. So they had an interest in the Cirrus company because they had a—there’s a number of those people came out of, I believe, ICL and other good computer companies in England and were experts at test generation—GenRad is a test company. So they were very interested in that company. In the meantime, that company didn’t have anything to sell. When they came and looked at us especially that we was beginning to make a couple of sales, especially with Prabhu in Wang, when GenRad heard that, some of the business developers and sales guys within GenRad said “Oh, give us HILO-2, we can sell that now.” So GenRad politically started to favor HILO-2. But whereas Cirrus wanted to transition to a new project that became HILO-3. So they wanted to change a lot of things, redevelop it, make it more central to being a test generator. Whereas, HILO-2 is more of a logic simulator and a fault simulator. They needed that ability as well, but it was more centered on a true test generator and that was what GenRad was really wanting. I’m trying to think of the year when Peter actually left because he didn’t particularly like where that was all going. None of us did, really, except that we started to interact with GenRad a fair amount and actually enjoying people in the US wanting to sell our software. And it politically became a bit of an issue of, well, why do we want to redo all of this into HILO-3. So between ourselves, GenRad and Cirrus there was a bit of a political triangle started to form. I had actually transitioned and—I was doing what I had done for many, many times, and I think I was pretty good at doing, and that’s the fault simulator within HILO-3. So I was working fulltime on that.

Golson: Let’s walk through the distinctions between HILO-1, HILO-2, HILO-3. Did the language change? Or is it just the implementation? You started with assembler on the ICL and now it’s implemented in BCPL, and then HILO-3 was—how would it be done?

Moorby: I think it’s fair to say that the gate level—both the syntax form of the netlist and the simulator—was pretty common all the way through. I didn’t implement the HILO-1 simulator, but I did do the HILO-2 and HILO-3. So things like the strength logic, that was used for the switch level, all really started happening at HILO-2. Above the gate level, HILO-1 only had what was thought of as—there’s many terms that have been used for it—a macro level. Basically, it went up to flip-flops and little adders, tiny little components, by today’s standard it would be tiny little components. But it didn’t have a language to express that in. It would just be hard-coded like a D-type flip-flop or four-bit adder, all sorts of things like that. And you have a whole library of them and you’d just put them into the netlist. And all of those components were hand-coded in assembler, on this machine. HILO-1 was a very, very, very fast simulator. I mean incredibly fast. It was highly tuned assembly level.

Golson: A gate-level simulator.

Moorby: Pretty much. Yes. It wouldn’t have been considered a switch-level, but a gate-level flip-flop kind of level. So maybe that was an inspiration to me, all the way through to say okay—in fact, maybe it was there, that when it came to Verilog, I wanted to—I always had this wanting—I think throughout this whole time you’re always trying to work out and figure out how to make the simulator go faster. Although, the dominant big problem was more focused in the fault simulator, not the logic simulator. And the big problem for fault simulator [was] how do you deal with—if you have say, 10,000 gates, you would typically have a couple of thousand faults to simulate. And the problem there is how do you get through that amount of analysis and that simulation to get to do what’s called the fault coverage? And that was the big problem. And for the longest time, that held the development of the basic simulator back because if you could push these problems aside like the fault simulator, you can really focus on “Okay, all I need to do is straightforward logic simulation. Now, how do I make that go really, really fast?” Which is what HILO-1 had. The main simulator in there was just the logic simulator written in assembler so they really got carried away making it very, very fast.

Golson: Does any of that still exist? I mean code…

Moorby: I have no idea. I guess you’re going to have to find out if GenRad is still around and knock on their door.

Golson: So GenRad owned part of Cirrus but then eventually the team was working directly for GenRad. Did they take over all of Cirrus?

Moorby: They did. But I left in ’83, so the year after this timeline <*referencing image above*>. I actually never worked for GenRad. It was to the nearest month that I had left and GenRad had bought Cirrus. They owned 25 percent but they just owned that portion of stock in the company. But GenRad was the only set of people selling HILO-2. HILO-3 struggled for the longest time to come out to be a product and GenRad was getting very frustrated with that. But the sales guys were happy because they had started to sell HILO-2 and wanted to stay with that.

Golson: What was the big change to HILO-3? Compared to HILO-2, what was changing?

Moorby: The language was the same. There’s a couple of tweaks here and there. Not too many, but it was all focused on test generation. So the big marketing angle for HILO-3, “This is the complete test generator, and oh by the way it has a fault simulator as well on the back.” But the transition—when I left in ’83—actually Peter Flake came back, it was quite a coincidence, he came back the month I left. So he took over what then had become a bit of a political issue because there were lots of political infighting, the reasons why HILO-3 hadn’t been finished. And some of the things were just simply not working very well. Test generation is a very big problem area to work in. And really, the problem never got solved except it all became scan-based design. That really took over the whole problem. And what they were trying to do in HILO-3 was not scan at all. And they brought in a lot of artificial intelligence ideas of how to do test generation. And it was just way over the top of spec’ing a project and never, ever being able to complete it. That was another big lesson there.

Golson: Yes. So you decided to leave the HILO team. What prompted you to leave?

Moorby: So we got to know Prabhu Goel back there, where did we say, around about '80.

Golson: Yes, '80, '81.

Moorby: I presented a fault simulation paper in '83 at a conference in the US. And Prabhu was there. And he said, "Do you want to come over to the US and join a startup?" I didn't say yes immediately but I only took about a week.

Golson: Because he had already started Gateway at that point, is that right?

Moorby: Yes.

Golson: Did you consider going anywhere else? I'm trying to understand the decision to leave—did you decide to leave HILO and then look around? Or Prabhu came to you?

Moorby: No. In fact, I think throughout my career I haven't tended to say "I can't put up with this any longer, I've got to leave," and then gone and interview a number of different places and then make a choice. It tended to always just flow from one to another. I think at that time when Prabhu gave me the offer, I mean it just feels right. Okay, yes, that's an obvious thing, the next step, just go and do it.

Golson: What did he offer to you from a work standpoint, did he say "You are going to work on…"

Moorby: No. From the time he was with us for two weeks in '80 we had got to know each other, and Peter, the three of us, we got to know each other really well. I think a tremendous amount of respect was built up of his ability and our ability. And I remember I said, "Well I'll come, yes." It was almost a mode of "I'll work on anything because I know it's going to be good." And obviously part of the same kind of work but whether it was going to be test generation or something new—actually he had been working with Chi-Lai Huang for a while but couldn't say anything. He wouldn't even tell me who it was because he hadn't finalized his visa, so that had to be kind of quiet. And he was working for a company. He was doing a PhD and I think he was working for someone. Yes, that's right it was working at Wang.

Golson: So Chi-Lai was working at Wang.

Moorby: Chi-Lai was working over there I remember, but he was the one who was given the job to work on HILO-2. So he was under Prabhu at Wang. And he had some sort of temporary visa. So I think Chi-Lai wanted to join Prabhu but they were keeping it quiet because they had to get the visa through first. And

Chi-Lai and I pretty much joined Prabhu at the same time. He was sorting his visa out, sorting my visa out to go to the US.

Golson: You come to the US. You sit down with Prabhu. Here’s this new startup which might be called Gateway. I don’t know if it had changed names.

Moorby: Yeah, it was called something else at that time. But that’s another story. We may get into that.

Golson: That’s right.

Moorby: I came over for one month, the Christmastime of ’83. And in that one month I had worked day and night pretty much with Chi-Lai and myself, and we put the language together of where we wanted to go with it. Chi-Lai was working on PhD for doing synthesis. He had his own language for that. So I came with the HILO experience and said okay—and many years of what I wanted to do next, how to do it right and all of that business. And in a period of one month pretty much put the language together.

Golson: The decision to create a new language—how did that decision come about?

Moorby: When I came over Prabhu thought that he with Barry Rosales had the test products covered. And, in fact, Chi-Lai in his spare time was actually working on the fault simulator for that. When I came in he said “Okay let’s…”—his grand vision was to do synthesis.

Golson: Prabhu’s grand vision…

Moorby: …vision because Chi-Lai had done a PhD in synthesis. And I said, “Well you need a new language for that, and a new simulator; I can do that.” So that’s why that was almost an instant decision, the first day I flew over for that one month period. I said, “Okay, let’s put the language together.” So Chi-Lai and I—he said how the language needed to be for synthesis. I said how it was to be [for]—the thing about developing a language is that you’ve got to consider the tools that it’s going to be for. And in those days, the primary focus was actually fault simulation, and how to do fault simulation and test generation.

And logic simulation was very secondary because customers—because logic simulator was so much slower than hardware. Hardware designers had the hardest time to use it. And they said, “We’ll just do breadboarding.” So the whole business of simulation for verification had not happened at that point. The market demand for that was very, very small. It was all about fault simulation, test generation, and then it was—well, synthesis was this future vision that if we could do synthesis how great it would be, but people didn’t know how to do it at that point in time. Timing analysis was a big thing, obviously. The formal analysis hadn’t got going, just similar to synthesis.

So the drivers of the language, up until that time, was really all mainly about how do you do fault simulation. And so, in fact, many ideas actually put into Verilog—that we can get on to—was really how do you also do fault simulation in that language. But the new influence was synthesis because Prabhu and Chi-Lai and I said “Well, yes, let’s design the language so that we know that we’ll be able to do synthesis as well.” So that influenced the language.

Golson: Did that surprise you that they were working on synthesis when you walked in and he says, “Oh by the way, here’s what we’re working on?”

Moorby: No. They were actually software developed and some projects that went on back in Brunel University. Synthesis wasn’t new. It was just very, very difficult to do and it hadn’t become—it was very much in the academic world. It hadn’t become—very few companies had made any kind of a—in fact, I don’t think there were any commercial outfits that were making any money on synthesis. I think there were some US universities had some synthesis programs. Brunel actually had developed something. I wasn’t part of that project so I have no idea how well it worked. But in the university setting with the benefit of being students or surrounded by students in a university, you had this mentality of having lectures, continuing to read academic papers and staying up on all of the academic things. And synthesis and formal analysis was all part of that.

Golson: This is interesting that your office there is at Brunel University. Even though you’ve got your tie on which makes you different from the other students.

Moorby: Yes, that started to change.

Golson: It started to change. <*laughter*> But you found yourself aware of the other research that was going on and lectures…

Moorby: Way back when joining—I joined Brunel University at the end of ’75. So you’re in that environment, where it was largely academic. You’re reading academic papers most of the time, talking to other students. So the pressure of making money just wasn’t there. But that’s when the tide came—when the tide goes out, okay, we’ve got to make some money.

Golson: It seems like you were the right person in exactly the right time there. Had you been thinking about well, gee, if I was given a clean slate, here’s how I would do a simulator and here’s how I would do a language. Had this been in your mind?

Moorby: Oh yes. So I think what went into HILO-2—and you’ve looked at all of those various constructs, which look a little bit Verilog but not quite. That was all formed largely between lots and lots of discussions with Peter and myself through all of these years that went into HILO-2. So towards the end of that, of course, the language is pretty much frozen, just having to get the product to be into a matured state. I would say from that point on all the way through HILO-3—because remember with the political situation with Cirrus nothing happened with the language. It was definitely at that mode that we don’t need a new language. In fact, if we do anything it’s going to be for test generation, not for the simulator. But the ideas, the development of how to do it better, of course, that never stops. It’s always going on. So way back to ’80, the thought processes were chugging away to how should this be better? How can I do this? How can I do that? All the way in the context of when you thought of those ideas—and, of course, the biggest thing that we all knew where it needed to go was what generally would be called a procedural-like language. And, of course, they had a strong influence from C. I mean, that’s how you write programs or write software. But, of course, we were modeling hardware. So how did these two come together? We knew that to make the programming or the modeling ability easier we needed a procedural language. But how do you do that? And how do you do fault simulation for all of that? How to do it in the fault simulator is probably the central problem I was mainly focused on. Because it was the fault simulator that was making the money back then, not the straightforward simulators.

Golson: In the HILO-2 days.

Moorby: Yes. And HILO-3. It wasn’t until the nineties when it switched. The test products stopped making money and it was all about verification and synthesis.

Golson: During the HILO days, you’re trying to make a simulator run fast. Were you running into limitations of the language itself?

Moorby: All of the time.

Golson: And saying, “Oh, I wish it had been done this way,” and that would allow me to run the simulator faster?

Moorby: Not so much. Once the language is solidified and you say, okay, a lot of people start with modeling components and all sorts of things. You knew you couldn’t muck about with it anymore. So then it becomes okay, that’s the structure I have to work with. Now, how do I make that go as fast as I can? So the focus is on that. As I keep saying, how do I do fault simulation within that structure? It’s not until you can throw the whole lot away and start again and you start thinking about how can I change the language to make the simulator to go faster, which is one of the advantages I think that Verilog had.

Golson: What was Prabhu’s reaction to this when you said “We have to do a new language and we’re going to do a new simulator.” Did he push back?

Moorby: He said, “Great. Get going. Do it.” I had to go back to England for a few months because of the visa. I had to wait for the visa to come through. I was working on it in those months. Actually, mainly still the language and developing the spec for the language, you know, more detail for the language and how the whole thing can be implemented. I think the decision was pretty straightforward to code it all up in C then because on a Unix machine with C had proved itself sufficiently well to know that that was the way to go.

Golson: And you had experience with C programming already on Unix?

Moorby: A little bit. HILO-3 was written in C. So I had like two years experience transitioning from BCPL into C. I mean it was largely you kept a fairly simple form of C because your focus is more on how to lay out the data structures and focus on details of the algorithm rather than writing as many lines of C code as you can. It was more of a “How do I avoid having to do a certain statement?” So you’re constantly working out shortcuts as to how to make the thing go faster.

Golson: One last question before we leave HILO behind. This, again, comes from Peter Flake who says, “Gosh, it was a real trick transferring fan-fold paper tape to rolled paper tape on two different machines.” You’re laughing now, do you recall that story? How that was?

Moorby: <*laughs*> If I remember right, that was transitioning out to get all of the code we had on the PDP-11 over to the Honeywell Multics machine. I don’t think the Honeywell Multics—did it have a paper tape read? I can’t remember. Possibly. Well, the PDP-11 was a departmental computer that we actually could touch, you know, put the paper tape through ourselves. The Honeywell Multics was a centralized mainframe for the university. You’re never allowed to go into those rooms. You have to hand the thing to somebody. So we had, if I remember, some really big reel of paper tape, okay, well we need to get—so you feel like giving this to a receptionist who’s going to like do what with it? You know, these are your crown jewels and you’re having to hand them to somebody who’s going to “Oh yeah, I’ll put them on a whatever.” You have no idea what they’re going to do with it. That was I think—actually, I vaguely remember Peter having to go back there to actually do it by hand and make sure it was done right. And, of course, they were paper tape readers and they’d notoriously get a number of errors in it. And if you have a big reel of tape, the odds are, that you’re going to have quite a few misreads in the whole thing. So I think I remember once it was actually in the machine, of course, then you’ve sent it through the compiler and then you find all the misreads. So you have to nitpick throughout the whole code to find all the characters that have been changed.

Golson: Moving ahead, you’re in the United States. You’ve joined Gateway. And I understand you spent six months, you lived in Prabhu Goel’s house.

Moorby: Yes, it wasn’t six months.

Golson: No? He remembers it at six months. Was it less?

Moorby: Maybe I was a problem for him. <*laughs*> It appeared to be longer. I did stay in his house for one month over that Christmastime of ’83. When I came back I think I stayed with him maybe for a month. I don’t think it was for very long. I rented a house.

Golson: But the one month that you were here over Christmas of ’83, that’s the genesis of Verilog, of the new language.

Moorby: Yes.

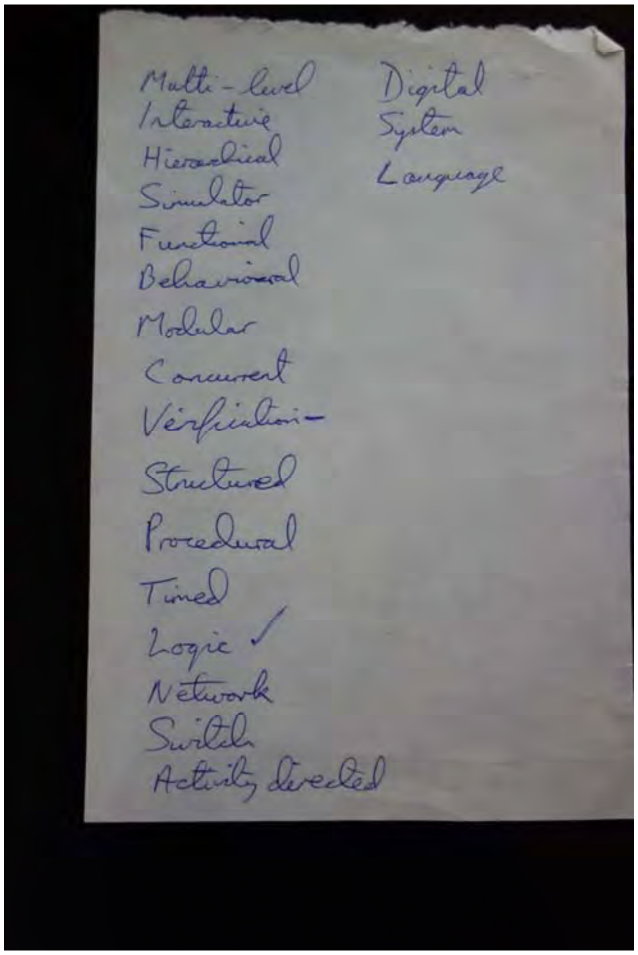
Golson: And it was you and Chi-Lai working on that. Where did the name come from? The name “Verilog,” where did that come from?

Moorby: That was more than a year later I think. So when I came back and I actually had the visa and started to work fulltime on it, it was all development. How to develop this thing as fast as possible and make money.

Golson: Yes.

Moorby: And actually I had a project acronym for it, called EST. I remember, somebody told me that it was some sort of weird group out in California called EST. So I was like, “Oh I can’t use that.” But EST stood for an “Expression of a System of Tasks,” a badly-created acronym but for the sake of thinking of something. Because when you have all of you project files on the computer you have to give it some sort of a title. So the date Q1 of ’85, we had got the simulator and obviously the language with the compiler and everything working to a point where we felt that we could sell it. And Prabhu came to me and said, “We have this opportunity to get the product description in the Sun Catalyst program but we need a name.” I said I don’t have a name. I have a project name that we can’t use and all of that.

Golson: Show us those.

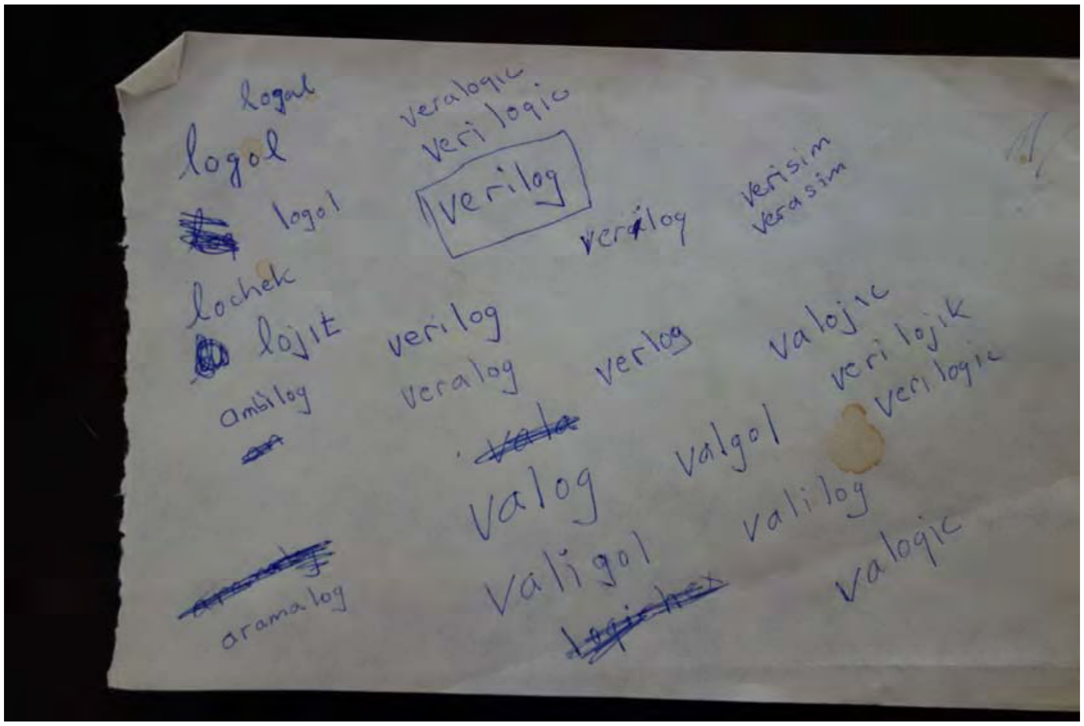


Multi-level
Interactive
Hierarchical
Simulator
Functional
Behavioral
Modular
Concurrent
Verification-
Structured
Procedural
Timed
Logic ✓
Network
Switch
Activity directed

Digital
System
Language

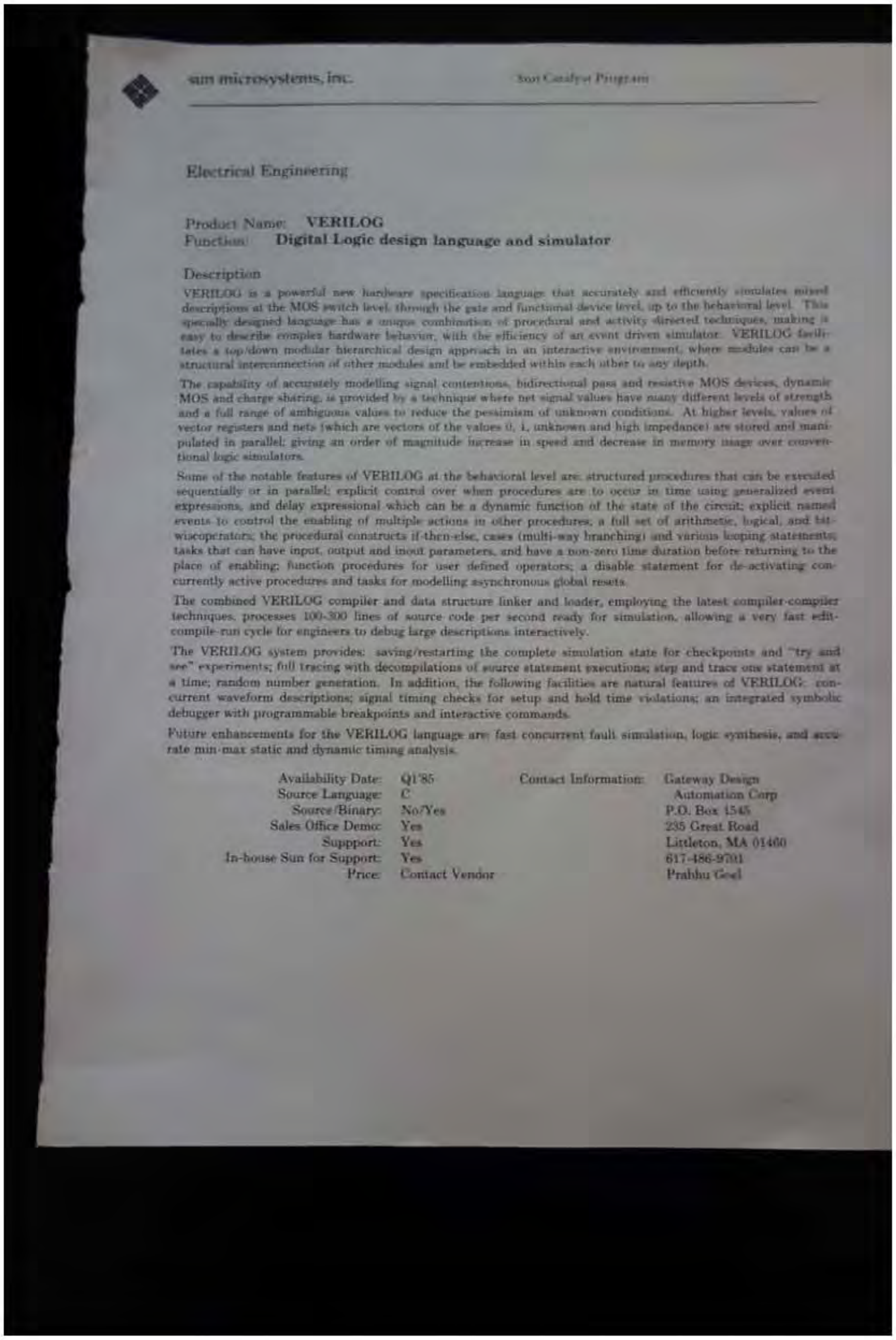
Moorby: Let’s start with this one. <*referencing image above*> When you think of a name, I think what you—well, my basic idea was well, why don’t I think of a whole bunch of words and put two of them together as a way of creating something unique. So this is the piece of paper where I started to scribble all of the possible words that meant something to do with the product. So you go through them and you try and think of what are the good words to use? And the obvious words that came out, “logic” was always a good strong word. And so we had obviously become amateur marketeers for creating names, thinking of names. So “verification” and “logic.” So those actually was even with the first list of words, were ticked off as being the best words to play on. And then you play this game of putting parts of each word together.

Moorby: <*Referencing image below*> So a whole bunch of wordplay to put parts of the two words together. And I think Verilog sort of popped out very quickly. There were a couple of—wow, there’s even a “Veralog.” <*laughter*> I think Verilog popped out, but it never quite sounded—initially I thought it didn’t roll over the tongue, really that well.



I said it's okay, good enough. We threw that in. So we can put this—the very first description of the product into the Sun Catalyst program. <*See image below*> So that came out in Q1 '85. The computer that developed—you know, when I came over some time in the spring of '84, Prabhu had bought a Sun-2 microsystem with a screen. If people know the Apollo, they were in competition with Apollo. But the Sun-2 was pure Unix. Apollo wasn't quite—it was a derivative of Unix. And this was really the start of Sun Microsystems. This was their first machine. Actually, they called it Sun-2 so I think they may have had something before that but, I'm not sure what they had before. But the Sun-2 was a dream machine to work on, from where I came from because the other people in Gateway were programming in FORTRAN for an IBM mainframe. So I said, okay, you do go that. I'll work on this workstation. I had the workstation pretty much all to myself and took something of the order of about a year to get the whole product together, the initial product together, that became, you know, this thing. And made an early sale to—I think we sold it to Apollo as a very, very first sale. And luckily they put it on the shelf and didn't use it because it was still not that much—it was still a lot of things that I said "Well I haven't done this, I haven't done that." They need all of these things. I think it was limited to 32-bit wide vectors and things—awkward things like that, that I knew I had to finish and complete. I think we had about another six months for another customer to buy it and actually start using it. In the world of startups, you have very limited time. You've got to get the thing out there and run like mad to get the thing finished.

Golson: How did the language evolve over that year? If I was to go and look at that very first product, how familiar would it be to me as a Verilog designer? If we go back to that initial one month where you say basically the language was put together, how much did it change over that year?



sun microsystems, inc. Sun Catalyst Program

Electrical Engineering

Product Name: **VERILOG**
Function: **Digital Logic design language and simulator**

Description

VERILOG is a powerful new hardware specification language that accurately and efficiently simulates mixed descriptions at the MOS switch level, through the gate and functional device level, up to the behavioral level. This specially designed language has a unique combination of procedural and activity directed techniques, making is easy to describe complex hardware behavior, with the efficiency of an event driven simulator. VERILOG facilitates a top/down modular hierarchical design approach in an interactive environment, where modules can be a structural interconnection of other modules and be embedded within each other to any depth.

The capability of accurately modelling signal contentions, bidirectional pass and resistive MOS devices, dynamic MOS and charge sharing, is provided by a technique where net signal values have many different levels of strength and a full range of ambiguous values to reduce the pessimism of unknown conditions. At higher levels, values of vector registers and nets (which are vectors of the values 0, 1, unknown and high impedance) are stored and manipulated in parallel; giving an order of magnitude increase in speed and decrease in memory usage over conventional logic simulators.

Some of the notable features of VERILOG at the behavioral level are: structured procedures that can be executed sequentially or in parallel; explicit control over when procedures are to occur in time using generalized event expressions, and delay expressional which can be a dynamic function of the state of the circuit; explicit named events to control the enabling of multiple actions in other procedures; a full set of arithmetic, logical, and bit-wiseoperators; the procedural constructs if-then-else, cases (multi-way branching) and various looping statements; tasks that can have input, output and inout parameters, and have a non-zero time duration before returning to the place of enabling; function procedures for user defined operators; a disable statement for de-activating concurrently active procedures and tasks for modelling asynchronous global resets.

The combined VERILOG compiler and data structure linker and loader, employing the latest compiler-compiler techniques, processes 100-300 lines of source code per second ready for simulation, allowing a very fast edit-compile-run cycle for engineers to debug large descriptions interactively.

The VERILOG system provides: saving/restarting the complete simulation state for checkpoints and "try and see" experiments; full tracing with decompilations of source statement executions; step and trace one statement at a time; random number generation. In addition, the following facilities are natural features of VERILOG: concurrent waveform descriptions; signal timing checks for setup and hold time violations; an integrated symbolic debugger with programmable breakpoints and interactive commands.

Future enhancements for the VERILOG language are: fast concurrent fault simulation, logic synthesis, and accurate min-max static and dynamic timing analysis.

Availability Date:	Q1'85	Contact Information:	Gateway Design Automation Corp
Source Language:	C		P.O. Box 1545
Source/Binary:	No/Yes		235 Great Road
Sales Office Demo:	Yes		Littleton, MA 01460
Suppport:	Yes		617-486-9701
In-house Sun for Support:	Yes		Prabhu Goel
Price:	Contact Vendor		

Moorby: I think it was pretty much what you would say is Verilog. There was some—a number of extra constructs that went in mostly by customer demand. And, of course, we always had a very, very high barrier of entry for anything new because the problem wasn’t so much—the language solved enough problems to work with it. And we’ll get on to things like the PLI and whatnot which allowed it to be extensible. But the basics of the language is all pretty stable.

Golson: Even that early?

Moorby: Even that early. But the problems were not in the language. The problem was in the tools. How do you get the simulator to go fast? Fairly early on, we had started to interact with—there were three synthesis companies had come out and Synopsys, obviously, was the one that jumped onto Verilog and the other two didn’t. We were trying to get them to adopt Verilog but Synopsys had adopted Verilog. So we were working with them to continue to mature various detailed aspects within the language. It wasn’t so much a change in the language. It was refining and getting the details right. If I remember right, one of the central issues is vector bit widths in expressions. What were the rules of extending other vectors to certain widths so the whole expression could fit together, and that you would know what the rules for synthesis were. And those were all matched by the simulator, actually, how it simulated. So in the ideal world they match perfectly.

Golson: That’s right. Verilog was clearly inspired by HILO. If you look at some of the syntax, particularly the netlist, the gate-level syntax is very similar.

Moorby: The gate level was largely the same. Well, the syntax structure—there’s not a lot of choice—it’s just straightforward netlist kind of things. The biggest development probably was getting the logic strength modeling just right for switch-level simulation.

Golson: Because that was something that HILO—HILO did not do switch level, is that correct?

Moorby: Yes, it did. It did switch level. HILO-1 didn’t. HILO-2 we put in—in fact, we put in a bidirectional gate, a bidirectional transistor. We had the initial ideas of how to do the logic strength in the right way to model it all correctly. And the goal is yes, you put the strength into—you’re modeling transistors, but you can’t go all the way back and take the sort of analysis time or the run time the program like the SPICE [Simulation Program with Integrated Circuit Emphasis] program would take. There’s no way you could go back to that kind of speed. You had to go the kind of speeds that logic simulation attained. And even that, obviously, wasn’t fast enough. So you couldn’t do anything that would slow the simulation down. So the problem was, how do you model MOS gates mainly, or transistors, accurately enough to make it useful? And, of course, even still then, the focus was how do you define simulation with that. Fault simulation was the driver that dictated that the simulator had to go as fast as possible because there was no way to complete the job. So it was a balancing act of how to put in a logic strength set of values, that could model MOS gates mainly. What started to become predominant in custom designs is that of modeling PMOS and NMOS gates and various other forms of that kind of electronics, with the strength modeling that both modeled the logic sufficiently well, that if you had a true bidirectional gate that you could still actually model that but still went very fast. So that was the challenge. And so that logic system, although it’s similar to HILO-2, I had developed some of the ideas and how to actually make the simulator to go even faster. So that was always the goal is to achieve that speed.

Golson: So there was some synergy there between the language design and the simulator design? How much back and forth was there?

Moorby: Yes absolutely. At the switch level, it was—well, actually at the switch level, the language really didn’t—I don’t think you could even call it the language. It was like a netlist where you could put delays and control some of the attributes of the gates and the transistors. That’s all you needed from a language point of view. So the problem really, was all about sufficiently good modeling accuracy, but then it was all simulation speed. And, especially, of course, fault simulation. In fact, the one advantage that I did actually have with Verilog is that the initial design decision was to just do the normal logic simulator and we would do the fault simulator afterwards because I think I expressed to Prabhu I said, well, you know, I’ve done the fault simulator with the logics and always together. And it always takes a long time to get the whole thing working correctly because a fault simulator is very demanding. And to get them right is really tough. So he said, “Well, we need a product in a short amount of time.” And I think he had the idea that we could start to sell a simulator early. So said I said okay that’s great. If I’ve only got to do a logic simulator, you know, I’ve done those in my sleep several times. And I have some ideas of maybe how to make it go really quite a lot faster. But the initial version of Verilog which was this initial description—what became the XL algorithm was not there. <*refers to image above*> So as you see the name was just Verilog and not Verilog-XL.

Golson: That’s right, we’ll get to that.

Moorby: So the initial version although I paid some focus to simulation speed, but I relaxed that. I said, okay, the goal is to get the compiler working and the simulator basically get it all working first so that somebody could actually use it. And the speed was adequate. It was still faster than TEGAS. I don’t think—I never got to know whether it was as fast as HILO-2. I think Peter may be able to answer that one. Possibly not because the focus was not on performance initially. It was get the job done. The benefit of that, actually, which I was always very pleased with but many people criticized it for all sorts of reasons afterwards—I wasn’t an expert compiler designer but I knew some of it. So I used tools like yacc and lex and so on, to do the front end, to do the syntax analysis. And I thought of a really simple scheme that rather than getting into all of the compilation techniques which can take you a very, very long time to get right. Of course, it was just interpreted. There wasn’t a lot of pressure to get the high performance at that point.

And I, using yacc, sort of did the syntax analysis, put the whole structure into memory in a one-pass form that meant that I could actually simulate with that structure directly. So there was no multiple passes and translating from one data structure to another. It was just what went into memory is what I used for the simulator. And doing all of that myself meant I didn’t have to define interfaces with other people’s work and so on. And I actually found—I mean it really surprised me that the syntax analysis using yacc and the compilation went so fast that I was actually—in fact Prabhu had gone and evaluated all of the simulators on the market. He knew the performance of all of the simulators. And we actually had some customers starting to look at Verilog and they would give some amount of feedback as to the speeds of other simulators. And I realized that the complete compilations—bringing in the text, doing the syntax analysis and building the data structure and getting to the point of simulation—was far faster than anybody else’s, even any what we used to call linker and loader. So even bypassing all of the time they would take to do the yacc-like syntax analysis I was already much faster than—in fact, I remember some of the run times, of course, with HILO-2 because that had a compilation system, a linker and a loader. And ours was much, much faster than just simply the loader. And I realized that okay, we don’t have to spend resources doing a separate compilation system. So once that was done, I could then really focus on the speed of the simulator, and that became the XL algorithm.

Although all the customers—in fact, some of the customers came back and specifically on their wish list, “We can’t buy this because it doesn’t have separate compilation. And it’s got to have that.” Because they’ve been trained by every other system and compiler, that compilation had to be done in separate units and then put together with a linker. I said, “Well, yeah, but I go faster than all of that so we don’t need all of that.” Some customers got very caught up on “You’re not allowed to do that, you’ve got to do separate compilations.” So that actually influenced then some of the language structures because there was actually one construct that we in the initial version called an external statement so that you could actually do some global external linking. Well, we skipped that obviously. The whole lot, all of the source always went in and compiled every time, every runtime, the whole thing was recompiled, but it was so fast that it wasn’t a problem—until later when the designs got into millions of gates and whatnot. In the early days, that was the initial decision to just let’s keep that really simple and now I can switch to focusing on the simulation speed.

Golson: It seems like that was your real goal was simulation speed…

Moorby: That was a demand in the industry. I mean the hardware accelerators were beginning to come out. And I remember there was one benchmark that AMD was throwing at us and they had given us some run times of a number of other simulators. There was something like about four of them. I clearly remember Daisy was one. Zycad was another. And the Zycad one, that was always hard to beat.

Golson: That’s a hardware accelerator. So we were talking about competing against the hardware accelerators, Daisy, Zycad and…

Moorby: Daisy, Zycad and there was another really quite capable simulator and I can’t remember, Simucad or something like that that was a really capable switch-level simulator. But the main competition was Zycad. They were in their big growth years because they had something that did run very, very fast. Of course, it was very expensive to buy but they were very fast. AMD gave us a benchmark and I looked at that. We could parse it—we translate it into the straightforward netlist form. And I got a simulation run time from it but it wasn’t very impressive. I said to Prabhu, “I know how to make this go really fast.” Give me a little bit of time and it was something like a few weeks kind of a timeframe to throw together something, just to get the initial evaluation of how fast we could get the simulator to go. And in combination with that, I was still working on the details of getting the logic strength levels right. So what I did was a sort of a specialized algorithm off on the side, that had most, if not all, of the gate-level primitives. We bypassed the bidirectional transistor because that was very, very tough to get simulation speed with that. So we have unidirectional PMOS gates and NMOS gates and trireg and all of the gates.

And so I carefully worked out how many logic strengths we needed to do decent modeling of all of these different transistor types. And how to do a central kernel of a gate-level simulator. I had to make the event wheel go very, very fast, how to do the evaluation of a gate, with a very, very simple very fast single table look-up kind of a technique. So I put one initial version together and I got the idea that the speed that I thought that we could achieve—I could just begin to feel and smell that we could get to compete with the hardware accelerator if we could maintain that kind of speed and put everything else around it that wouldn’t slow it up. And if I remember right, I went through a couple of different versions of that code, but I had this benchmark to work with. So I knew—and the benchmark didn’t have a large number of primitives in it. So I didn’t have to develop too much of the simulator to test it to see how fast it would go.

Golson: And this is original Verilog?

Moorby: This was a transition, but the code I was actually working on this really became what the Verilog-XL part.

Golson: So this AMD benchmark is happening after Verilog has been in the market for a while.

Moorby: Correct.

Golson: I see.

Moorby: So the full language was for a much more of all kinds of different kind of design styles. AMD, of course, was all into very custom design—everything was PMOS gates, NMOS gates, and CMOS and trireg. And they were putting together designs which are really pushing the boundary of capacity. I’m trying to remember how many gates. There would be probably up to 10, 20, 30,000 gates, transistors, PMOS and NMOS gates. So the compilation system that I talked about was adequate. You’re able to read in them really quite fast and get to the point of simulation. So it all came down to competing with these other simulators. And AMD was good to actually tell us the target, and some of the speeds that these other simulators were achieving. So we went through them all. We easily beat the Daisy system so I thought okay that’s an easy one. This other software simulator I think came out about three or four times faster. So that was good.

Golson: You were three or four times faster than this other one?

Moorby: Yes, somewhere in that order. It was a very capable, very efficient switch-level simulator. It gave us trouble for many, many years after actually. But it was still—that was a software simulator. And because we were about three or four times faster, they said, okay, we have the edge over that, that’s great. The goal was to try and get close to the Zycad speed. And the speed that AMD quoted us I measured it to be about two to three X off, slower than the Zycad. So when I told that to Prabhu he said, “Wow, we’ve got something here.” I mean they sell their hardware for something like a million dollars, the full system, and we’re just a simple software simulator and can be effectively just as fast. So we knew we had something with that. So the rest of it was completing it—filling in all of the different gates and switch-level components. There was somewhat of an awkward situation which I’m sure many later customers of Verilog-XL felt there was this disconnect between—if you had everything in gate level, you seemed to go really, really fast. But as soon as you put in any kind of an RTL statement it would slow it up. And this was because there was really two simulators in there and two time wheels and so on. And so you would have this interface of passing values back and forth across this API. And the slightly awkwardness in having the two event wheels. But it meant, what we call the behavioral level around all of the gates, that you could write your test bench and do all sorts of high-level statements with that. And if you didn’t have too much of that, the simulation speed wouldn’t be too much of a problem. But if you can get all the activity down into that switch-level simulation then it would really go fast.

And the initial version of that was on a Sun-2 machine. I started to experiment with writing in assembler because the code was very, very small and compact. So actually it was counting the instructions. I remember that the number was—actually I think it became a goal. It was 50 instructions per event.

Golson: Fifty?

Moorby: Fifty. We developed a benchmark statistics but the average gate had a fan out of two-and-a-half. So you’d actually with two-and-a-half either outputs or inputs and the work you have to do to evaluate a gate. So you go through the code, instruction by instruction and add them up. And then you look at the code, how do I get rid of a couple of instructions here and a couple of instructions there and so on? So that was finely tuned to get to top speed. And then from that initial port to the Sun-2 we started to look at other machines. AMD said what about their mainframe which is an IBM 360.

Golson: Oh, you’re back to IBM now.

Moorby: Oh, that machine. What are we going to do with that because it was very, very painful to do anything with it. So yes, the assembler—I mean we knew the machine went very, very fast. But to get time on the machine, there was no C compiler for it at that point in time. The C compiler came a little later. So we thought of studying how to do the assembler version of the actual algorithm on the IBM 360. And we did that and AMD was extremely happy with that speed. That was very successful. And unfortunately, then we had to port to the IBM from then on. But other ports went a lot better. The Apollo was a good machine to work on. I think that another good one was a Silicon Graphics machine. It was very fast. That was a good port. It ported on to that machine very, very easily because it’s all standard Unix based.

Golson: So let me back you up a little bit on to some more of the language issues. One thing you mentioned was delay. There was delay modeling in Verilog. And was that something new? Or did HILO have the idea of delay modeling?

Moorby: No, accurate delay modeling went all the way back to HILO-1. So I had learned all of the tricks of the trade pretty much from the HILO-1 team as to how to balance the delay modeling with a simulation speed. And we did a fairly accurate delay modeling in HILO-2. I remember from my timing analysis research I knew what the problems were. But at that point, I could never—well, there was this dynamic that went on with the customers. The customers seem to form the impression that a logic simulator ought to be absolutely accurate with timing. And I think when you have a pulse going through, the classic problem is when you have a pulse equal to the delay of a gate and that pulse is passing through a gate, what should a logic simulator do? Should it be really accurate, or should it be approximate or whatever? So in HILO-2 we did quite a lot of attempting to get it really good. And, in fact, I think we had a mode where we could actually turn the pulse into an X—it’s known as using the X value to make it unknown so that it would cause—rather than allowing a flip-flop to be clocked, you could say well, it’s unknown whether the flip-flop is actually going to be clocked because it’s so narrow. So we’ll put an X out there and have the flop go to X. And then if that matters in the design, then you will get to know it and you’ll find a bug in the design or work around it.

So a lot of very careful modeling went into HILO-2 and HILO-3 as well that meant the inner event wheel of the simulator was affected by this because in order to do that modeling correctly, the central algorithm on every event it had to test to see what events had been set up before, and it’s all about do you get this event coming in and the question is you do a decision table to look up to see what should I do next? Well, if you’ve always got to do an extra piece of work to say, well do I have an old value here or not? Do I have a pulse here? You know, all of these decisions, that slows the inner workings of the simulator. And knowing that that sort of thing slowed the simulator up, for all of the years coming out of HILO-2 and starting Verilog and that I think, do we really have to do that? Do I really need to do that amount of detailed modeling when I know that this is not real timing analysis? I mean it’s no substitute for a real—especially a static timing analyzer. And, in fact, the world was going more and more towards using static timing analysis, without any timing to verify the timing. I said, well, if designs have to meet that criterion, why on earth is a simulator modeling these glitches so accurately? What’s the point of this? So in the XL algorithm I did a little shortcut that actually made the simulator go faster. And that was, on the output of a gate I restricted it so that only one value could only ever exist at one time for the scheduled future. So it’s got the current value and what it might be going to some point in the future. And that was it. Because to get any more modeling in there, you actually have to keep a queue. And with memory, a queue implies going to get a little chunk of memory, bring it in, I’m filling it with data and then I’m putting it back into the garbage list to be garbage collected and so on. Well, if I didn’t worry about these glitches, I don’t have to have any of that. And the decision-making going on in the simulator would be so much simpler and faster. So in terms of those 50 instructions that was the way to get down to 50 instructions. So I told this to Prabhu and everybody around. We didn’t say this to the customers, obviously, because the problem with the customers coming back saying, “Oh, no, no you’ve got to model glitches accurately. You can’t bypass—you can’t short that one. You’ve got to do it accurately.” So I was like, let’s not listen to customers just yet. And it was proved to be correct because I think the thing that drove the sales was all about simulation speed. Really, it was competing with the hardware accelerator. And the question is, is the modeling adequate? Was it doing the job? And I said, well, really that kind of analysis is up to the timing analyzer, not really the logic—the logic simulator has got a lot of logic verification to get through and you want to get through that as fast as you can.

Golson: I would imagine that the hardware accelerators like Zycad, they had their own way of handling delays and modeling delays and that would not be…

Moorby: Well, every simulator was different. So who was correct? And that was another question that kept on going over and over in my mind. I said, well, if everybody is different, then does it matter? So we never directly told any customers. I think years later some people had decoded the simulator sufficiently to say I’ve got this pulse going through the gate and sometimes it goes through and sometimes it doesn’t and they’d say “What’s going on here?” “I don’t know, let’s have a look at it.” <*laughs*> And then you always twisted the answer back to say, well, why are you trying to do timing analysis?

Golson: Yes. Yes. Or you say, what do you expect the actual real gate to do when you give it that situation. Sometimes it will work.

Moorby: And the answer was really to know the answer to that you had to run SPICE. Therefore, okay, go run SPICE. <*laughs*>

Golson: So you’ve been talking about Verilog-XL and the XL algorithm. So how did that all come about—the idea of “Well, we need to make the simulator faster.” Was it, you knew it needed to come faster and you had this idea? Was it the demand that drove…?

Moorby: We were well aware that the hardware designers out there were having a real hard time using logic simulation to verify their designs. They would just simply say, “It doesn’t run fast enough. I could put a breadboard together in a day or two and that runs at speed.” So that was the competition way back. So I think the hardware accelerators started to change that somewhat because they would be going so fast they could start to compete with breadboarding. And, of course, there was this process where—the work and the benchmark with AMD, that was a very much of a custom design. But the capacities were going up. So initially we would be working with 10,000 [gates] and actually at 20,000 and then it was 50,000 and then it was going up so that it was doubling every year or two. So we knew that the difficulty of that was verifying the design before you actually made it. And then, of course, the problem—the curve started to really take off where the cost of making, fabricating the chip was costing so much that making a mistake was more and more painful to them. So it was pushing out this old mentality of using a breadboard. That was obviously not going anywhere and getting more and more difficult to do.

So when you push into especially custom designs and that’s why AMD went out there looking for the fastest simulator because they had a very severe problem of verification of their custom designs. And their designs were really, really big for those days. Right at that point, this whole market was starting to open up with ASIC design. So the roles were a little different, but still what that market absolutely needed was fast logic simulation for verification. And many of the ASIC vendors already had developed their own simulators, though most of them were all self-developed. I believe most of them that I encountered were running on an IBM mainframe. And their customers would lease time on those machines to run through their tools, but they had to run through their simulator and obey their rules of design, to pass what’s called the certification process so that the ASIC vendor would guarantee that if you ran these tests and they passed that they would guarantee the chip they delivered met that spec. So simulation speed was central to that whole business. And many of the ASIC vendors were obviously starting to look at the hardware accelerators. And we’ve started to see ourselves as being able to compete in that space—to push out the very fast simulator.

So Prabhu had this vision of saying, okay, the future for us is to win the ASIC market, to become the golden simulators for these ASIC vendors. And that was an absolutely brilliant vision because that really made the whole business thing really started to take off, but there were a whole bunch of rules that we had to meet. The simulator had to do certain things. And some of those things were like pulses going through the gate and being modeled correctly. <*laughs*> They needed things like pin-to-pin delays and timing checks, all of that stuff. Part of the ASIC sign off process was a form of doing timing analysis. And I could never figure out—that was not a guarantee that the design was really going to work. So I think behind the scenes the hardware designers would always end up having to run a true timing analyzer. But the ASIC vendors put a lot of pressure on us to put features into the simulator that met their golden simulator. They wouldn’t allow us to replace their golden simulator without that. So we worked really, really hard on putting pin-to-pin delays, timing checks, more accurate delay modelings and so on, which all had the effect of slowing the simulator down.

Golson: So with simulation speed, you had a couple of marketing explanations for how Verilog-XL was sold. I have two of them here. One was "adaptive behavior analysis." That was one explanation. And the other was "clock suppression." But really what was it?

Moorby: <*laughs*> It was not that.

Golson: Exactly, yes. It wasn't that. So what was it? Was there one specific thing? Or are there several? What was the big breakthrough in your mind?

Moorby: Other than what I had gone through which is a very, very careful coding, pretty much at the assembler level initially, although it was written in C as well for it to be portable. And on some machines that C went almost as fast as the assembler versions. That was it. There wasn't really any magic to it, other than, you know, some of the very, very carefully selected shortcuts.

Golson: Were you aware that your competitors were trying to figure out how it worked?

Moorby: Absolutely. The reason that phrase you came out with came up is that Prabhu came to me and said, "We have a bit of a problem here because everybody is going to jump on it. Well, customers are wanting to know…" Part of the sales cycle, you know, the customers come in and say, "Well, we need to know how you do this." <*laughter*> So it was supplying the need of an explanation. In fact, I brought out some memories of academic world, some of the academic research Peter Flake and myself had got into about how to do simulation faster and faster. And I thought, well, okay "clock suppression" is one. Well, "adaptive" sounds good. So I'll just put something together and okay throw that one out.

Golson: And where did the name XL, Verilog-XL where did that come from?

Moorby: A marketing guy joined us—it didn't come from me. It was a pure marketing exercise to say—I think he came in one day and said, "Well, the two letters—I think it was like this year's car model letters to use for a very high performance was XL. So we're going to use XL." <*laughs*>

Golson: You mentioned Verilog-XL becoming a golden simulator for the ASIC signoff, and a number of the changes that you had to put in: pin-to-pin delays, specify blocks. You had this wonderful language. And it had this simplicity to it. And here's those nasty customers coming in and forcing you to put these changes that slow down your simulation.

Moorby: Firstly, it wasn't the customers. It was the ASIC vendors.

Golson: Okay, it was the ASIC vendors. All right.

Moorby: The customers would have to go with what the ASIC vendors said.

Golson: How did that affect you? How did that make you feel? It's like "Oh yet another thing I have to go and put in?" Or was it just yet, another puzzle, another challenge, they wanted to do this?

Moorby: I never said no to making money. I mean Prabhu came in and said we had these customers we were chasing. I think Fujitsu was one of the leaders in the ASIC vendor.

Golson: And Motorola was the first.

Moorby: And then Motorola. And the guy in Motorola was insisting that we do certain things if we were to have the business. So Prabhu saw the writing on the wall that to make it to this golden simulator status we have to do these things. I said, okay, all right, we'll just do it. But we did it in a minimalist form just to satisfy those obligations. So that's why a lot of the constructs went into what—I would say infamously, I'd say went into the specify block. And then, of course, then decades later nobody could work out where do these specify blocks, what's the point of these? Well, they were pushed into the language because of the ASIC vendors.

Golson: On your sheet where you wrote down the keywords that you used to come up with, the very first one on the sheet was "multilevel." And I know you've said that was one of your key goals that was driving the development of the simulator. So what levels make it "multilevel"? What levels were you trying to get?

Moorby: I think coming out of the HILO-2, HILO-3 days most of it was a given, so you had obviously the switch level, gate level. I don't know what generally you would call the other levels. So we should talk about UDPs at some point. I thought that was very interesting. That dealt with the flip-flop level, with speed, because there are many different varieties, variations of flip-flops that you can have. The problem then is we were not going to get into developing a custom primitive for every one or those flip-flops. And the problem with going to the RTL level and to get the logic right, meant that you've got a lot of code around it. And if you have a lot of code, that means having to simulate all of that code. So for a simple thing like a flip-flop, you couldn't afford to go that slow in a simulator. So we'll get on to UDPs as the answer to that. So you've got switch-, gate-, flop-level if you like. And then an event-directed kind of a language. That was in HILO too. But knowing that in order to really get to deal with large complex modeling, you have to provide at least a fairly decent procedural language. And that was trying to get that right, so remember to be able to do fault simulation and logic simulation with procedural languages what was—I couldn't initially imagine how we could do that and have good simulation speed as a balance to being able to have good ability to model. So introducing things that were procedural was obviously the next level up. We knew that we probably would never get to even maybe higher than that. But if we could do a pretty decent procedural language where you could model these higher level components effectively. And to be able to do all sorts of modeling, not only flip-flop but go up to microprocessors, so you could describe all of the instruction decoding, logic and whatnot. And that takes a lot of code. And we tried to model that in HILO-2 and it was very, very clumsy because there was no ability to have a procedure that you could call. We had a system of passing events around which made it very clumsy to call a procedure and have it return which is the classic procedural construct.

Golson: The procedural constructs, the syntax that was put into the language, I can see the influence of several languages: C, Pascal but also occam which many people don’t know about. So talk about your history with occam and how that drove Verilog?

Moorby: Occam is a brilliant, small—the term occam came from a guy called William of Ockham and he came up with a very famous phrase: Occam’s Razor is the title. I can’t remember the details of all of his philosophy. But it was about, if there’s a choice between two things, then you would choose the simpler of the two. And that’s what Occam’s Razor was about. And I picked up a paper and studied occam a little bit. And in the occam language there’s just a few constructs that I could see that solved a problem I was thinking about to dealing with fault simulation in a modeling language that had yet to be invented, obviously. And those two constructs—there were two key words that was used in occam called SEQ and PAR. The SEQ is essentially a sequential block where you could say begin-end or curly braces in C. But the PAR, that became the fork-join construct. It was a structured concurrent expression of how to execute several statements in parallel. With the structure, at the end of the statement, the control threads would actually recombine into one. And, in fact, the occam language it’s connected to a whole—there’s a professor, I think it was Professor Hoare at Oxford University and he developed a whole formal system that was based on these constructs SEQ PAR and I think he had a couple of other constructs. That if you expressed your hardware that way you could actually do formal analysis. When I saw that, I saw that as a solution to a problem that I was thinking about in fault simulation, that you get divergent activity that basically explodes as the activity propagates. And the question is how do you keep that under control? So I thought of this wonderful idea, of taking these constructs and putting them into Verilog and that became the sequential begin-end and the concurrent fork-join. So the fault simulator, of course, is another story that maybe we’ll get on to. But that did come later, but the focus was, of course, on the logic simulator.

Golson: One of the key things about Verilog or any hardware description language is the parallelism of it—that’s the way hardware works. And so other than occam were there other prior procedural languages that you had…

Moorby: Well, C—it was a natural given to use all of the operators and the expression structure of C. In fact, Chi-Lai Huang with his PhD work had used the same. Actually, I think something similar but he came more from the Pascal language. So it’s sort of a melding of a bunch of ideas there. But I think very quickly it was obvious to match the C operators. But there was this additional problem that in a hardware description language you have vectors at different widths. And you have something a little similar in C because you have variables of different widths, and there are some rules that you follow to work out how an expression is evaluated. But we didn’t want those exact rules completely. We wanted the same set of operators, but not the same rules. So Chi-Lai and I worked out a whole bunch of rules of how to do it in an HDL that made sense for synthesis. And it was a bit of a goal of trying to optimize how many bits you would actually use in the middle of the expressions, and so on. And to do our best to come out with natural rules that would be relatively easy to learn.

Golson: I see.

Moorby: So that was the C influence, the Pascal influence I think you can see a little bit and occam. Obviously the HILO-2, don’t forget all of that influence.

Golson: Yes, that’s right.

Moorby: The other <inaudible> why I think it was those structured statements, the other one was the ability to spawn off a process. So the event trigger mechanism I think would say okay well that’s good enough to spawn off a process.

Golson: Who were your early customers for Verilog? I mean very early on. You said that GenRad bought a license and didn’t use it.

Moorby: No, not GenRad, Apollo did.

Golson: I’m sorry, Apollo bought one. So when you started getting feedback from your customers, what were they using it for? What did you get?

Moorby: General Electric bought a copy. And they wanted to do some high level modeling of an architecture design they were building, I can’t remember who for, but the guys there were extremely smart. They put together a pretty large design very quickly. Wrote it all in Verilog. I think they wanted to be able to express the low level concurrency in a way they had in mind. And they wanted to do, I think, some statistical analysis on how the system would perform. Put it all together and then realized how slow it was. And we said, oh, well, we’re not focused on that at the moment. So they actually in a very short period of time translated it all into C and said, okay, well, this goes much, much faster so we’ll carry on with that. But really, we were all focused back at the gate and switch level competing with the hardware accelerators and really very fast getting into the growing ASIC market. And what had started around about a similar time but took longer to develop was the relationship with Synopsys and the synthesis tools.

Golson: So how did that come about? What was your first interaction with the folks at Synopsys? They came out of GE and you mentioned GE. Was it all the same team?

Moorby: No, I wasn’t aware of where they had come from at that point because there were the three companies…

Golson: Yes.

Moorby: Sorry, three startup companies. More specifically they were startups, Synopsys, Trimeter and SILC. And they all had various abilities. At the time when we started to talk to them because Prabhu had the idea of telling them, insisting almost that they really had to use Verilog because that was the future. SILC had started and were more interested in doing their own language. Trimeter to be honest I can’t remember what they did, how they ended up. And Synopsys said well, Verilog does the trick and they said, okay, and they jumped all over Verilog.

We got a little diverted there. What was the question before that?

Golson: Who the early customers were for Verilog and what were they using it for?

Moorby: Because we had the performance problem at the behavioral level, other than one or two sales at that level it didn’t really take off. So we realized that all of our focus should be pursuing this ASIC market. And we had a pretty healthy market for the custom design for AMD. Well, Motorola had a fair amount of custom design. But they obviously wanted to push and grow their own ASIC market as well. So the custom designs and the ASIC vendors type pressure was a little bit different because one was all switch level. And the other was all gate level with all of this extra stuff: pin-to-pin delays, timing analysis and so on that they seem to insist on having. So because that business was clearly taking off faster than anything else then that became the focus.

So the synthesis stuff was still a thing on the side that we still had the vision to do ourselves, good that we had started with that vision. We could see these three startups are already started, and I think at this point of time, they had a little bit of a slower growth—we knew that we would get a lot of customers in the ASIC market for being the golden simulator so we chased that and constantly put off the project to do synthesis until it became too late.

Golson: Were you ever surprised by what your customers were using Verilog for? When you saw something, “Wow I never imagined that they might have done that sort of a design or used that construct in that way?” Do you recall anything like that?

Moorby: I don’t think so. <*pauses*> If I think long and hard I probably think of some—not so much of a design style but some of the constructs they seemed to try to use. Yeah, we’ll get into some of that. Some of those, the #0.

Golson: Yes. <*laughs*>

Moorby: The infamous #0 construct.

Golson: Yes, we’ll get to that maybe after lunch.

Moorby: That surprised me, yes. I said what on earth are they doing with that? <*laughs*>

Golson: The idea of text based HDL design—maybe we’re moving ahead a little bit because you said initially the customers were not using the behavioral constructs so much. But once they did, it’s a big change from using graphical schematic capture to doing text-based hardware design. How much of a struggle was that to convince the customers? What was that like for you?

Moorby: So this was a process of really transitioning from what one would call the pure gate level which was largely all netlist. And nobody could really imagine people writing that text. The graphics, is a much better way to do netlists. So we always thought that that wouldn’t be used as a language per se of entering in a text editor. That would always come from some kind of a schematic layout system. So I think in the combination of what we had in the language with really what Synopsys was making a success in synthesis, that they were edging up the level mainly into really combinational expressions and some of the constructs that would do the simplest sort of flip-flops. And then they—it wasn’t in their initial product offering, but they then started to push the use of writing procedures in an always block so that you could start off with the certain triggers and actually be writing a good page or two of flow of logic. And it started to become easy to do. And they would synthesize in that logic so effectively that there was really no looking back. So there was no—that, and being able to write decent size expressions of continuous assignments and so on. I think it bypassed the need to write gates out in a form of a netlist. And then the world started to very, very quickly pick up on that and that was the way to go. And I mean people are not using synthesizers. We’re all—you know, obviously we’re all still sort of doing graphical entry to generate all of the gates.

Golson: At that time.

Moorby: At that time.

Golson: Yes. So apparently there’s an interesting story about how stochastic modeling was added to Verilog for Hughes Aircraft.

Moorby: Hughes Aircraft. Actually, I didn’t go out and visit and so I didn’t have a hands-on knowledge of what they were saying, but they—I think they started to use Verilog in a similar way that GE did before. And I don’t think they—I hadn’t heard comments coming back saying “Well this is way too slow” at that point. But I think in their need to do this high-level modeling and a systems statistical analysis as to how the system would respond in certain situations they needed all of the various constructs you need to do stochastic analysis .You need queues and some of the random functions, you need all of the basic Gaussian random and Poisson random. And they came out saying—I think they had access to another system that did graphics of some kind for their statistical analysis. “We really need to have some kind of graphics and you don’t have any.” So Prabhu came home and said, “Well, how long will it take to throw something together?” He would come back to them and say, “Well, tell us specifically what you want.” And they would say, “Well, we just need this input to display the statistics of this queue. So we want see dynamically the queues growing and shrinking.” So he came back with that and I said, well, let’s throw something together. In the space of I think it was like a weekend we just put some basic things together. I think he went out with a demo and they love it. That’s what they want. It’s good stuff. I mean very, very simple. I mean some of that stuff can be really quite simple to be effective.

Golson: So you’re making these changes to the language. You’ve got the ASIC vendors, you’ve got some customers, who ask, “Put this feature in, put this feature in.” At what point did you feel, okay, the language is done? There’s no more. Or was it just now looking back you realize, oh, we haven’t changed anything in a while. Do you recall having [such] a moment?

Moorby: What I considered and thought of as “the language” was done way back in that one month. Everything else was just warts on the side.

Golson: <*laughter*> We’ll get to SystemVerilog. That’s a really big wart, perhaps. We’ll get to that later.

Moorby: No, no, that’s not a wart. <*laughter*> Specify blocks. I think warts on the side is a good description of that because they were not general. You couldn’t put things into a specify block. The reason why it was encased in what we called the specify block is to keep it separate from the rest of the language, because one of the biggest problems with that is if you put that stuff in every flip-flop… They would have modules with a single gate in it. And they would have like a dozen timing checks or pin-to-pin delays on a gate. So rather than have the gate evaluate—you remember the 50 instructions, it would explode into probably 1,000 or so times slower. Well, when I looked at that… One of the biggest problems—not only did it slow the simulator down, although we actually put all of that construct into the XL algorithm so it did go a pretty good speed -- but the problem was the amount of memory it would take during the compilation. So the reason why I thought of it as a wart on the side is that the way we dealt with the memory instantiation inside was done in a very special way. So it was just to get through compiling it in a decent—without the memory exploding. And the other main feature that had to go with all of that was back annotation. So once it’s in memory and compiled, they demanded the ability to have a back annotation file with all of the delay specs in it that they could load up. So all of that was done for those requirements. And really, the specify block never was what I consider part of the language. So that’s why I would call it a wart. The wart path had a lot of problems. I mean there was a lot of cutting corners just because we had a very tight set of constraints and requirements to meet. All we had to do is to meet those requirements. We didn’t have to get carried away with anything at all. We didn’t have the luxury of saying, well, let’s not just do that, let’s make it more general and make it so that we can do a lot more with it. We never went there. Just do the absolute minimum. Get the thing working and that’s done. So that’s part of being a startup. That’s part of fighting hard in the marketplace to compete with other very capable simulators.

Golson: One thing that I know popularized the language was the textbook. You wrote with Don Thomas, you wrote this very influential textbook called “The Verilog Hardware Description Language.” But, of course, we in the industry always called it the “Thomas Moorby book.” How did that come about, the idea of writing that as a way to popularize the language? What was the thought behind that?

Moorby: It started as an effort to get into the universities. We had a couple of good contacts. We’d done Thomas. Obviously, we knew him all ready. I think there was—he was one of the main professors that we had a good contact with. There were some others but I think Don expressed the wish to teach it because he’s—I think at that point in time I think the rest of pretty much the world in the academic world didn’t want to know anything about VHDL.

Because VHDL was largely—well half of it, at least, was developed in academia. Obviously, sponsored by the DOD. But all of the professors were jumping on that bandwagon.

Golson: On the VHDL bandwagon.

Moorby: The VHDL bandwagon. Not at that point in time, where I could see, was related to anything practical. They just said how do we put this great language together? I said, what about the tools it’s going to cover? They didn’t have that mindset. They said, we’re doing it to model hardware and the tools can do whatever they have to do. So I go well… <*laughs*>

Golson: Very academic.

Moorby: …of all of the experience I had I said, well, you’re going to have a few problems if you try and make it that way. So I’m sure we’ll get on to VHDL a lot more later. But that partly gives a little bit of a scenario from my perspective how I viewed, what was going on with VHDL.

Golson: So Don was one of the enlightened academics who saw Verilog for that.

Moorby: He just wanted to get a simulator that worked and started to teach and just get going with something. In university circles, the language and simulator called ISP was very popular. And I think he could see that Verilog was much better than that and going somewhere, because he could also see Synopsys doing something with it. And in the ASIC vendors it was going somewhere. And he wanted to teach it. And he was practical enough to say, “Well, I need a simulator, and VHDL, they’re years away from a simulator.”

So he wrote a text for his course. And he gave us a copy of that and we all read it. I said, wow, he can write. He’s really good at explaining this kind of technical detail. And I think Prabhu had the idea of saying, “Oh, how do we convince Don to write the rest of it?” And he thought of the idea of getting me to get in there with him and do it together. I can’t write that well but I’ll try and encourage him to write more. So if you ever see the first edition it was very well written but quite small until it grew and grew, which was to do with him writing more and more for his students. And probably getting some of his students to do some.

Golson: No doubt.

Moorby: A lot of the exercise, I think, came from his students.

Golson: That’s right. That’s right.

END OF PART ONE

START OF PART TWO

Golson: All right. So picking up again, when Verilog, particularly, Verilog-XL was being sold, GenRad was still selling HILO. And did you realize you were in competition with your own previous product? Were you aware of that?

Moorby: A little bit, but we didn't really cross paths all that much. I think they were still focused on the test market, and we were pushing hard into the ASIC market, so I think that kept us fairly clear, you know, without crossing paths too much.

Golson: You've talked about how Verilog and Verilog-XL was an interpreter for the language. Did you think of compilation, or were you surprised when Chronologic came out with the first compiled Verilog simulator? I was wondering how you reacted to that.

Moorby: It was something of the order of about a year before we opened up the language. Because when the language was opened up, Chronologic jumped onto that really fast. About a year before that, we actually had named a project, an internal project, called 10-by-10. I think Prabhu actually named it. And the 10-by-10 was 10X faster and 10X less memory.

Because the amount of memory—because although the compilation process was fast, it had—with the process of instantiating the modules, it came from the decision earlier to make the instantiation of module very easy. Basically you just do a straight copy of all the memory that the module would take. So as designs grew, the amount of memory required to go through compilation and actually get into simulation was—there was a lot more memory actually used for the compilation that was not taken away for the simulator. So it did become a problem eventually. And not so much once it got into the simulator. Because the amount of memory it took—and especially the XL gate simulator part, it used up actually a very, very small amount of memory. But the compilation process, it used a lot of memory.

So part of that project, of course, was to try to figure out a way to really reduce the memory. And that was an ongoing problem forever. It is a serious problem in simulation as to how to deal with the instantiation of all the modules, and get that right. So the 10X more speed in simulation was actually about the behavioral-level speed, obviously. Because the gate-level speed we were doing, there was no way to make that ten times faster. Unless you talk about the hardware accelerator, which did—but in software it was the optimal kind of speed. So the 10X was that of, okay, at the behavioral level where it was all beginning to take off and expand into—I did some basic architectural work to work out how typical pieces of code, just how fast it could potentially go theoretically. So I did some of those theoretical studies. I said, "Well, it ought to be able to go ten times faster, if not more than that." So Prabhu really wanted to push ahead into this 10-by-10 project. And wasn't too long—I may well have used the term “compile the code,” although I think that term really came through later, as a true—to match what compilers did in terms of generating code. I think that from the initial studies of what could be achieved, that had set the goal. And we had started one main sub-project, I think there was one or two people on it, that dug into figuring out how to make that behavioral-level simulator go much faster. That was just before the language was opened. And I think it may be another one of these lessons. It is that if that isn't your central focus, you tend—especially start-up companies—tend not to do that. Not to get anywhere with little projects off on the side.

Golson: Yes, yes.

Moorby: And the central focus was all about how to become the golden simulator, and saturate that, and take over all the ASIC vendors. And I mean, that was the big win. And the behavioral level’s speed, I think, really it was a growing problem. At the time when we had recognized the problem, you know, a company like Sun Microsystems, and I think a few other customers were beating us up pretty badly to say, "Well, why is your behavioral-level simulator so slow?" But it wasn't hurting our main growth pattern. So I think it was a clear case of not putting sufficient amount of effort into really doing a good job on that. Plus whatever we would do with that, I always thought that the reason—at that time, of course, it was Cadence who was responsible to make that behavior-level simulator go faster. And that eventually allowed Chronologic into the market and that competition situation. They clearly didn't put enough effort into it. And probably the biggest effect that held everything back was that whatever we did had to be perfectly backwards compatible. So you had all this legacy code. All these customers. And you couldn't upset any of them obviously by changing anything. So everything had to work exactly the same. So the speed up had to be completely seamless. That, in conjunction with the fact that it wasn't the central project that was going on.

Golson: Yes.

Moorby: That makes it really, really hard to do. And it allows very capable people and company like Chronologic to get in the door and actually do a good job and not be encumbered by legacy code. It can just focus on that particular area of expertise. And that's what they did. Chronologic did a tremendous job of basically teaching Cadence a lesson as to how to do it right. And then it became a battle between gate-level speed and switch-level speed, is it there? Of course there was that play between the ASIC market, which was still growing, and what was happening in synthesis. And because the synthesis level of description was growing up into what, generally, we'd call the true RTL level. And so a company like Sun Microsystems would excel in that area, and taking full advantage of the kind of simulation speeds that Chronologic was offering. Even though, for the longest time, Cadence kept hold of the golden simulator status for the ASIC vendors. And then probably early '90s, when I think the emphasis started to switch, where the revenue growth was more in this higher level verification area. But worked well with synthesis. And then that allowed Chronologic to—I think they ended up with more revenue in the end, than logic simulation revenue. And then a little later on, after I'd left Cadence, they'd woken up to the fact that they needed to do something about it.

Golson: I understand there was also a hardware product that you worked on, gate-level hardware product?

Moorby: Wasn't soon after—actually the XL algorithm. It was before Gateway was bought by Cadence probably a year or so before that. Looking at the code—and still, of course, the heat was on in terms of competing with the hardware accelerators, especially Zycad and some of the other—IKOS was starting to happen as well. It was—you could never have enough speed. I thought, "Well, okay." The algorithm itself was really quite simple, very compact, and although some of the parts of it was complicated, but the code was still very compact. And all the table lookups were very compact. I thought, "Oh, it wouldn't be too difficult to build hardware for that." So I started a little pet project off on the side. Started to write the hardware, how it could go in Verilog. And we got a hardware designer or two, and they actually built boards for it. And it really did go—Have to think of the numbers. More than 10X faster. The fastest processes that is, compared to, you know, to the hardware. So with an average sort of clock rate of the hardware. It went more than ten times faster. So with that 10X speed up, it was just about worthwhile producing hardware for it.

Golson: And how much of the language did it run?

Moorby: It was just the gates.

Golson: Just the gates.

Moorby: Gate and switch.

Golson: I see.

Moorby: Although, the first version of it, the marketing people refused to sell it, because it didn't have all the ASIC specify blocks, the timing checks and pin-to-pin delays. So they said, "Oh, well, to make it with a real product, we needed those features." That essentially doubled the complexity of it. So the actual hardware boards got really more complicated, and harder to produce. Harder to get right, and all of that. But we did it. I mean, it had a good speed to it. And then there was a big upheaval and shuffle of management in Cadence, and the new management didn't like it. Cadence was very definitely a software-only company. And they didn't like the sound of hardware.

Golson: For many years, Verilog was a proprietary language, as we say. And so Verilog was a proprietary language—can you talk a little bit about that? I'm wondering how it affected your work at all, and how did it come to no longer be proprietary?

Moorby: Well, obviously, the influence of VHDL, because VHDL was always an open language, and it had started to make some strides in coming out of academia, and actually they had defined it—there was a simulator, I think had started to work, and the industry was gearing up to say, "Well, this is the future." And so probably a couple of years before it was opened, the pressure, obviously, was mounting to open up the language. Customers began to really put a lot of pressure on the company to do that.

In the end, many people seemed to think that if it'd been opened earlier, that VHDL wouldn't have happened; or the other way around. If it delayed by another year, you know, Verilog would have gone away, and VHDL would have dominated all that. I've been thinking more about that recently. I'm not so sure—I think there was a flexibilityflexibility, probably of maybe two/three years where it didn't really matter. I think, eventually, I mean, obviously, if it didn't open up, the customers wouldn't have carried on with it. But I think if it opened up like two years before it did, or two years after, I'm not sure whether it would have made any difference.

Golson: From the customer's standpoint, what was the reason for the customers wanting the language to be open? They were looking for competition to Verilog-XL? Just some other simulator?

Moorby: Well, the simulators—I think Cadence was very surprised that a company like Chronologic had such a capable simulator. It definitely took Cadence by surprise on the simulator. It wasn't that the language was open for that. That was sort of, "Well, that can happen, we hope it doesn't. And we think nobody's going to do another simulator that good," kind of thing.

The opening up of the language is all about all the other tools that have to be developed to go with the main tools. So obviously, the main tool, we had the simulator, fault simulator, timing analysis, and Synopsys, we had the synthesis. But a lot more of the infrastructure of the business had to be developed, and we were all aware of that. Graphical display systems of debugging the design had to be developed. People had some very rudimentary kind of solutions. We had interfaces to all the main workstations with Daisy, Valid and Mentor. Different levels of ability and quality. But everybody knew that there were many, many tools that needed to be developed. That was the main focus and reason for opening up the language. And that's where all the pressure came from to say, "Well, the big main customers, you know, needed all this extra infrastructure to make it more productive."

Golson: And then fairly soon after that IEEE standardization became—well, I guess before that was OVI [Open Verilog International]?

Moorby: Yes. OVI was the main body that it was given to.

Golson: That's correct, that's correct. And there were all sorts of suggestions for how to extend the language, improve the language. And were you part of any of that that was going on?

Moorby: I think I went through several stages of my thinking of what I thought about OVI and where it was going, all that. I think I remember my first reaction to the announcement: "Okay, we’re opening, it's open." Well the main thing, there were going to be a committee of people now deciding what to do with this, that kind of thing. Well, good luck to them. <*laughs*> I never believed that they could go very far with it. Because my experience was, "Well, you can't really do much, unless you develop tools as well," and understand that play between what is needed in the tools and how to make the language good to express whatever you're modeling, to communicate with the tools. So I had real mixed feeling at the beginning to think where this would all go. So I think that they probably all-in-all in the early years, did a really good job, because they didn't extend too much. I mean, they did a few—some extensions in 2001 in that standard. But up to that point, they were really, I think very good at clearing up on nitpicks of—you know, there's all really detailed nitpicking that, okay, let's define this better, so that different simulators will be able to interplay a little better.

I remember that probably the smartest thing that happened in that—well, first of all, Cadence, I think, was sitting back a bit too relaxed. Because Cadence was not pushing on the external committee too much. Chronologic actually saturated the committees with all their people. Got in there and used it in a way that they accepted that Verilog-XL was what they generally called the de facto standard. But they were continually trying to reverse engineer how things worked. But on top of that, if they got frustrated with something, they'd just go and put pressure on the committee to define something in a particular way. Or relax a constraint, so that they could actually claim that they met the standard. I think there might have been one or two things they put in there where they were able to do, but Verilog-XL didn't do it, so they became a controversial kind of issue.

Golson: I see.

Moorby: I think, although there're many games that're played, obviously, in terms of trying to get the edge competitively. But I think Chronologic, those people, I think, did it really, really—gave a lot, you know, put a lot of justice into what they did. And really did a good job in trying to—not only standardize it just for their own benefit, obviously, because they were trying to compete with Cadence, but I think there's a very beneficial all around to all the other people trying to figure out what was going on.

Golson: So at that point, what products were you working on at Gateway, and after it got acquired by Cadence? So Verilog-XL, you're deeply involved with. What other products were you involved in the development of?

Moorby: After a period of time focusing on the hardware accelerator, and I think I had a few other projects going on, but nothing that can be defined at the moment. The biggest one, when there was a fairly big upheaval in management in Cadence. That was '91, I'd say, late '91. Because I decided to leave Cadence in '92. I think '91 there was a big upheaval of management. People like Prabhu Goel left. There was a big change of management. And it was right at that cusp where the industry was really demanding VHDL. Everybody was saying that VHDL is good enough, and it was. And it was going to eventually take over. But in Cadence, they don't have a very good simulator. They're nowhere with it. So I had my arm twisted, I think it's fair to say, to head that engineering team to do something about that. And so we had a project going on called VHDL-XL, which was basically Verilog-XL underneath the seams, but with building a new parser for the new language, but it'd all be, you know, trying to somehow sort of retrofitted with the Verilog-XL engine underneath. The big event of that is all the ASIC libraries were all in place, and that was seen to be obviously, its main differentiator, and advantage in the market. The problem is that it didn't have hardly any perform—it didn't have the performance that it needed. Another case, that rather than focusing on compiled code at the RTL level, you know, it was more, "Oh, no, we need to have a VHDL capability." But still based on the old Verilog-XL underneath it. So that didn't have the performance. And you put all the wrappers around it, and its performance is not very good at all. Not very competitive, that was for sure.

So when I joined that project, there was two guys out Midwest—forgive me for not remembering their names—but they had put a compiled code VHDL simulator together. And when I joined that project, I fairly quickly—it wasn't too long before I met these couple of guys at a conference. And they were benchmarking their simulator. I knew from benchmarking all of our own stuff, and generally how fast this stuff should go, that they had the performance. They hadn't completed the language. You know, still some way to go and so on. But I very quickly jumped onto the idea that the Cadence had to have that technology, because, you know, it was clear that in order to really win in the VHDL simulation market, you had to be—I mean, not only have a really rock-solid compiler, but you had to be the fastest. It was always about that. And there was no way that we were going to compete with that, unless we had that kind of technology. So I think the project became known as the Leapfrog and became NC-Verilog. So NC stood for Native Compiled, meaning that you go from the compilation stage right into the instruction set of the target machine. And that would be the way to get the high speed. They did that for VHDL first. NC, it became also the engine behind NC-Verilog later on. But I don't think they called it NC-VHDL, but the VHDL simulator from Cadence was based on that technology.

I mean, generally, VHDL is coded to a higher level, higher RTL on a behavioral level. Which is where compiled code really comes into it. There was, I think, a lot of controversial debates that went on as to whether you want that kind of compiled code at the gate level. There was problems in using compilation techniques at the gate level. But I think, Verilog-XL had largely sewn up its thing with the ASIC vendors, became the golden simulator, in general. And then the competition element from Chronologic, you know, where they wanted to be there and be the golden simulator, really came into that from a different angle. And they, obviously, had the strength of being more in-tune with the level for synthesis. So I think, eventually, that's how they took advantage of that. But we were forever struggling with the low-level, gate-level, switch-level especially, and had to get the speed at that level.

Golson: I have some questions about interesting things about Verilog syntax that my fellow designers and I have always, when we get together over a beer, it's like, "How come it's this way?!" And so let me ask—go right to the source here and ask you why, if you can solve some of these mysteries perhaps, or tell us what you were thinking.

So the first one is wire versus reg. And the reg keyword. Did you imagine that it would always be an actual hardware register, that that's what it was modeling? And, you can use reg as part of the model of a purely combinational logic. Did you imagine it would be used that way, or was that a surprise?

Moorby: I think there're two parts in that. The reg becoming part of combinational logic was what Synopys—

Golson: Was that your intent?

Moorby: No, it's really what Synopsys did with the always block.

Golson: I see.

Moorby: It became quite a useful way of writing combinational logic. But to use always [followed by] list of inputs, and then some behavioral code. And of course, because it was an always block, the left-hand side variables had to be reg, not wire.

Golson: Yes.

Moorby: If you complete all the branches, so it becomes combinational logic, then that's a really neat way of writing combinational logic. The original intent, actually, for combinational logic in conjunction with writing behavioral-level code, was that of starting out with a continuous assignment. But in the continuous, on the right-hand side, to call a function.

So in the function, you’d fill in all the inputs. You call the function. You can write all your behavioral code just as later on you wrote in the always block. And then the value that was returned, obviously, was returned to the wire in the continuous assignment. That's how, we thought, that combinational logic would be written, in conjunction with behavioral code.

So Synopsys never implemented that. For the longest time, they avoided the function, I believe. They used this other style, popularized the other style, and it's like, "Well, okay, that's just as good." So that became the way to write combinational logic with behavioral code.

Golson: So another thing we can blame on Synopsys is the model for a flop that has an asynchronous reset. And the synthesis tool looks for a particular template that you've modeled, you've written it and it says, "Oh, you mean an asynchronous flip-flop."

But it's not correct Verilog. It doesn't exactly model it correctly. And I'm wondering if you had a particularly way in mind that when you're dreaming up the language, "Oh, here's how you would write a flop with an asynchronous reset." I mean, you have to use a disable statement? Did you have a way in mind that that would work?

Moorby: Yes, the disable was sort of a general catchall for this kind of a thing that we knew that could come in and say, "Oh, that'll solve lots of different problems." Let me start out by saying, "Yes, Synopsys, it's all Synopsys' fault!" <*laughter*> I think they had a need to be able to somehow model all the different variations of flops. And figure out, okay, how do we synthesize this? Which was sort of a—first of all as you say, a recognition problem of saying, "Oh, it's one of these kinds. It's an asynchronous reset, because you coded the statement in a certain way." That was all good—that was good for synthesis. I think when it came to simulation, that code was hard to get to run really fast. I forget what Chronologic did a little later on that made it go much faster, obviously. But I think it was still maybe not fast enough, for low-level primitive like flops. So knowing there're all different kinds of flip-flops out there, needed to be modeled. And we needed the simulator to still go, you know, the fastest possible.

So that's where the idea of the UDPs came from. Obviously, for the combinational one, that was more of a generalized combinational UDP that—well, the whole design idea of the UDP is it all fitted in with the actual algorithm. So it was an incredibly fast-table lookup technique—single table lookup to have an input change and to evaluate the next state, and all that business. That went somewhere about 50 percent slower than the other built-in primitives like AND gates and whatnot. But the technique was very, very similar inside. And then in addition to the combinational UDP, so, "Okay, well, let's just do a simple sequential UDP that's got a single state to it. That could model pretty much all the flip-flops out there." So we did that, and actually, I mean, other than, obviously, some timing issues—'cause you're dealing with an event-driven simulator, it depends on the—especially with a sequential UDP, it depends on the order in which you evaluate the inputs. But if you kept things like the clock input, changes of the clock input clean, and not change at the same time as other inputs, you know, back to your timing analysis problem that you really want to do separately. But if you obeyed those sorts of rules, kept it clean, the simulator went incredibly much faster than anything else that you could ever do. At the behavioral level you would never get close to that kind of speed. So it was avoiding dealing with the performance problem at the behavioral level, which we knew we had, but couldn't seem to put enough resources into a big project to do the right thing there. So the UDPs was a—I think, you could say it was a sort of a stop-gap kind of a measure, but I think the performance it produced, it was the right thing to do.

Golson: And that came about, UDPs were added as part of the XL, Verilog-XL? That's when it came in?

Moorby: It was part of the XL very tight loop event-driven mechanisms. The same table lookup techniques as the rest. It was the same idea that you can only have one future value on its output. So the sequential UDP would just have a single internal state that it stored. That was part of the—it just kept a simple state of all the inputs, and its current state, internal state. And that would be used as the next address for the table lookup to decide what to do next when an input event came in.

Golson: So, tell me about expression sizing—the syntax for defining a constant, say, 32'hABCD…

Moorby: Are you making out you don't like that? <*laughs*>

Golson: I’m just curious. I think it’s wonderful, but did you consider other ways? I’m wondering where that came from.

Moorby: I’m avoiding the question, because I don’t remember.

Golson: That’s fine.

Moorby: I think it came from many different sources. I think it was similar to how HILO did it. I think in the end it was a balancing act between—because you don’t want the syntax analysis in the yacc to become—you don’t want to have to do any future lookups to figure out what syntax object you’re dealing with. So, you try and work out what is the simplest syntax for the compiler and what sort of looks okay and that can do the job. When I think about it, actually, I can’t think of any other good way of doing it.

Golson: Again, it comes back to the fact that you’re defining the language while you’re writing the compiler for the frontend of the simulator all at the same time. You’re thinking about all those things.

Moorby: Yes.

Golson: Why use begin-end instead of curly braces? That’s the first thing that all the C programmers complain about.

Moorby: So, you remember the occam language?

Golson: Yes.

Moorby: So that’s SEC-PAR. They were two key words and needed—I think it was an influence of wanting the two constructs to match, so it became like a natural thing to want to use keywords for both—and there was the Pascal influence. Pascal uses begin-end.

And the curly brace was used for concatenations. If curly braces were not being used for concatenation, I don’t know. Maybe it would’ve been a different history there. I agree with you. I mean, the begin-end was just written so much that even I was annoyed by it.

Golson: After a while?

Moorby: After a while of writing a little bit of Verilog.

Golson: One thing that, in my experience, Verilog designers cannot seem to agree on is how you represent an active-low signal. What’s the name for an active low signal? They’ll write things like “qN” or “qL” or “q_L” or “qbar.” And so, I’m going to the source here, what…

Moorby: I think I probably wrote descriptions of all those variations, but now, what I tend to try to do in writing software—if you’ve got two things that go together—a good example is doing complex arithmetic now and you’re having to write equations dealing with the real and imaginary parts of the complex number. So, I would tend to name the two with the same number of characters and try to make it meaningful so that when you looked at it, say, “That’s the real part; that’s the imaginary part.” The benefit of having them the same number of characters is the right-hand side expressions start in the same place, so you can actually—when you look at it, it kind of all lines up. But in hardware, whether they ever think of it, if you have a q and qbar that—in the right-hand side, they’re not going to—well, if you write an expressions for all of these, not that you necessarily do, but—so, I personally do not like underscores, but I think—in fact, when you go through many machines and editors and screen displays and printouts and whatnot, about half of that media, that the underscore can get lost and it’s hard to see. Some of the printouts, if you ever remember them, but they kind of merge with the line underneath and roll down on a CRT screen that—hard to see and all that and you think, “Have I got two variables here?” because you can’t see the underscore. So, I sort of went against underscores a long time ago. So I prefer the style of doing capital letters.

Golson: Again, I think that’s a Pascal—is that a Pascal typical thing?

Moorby: I don’t think so.

Golson: Not so much?

Moorby: I find that a lot of professional programmers these days prefer to not use the underscore. It’s a mix. You see all the big mix of it, but I personally like the style of not using underscores and using capitalized letters and, in fact, build up multiple words with the capitals for each word.

Golson: So, why have “always”? You could just say “initial 1” or…

Moorby: <*smiles*> All right…

Golson: Okay, now we’re getting him going! I mean, it’s like syntactic sugar there, but why?

Moorby: Well, basically it came out making the compiler easier. <*laughs*> In a simple form of yacc that you don't want any more than one token lookup ahead to know what you've got. So you use a keyword to say, "I have this kind of block." Now you know what to expect next. But, thinking back, I mean, thinking with hindsight, I think out of the keywords—you've got "always," "initial" and probably "forever"..

Golson: "Forever."

Moorby: …which are the main loops without any extras go with it. First of all, Scott Sandler came up with probably the right alternative to what "initial" should have been, and that was "once." He said, "Phil, you shouldn't have called it 'initial.' It should've been 'once,'" because "initial" is sometimes used with a timing control within it, so it isn't necessarily initial. So, you can have "initial" this, that and then do "forever," right, so it will go on forever. So the keyword "initial" probably was quite correctly the wrong word, so "once" would've been good. And I think between "always" and "forever" there's one of them should have gone and maybe both. So, you can always write "initial" and once you get into it to write "forever"—I mean, there's many alternate—maybe out of those three keywords you could've only just—try and work through—tried to have it so there's only one, maybe two.

Golson: What about, again, the software people always are like, "How come you left out increment, plus-plus, on a variable?"

Moorby: Well, so, the operator comes from C and all expressions in C can also be assignments. So, an assignment is an expression in C, and the thing with plus-plus and minus-minus and the variant whether it's before or after and all that business is basically both it reads the variable and writes the variable, so it's really an assignment kind of a construct. And in Verilog, we wanted to obviously keep things simple, and there wasn't anything that was both something that read variables and wrote variables within the same expression.

And I think it was my—let me throw in my own personal taste—that it's a little bit like all the precedence order within expressions. You've got these seasoned software types of engineers, who will learn all the rules of operator precedence and now write an expression line with no parenthesis, and so, well, isn't it obvious what the precedence is? Don't you read the manual or the LRM to know what the precedence order is? Well, I'm of the type who says, no, half the people out there are not going to learn those rules, and what is important is not who writes the code. It's who reads the code, because if you've got code that'll last for a long time, you have more people reading the code than you have writing. And if you have half of those readers who don't know precedence order and if you do allow them to touch that code, what are they going to do with this thing? And I think the plus-plus, minus-minus, I think I'd put them in the same category that it's hard to learn the rules of what you can—I mean, obviously when you put it in the third place of a for-loop, you just simply—I just want to increment it each time round and it's simple. Then there is an argument; however, I come back with that is—although the construct didn't actually get into Verilog in the first place, but I think it did eventually—to have a plus-equal expression. And when you think about it, that's three characters versus two. So, in my mind, it's like why have a whole extra construct that you have to learn and then learn all the side-effects and whatnot, when plus equals one actually works quite just as well. So, I don't think it was a mistake for that construct, the plus-plus operator being in Verilog in the first place, but I think I would have liked—the mistake was not having the plus-equal and the minus-equal and maybe some of the other variations of that in the language in the first place, because otherwise, if you've got especially a large concatenation of signals and that and you just want to add one to it, you really would like to just put plus equals one. I think that went in eventually, but it should've been in there in the first place.

Golson: Let's talk about nonblocking assignments. So, nonblocking assignments were not in the language initially. If you look through all five editions of the Thomas Moorby book, it's not until the third edition that they start talking about it.

Moorby: I never realized that.

Golson: You have to be like me and have all five editions. <*laughter*> And from working in the industry, I know what was going on from the user's perspective of why it came about, but from your standpoint inside Gateway, why, and what forced it to happen? What was going on that nonblocking assignments…

Moorby: In behavioral code when you're writing assignment statement after assignment statements, in classical RTL languages versus software procedural languages, you go from one extreme to the other. Either the left-hand side is delayed by whatever you want to call it, a delta loop, a clock cycle, so everything is a nonblocking assignment, or everything is a blocking assignment and it's procedural: the left-hand side is updated before you go on to the next assignment. So, go back to HILO-2. It followed the classical RTL model, where everything was a nonblocking assignment. And that was good and it all fitted in with the finite-state machine theory.

When it came to Verilog initially, the debates I had with Chi-Lai and going through to try to minimize language, do absolute minimum, we know we wanted to make the behavioral code more procedural-like. So, I said the assignments are blocking so it's like any programming language, but how do we deal with—how do we model a classical finite-state machine? And we said, well, you assign it to a reg and you'd have a continuous assignment with a delay on it with the reg on the right-hand side assigned to a wire, and that would be your delay. So, one day—I almost remember it quite clear, actually—a guy called Dave Rich came in. He was the one who was mainly interfacing with Synopsys and customers, trying to make it all work well. And he said, "Phil, I know we don't want to do too many enhancements, but this is really painful." So he went through that every time you wanted to write an assignment in the finite-state machine for synthesis, he had to go through this—write it to a reg, then have to somewhere else go write a continuous assignment to get the delay effect. So, he said, "Why can't you just delay it?" So, I went through the usual, "Well, what if you have a mix of—so you call them procedural assignments, say, and RTL assignment. Well, what happens when you mix them? What are the rules? Don't you get confused?"

And he sort of went, “Well, you get confused,” but the advantage obviously was pretty clear that we were missing out on the classical RTL expression with finite-state machines. So, Dave Rich pretty quickly convinced me that that had to go in. So, you remember our high barrier of entry for any enhancements, well that one went..

Golson: That one made…

Moorby: That one got in pretty easy once it was understood where the synthesis tools were going and how they really did need it, and everybody was beginning to write it that way. So, once a decision was made, I said, “I know how to do that, not too difficult.” I mean, it involves an extra step inside the simulator to actually post an event, to delay an event to update the left-hand side. So, it slowed it up a tiny bit.

Golson: And did you come up with the syntax for that?

Moorby: I think the syntax was pretty—I think that was pretty fast. It was pretty obvious what to use. I think it’s the same one that HILO-2 used. I can’t remember. HILO-2 probably just used the equals, because I think everything was a delayed RTL assignment.

ISP actually had—I think it had the ability to put a keyword called “next” on the end of the line, I think, to change it from being a nonblocking assignment to a blocking assignment, and I think I remember talking to somebody. I said, people get very, very confused in ever trying to mix the two.

Golson: So, why use bit vectors to represent signal strength? Do you understand that question?

Moorby: No.

Golson: Well, then let’s move on. <*laughter*>

Moorby: Logic strength was always there were only scalars with all the different logic strengths for MOS gates, for switch level, but you could only put strength on a scalar, whereas vectors, that was generally defined as being a variable that had more than one bit, that could only hold four values per bit. So, I think it was one of those trade-offs that was—I think, worked in practice. I think VHDL went all the way to being able to do whatever you want it to do, but I think what we found was actually quite a good balance between high-level modeling, where four values was sufficient, and the need at the switch level to be able to do all the strength modeling, which was limited to a scalar object.

Golson: So, speaking of VHDL, I’ve seen former VHDL designers come to Verilog and they try to write Verilog like VHDL and they try to make configurations. Have you ever seen that? No?

Moorby: Not too much. Well, you tell me. Are they very successful? Was it useful? What were they doing?

Golson: It’s frustrating that while they’re trying to—I mean, the idea of VHDL configurations, where you have this way to externally map how it’s constructed and they want to write their Verilog code the same way, so they go to these absurd lengths to try—“Well, I’m going to use `includes and the file names will stay the same, even though the file is different than”—so..

Moorby: Didn’t somebody do a really good job on using Perl to…

Golson: I’m sure.

Moorby: Actually, I’d worked on for a long time and actually wanted to do the enhancements for it, to do—using a procedural language to create module structures, right, which I think it—the essence of what configurations give you. And you need to be able to parameterize it in such a way so that you can have a for-loop, for instance, to run down a vector of modules and connect them up in clever ways. So, most of it will be very highly structured, so the code will be quite condensed, so at the end of the for-loop you do something special and you can have an if statement in there to do something special. But this is just generating structure, so it’s nothing to do with behavioral, procedural code. It’s procedural code in the compiler, sort of. I worked on that for the longest time and thought that that—that should be where the language went next. It never really went anywhere until the…

Golson: Well, eventually the generate statement was added.

Moorby: But that’s a very limited form of what you wanted, and it did maybe 90 percent of the job. I lost touch with how useful the generate and whatnot became. Am I right to think that some of the extensions using things like Perl became more popular?

Golson: Yes, until the Verilog 2001 and eventually SystemVerilog…

Moorby: So they did a good job with doing the generate.

Golson: Where did the PLI come from, and at what point did that get added, because that seems to have a huge impact on the success of Verilog that there was this PLI. So, can you explore that a bit?

Moorby: Again, learning a few lessons from the HILO days and forever been confronted with situations, where the customers say, “Well, I want to just have this extra feature, so take a couple of wires here and there and do something miraculous with them and inject a value in over here,” kind of thing. I said, “Well, we can’t keep doing enhancements for all sorts of different customers,” so HILO-2 and HILO-3 never thought of the idea of making it kind of extensible. There was no concept of user-defined functionality. So, realizing that right at the beginning, I said, “It’s going to be extendable.” I mean, I hadn’t worked out the details on Day 1, but what was fairly obvious where the extensions would come from, started with function like $time, $display. What were the others? I can’t even remember some of it. There were not too many built-in ones like that, but I started with the mechanism by which sort of an API to those functions that could be off on the side of the main simulator and then very quickly created some of the TF routines. The built-in ones, $time $display and so on, were obvious ones to use. Some of the TF ones were as a mechanism to get handles to variables so you can read a value. The main job is that at a particular time to jump out into the C world, read a value and write a value, and then once you’ve got that, you can say, “Well, why don’t you go and do it yourself?” So, that’s where the TF routines started. I can’t remember the set of them. They just barely did the job, I think it was fair to say, and it was effective enough that it became extremely popular. People were doing all sorts of things with them.

In fact, some of the functions like the programmable logic arrays… It allowed groups within Gateway that were off from the central group of the simulator, so they could with this API interface to the simulator, were able to develop that functionality. So you could do that within your own company, or you could say to the customer, “Well, why don’t you do that?” depending on who the customer is and how much you wanted to keep him happy. So that’s how that grew out, and people just thought, “Wow, this is powerful. I can now do anything almost.” And, of course, you can’t do everything, so you always want—you always need enhancement. But the naming TF and then ACC was all about not wanting the library names to clash, because the ACC routines was we’ve got a basic set of TFs but we really want to start from scratch and do it again and do the job properly. So they defined a set of ACC routines to be able to do what people in general were wanting to do but more powerfully. And then the whole thing grew more and more into more and more capability. I guess there’s no end to the sort of things you really want to do.

Golson: Referring back a bit to the nonblocking assignments, I mean, another crazy thing users started to do is use #0 to play games with the event loop, and so what did you think when you first saw that and saw people using #0?

Moorby: I looked at that and I said, “Why are they doing that? What’s the point?” thinking it was obviously redundant. So why slow the simulator up? But you dig into it; obviously there’s always a reason and then thinking, “They’re trying to”—I think that’s when clock logic started to get more than trivial, and so you got a clock fanout and you’re trying to delay the signal going to say one flop versus another, whereas the hardware designer knows the order of events, only he said, “Well, it’s obvious that that one gets triggered before that one.” And some clever customer figured out, said, “Let’s try #0,” and it probably worked. “That’s it. I got to put #0 over it.”

Golson: I call that voodoo Verilog. It’s like, “It worked once. We’ll keep using that from now on.”

Moorby: And then got very upset when it stopped working with somebody else’s simulator or an update to the simulators or whatever.

Golson: When you first started working on Verilog with Gateway, you said you had your Sun-2 workstation that you were doing the development with. And how did your development environment change over the years? Do you recall what machines you used? Were there any software environments that you used to help your development? I’m curious as to how—I mean, we started with paper tape and 16-bit machines, and how has it changed?

Moorby: Well, I think that the Sun workstations got better and better, obviously got very taken up in speed, which is great. The Sun-2 really didn’t have a good windowing system, so I think that the next big jump were to have real windows, that you could have lots and lots of windows open on your screen so when your stack got pushed to a new task, you still have the windows open behind so you could go back to it. That was a big jump in productivity. I think I just remembered having to jump from machine type to machine type doing porting, so I think I had experience with all different kinds of machines. They all had their ups and downs, little quirks, and certainly the IBM 360 was the quirkiest of all.

Golson: I can imagine, very different from the other environments. I mean, it’s one thing to go from a Sun to an Apollo to a VAX, perhaps, but then…

Moorby: Verilog was all written in C pretty much. There was some assembly, of course, but that was a special—well, even the XL was written in C, so it could all be ported. There was no way to port a machine unless they had some sort of C compiler, so that was the first question to say, “Well, do you have a C compiler?” It took a while before everybody ended up with a C compiler as well, but the first C compiler in the IBM 360 was from Waterloo. I think that was the first one, quite a good, capable C compiler, so we tried that out and it went okay, but the assembler code that was generated from the C was absolutely terrible. So I said, well, you can’t go to the real big mainframe computer without going really, really fast. We knew how fast the machine could go, which was probably the fastest machine in the world at that time, one of them. So, we started a project to write the assembler code for it, and the good news is I didn’t do that. <*laughs*> Somebody else did that.

Golson: So did that get completed? Did you complete that project?

Moorby: Yes, and it went blazingly fast. It was very good. And I think AMD and a few others took big advantage of it, but it wasn’t too many years then after that, where everybody started to think, “Well, these mainframes, they are not only incredibly expensive, they’re not actually very fast anymore,” and all these headaches of using it, because you didn’t have access to it on a continuous basis, so it was largely time-shared and the workstations were beginning to take over.

Golson: Your test environment for Verilog, did you have a suite of Verilog source code or Verilog language files that you ran and gate-level net lists? How did that expand? How did you get your tests?

Moorby: Incrementally right from the beginning. Of course, actually created all the language constructs and to just hammer out lots and lots of test cases that would try constructs in as many different combinations as you could think of and just keep on writing them, producing them. And then every time we’d have a—certainly any kind of a—well, every bug that came in that had—you’d always ask, “Well, we need the source code and we need to reproduce it,” that went in and a lot of benchmark went in.

So, although it takes a while, you’ve got to keep focused on knowing the value of the test suite and keep building it, and you just continually get more and more complete over testing every possibility. So, every time you compile the simulator, you would run through all the test cases, and of course that would be growing and growing, but initially it would probably a few minutes to run through and then it just continued to grow obviously to be many hours and then overnight. It takes a long time to run through all the regression tests. I think with any kind of a big software project like that, if you don’t build up your test cases like that, then you start having really bad bugs. The customers will find out eventually.

Golson: You’ve mentioned VHDL, but when did you first find out about VHDL as a language, and how did that impact your initial development of Verilog?

Moorby: Well, first of all, VHDL had been talked about way back. I mean, we were hearing it from when I was still in England. It was probably somewhere in the…

Golson: Seventies, then.

Moorby: …middle of the HILO-2 project, where the Americans were making waves about this great new language they were thinking of starting, but they had kept it very insular in the US. They didn’t want too much of—because it was all paid for by the Ministry of Defence…

Golson: Department of Defense in the US.

Moorby: Department of Defense in the US. So they wanted only US-based companies to really benefit from that funding, because in England—in fact, I think we might’ve even put in some sort of a contract to try to make HILO that language. I vaguely remember Gerry Musgrave wanting to do that but got nowhere with it, because it was like, “Well, this is a US endeavor, stay away.”

So, I was aware of it way, way back and read some—I think it was some conferences had started on it. And then I think the year that we had launched Verilog—so that would’ve been, let’s say, ’85—I went down to Washington, D.C., for a VHDL conference to figure out, well, what have they got? Have they got any ideas I haven’t thought about? I'm trying to remember whether there were some constructs that I may have taken. I don't think so, but I remembered waiting after all the talkers are finished, and all the mainly professors milling around and talking with each other. I went, "Do you have any examples that I could get for VHDL?" And they all looked at me, "Examples? What do you mean? We haven't thought of any examples yet. We don't have any," Oh, really? With that answer I kind of think, huh, they've got a real long way to go. I mean not only to not put examples together to drive the design of the language, but there were no tools. You could have asked them, "Well, how is this going to work with fault simulation, and have you ever thought about timing analysis or not?" Say, "Oh no, well, the tools they'll just drop out. They're not important to help design the language." Okay, interesting. So I think what eventually came about is that the language from a language point of view was very strong. It was a strongly-typed language, and very well put together for what it is. But as we all know, the struggle with trying to make the tools work with it took a very, very long time. So that was to Verilog's advantage, I guess.

Golson: So when the so-called language wars were going on in the '90s, and the conventional wisdom was, well, VHDL is going to win…

Moorby: I believed it too.

Golson: You believed it too.

Moorby: Yes.

Golson: Because by then you were working in VHDL.

Moorby: I was working on it, that's right before leaving Cadence, yes, I worked on it. The language was good. I mean, I liked the language. Once you got over the learning of the about what, three or four chapters of the syntax.

Golson: <*laughs*> That's right, as opposed to the two pages of your book.

Moorby: And you got familiar with that, I thought, "This is a good language." You see it was obviously based on Ada, which was a very capable language. Okay, this is good, and I think I had convinced myself that you could eventually make all the tools work with it. I thought, "Okay, well yeah."

I think maybe a lot of people believe that the language—or the thought about putting the language together—is the driver, and the tools just follow, but that's not true. Certainly with VHDL that may have been the case, and that caused it to create lots of problems for the tools. Eventually I think they worked out all the details, but I could never see how you could approach this whole thing in EDA without primarily allowing the tools to drive what you did, and then you just did your best with the language. You still want a good language, obviously, and you want to keep on working on it, but the value that the companies put into language development was abysmal. I remember going through the Gateway to Cadence days, after the initial development as the language was being opened. Cadence had—the mentality was not to put anybody on language development, nobody. Nobody was a “language person.” Everything was fixing bugs, say, in the compiler, and making the simulator go faster, developing libraries and building tools. The language it was like, "Well, no-no." By then it was like, "Well, OVI is in charge of that now, and anyway the world is going to VHDL anyway, so we're just working on tools," because that's where they saw their revenue coming from, and that was the problem. Language had no value. Everybody was sort of, "Yeah, okay, but the language needed improvements," and everybody was sort of looking at everybody else to do something. So all the companies were saying, "Well, isn't that responsibility for OVI now," and then it became all the various committees, so that's their problem. I don't know.

In the software world can you imagine what it's like in the software? I mean who drives the next best language, so to speak? What was it that drove the design of C++? Was it the tools? There isn't an equivalent, an analogy with the set of tools in that space. So they did the best themselves from a programming point of view to develop the best programming for its own sake not to make tools better or to focus on anything that had to work with it. Obviously they had to do it so that it ran fast, but there's no equivalent like how do we make the synthesizer work or a fault simulator work. So I think in the EDA space I think people are too easy to forget that the tools were the revenue drivers, and we just did our best to put the language around it.

Golson: Speaking of the C language, as the language wars tapered off between VHDL and Verilog, the C language, and using C for hardware description started to take off, and there was SystemC, for example. So what are your thoughts on SystemC?

Moorby: Way back, if I can remember, my thoughts were, "Well, if VHDL sort of takes over, that's fine, or whatever." In my point of view it wasn't a real big deal, because I knew it was all about the tools. But I knew that the only other possible candidate, really, was C++. So when I first saw SystemC I thought, "Oh, that's an obvious way to go. C++ is very, very capable, a great programming language, but how do they do the low-level event-driven gate-level delays and whatnot." Well, so I had the opportunity in Co-Design

[Co-Design Automation, Inc.] actually, to jump on that and figure out how it—because there was a public domain SystemC system simulator.

So I got a hold of that and figured out that it was based on a threading model. It was a threading model that's got no parallelism. So every separate task or whatever you want to call it, every separate thread in the hardware description language would need its own stack. So when you came to a blocking statement like a delay you had to freeze that stack and jump somewhere else to something else that needed to run, load up its stack and start executing. There's a serialized threading package, I think it was called QThreads that was used underneath SystemC, that provided that task switching mechanism. I thought, "Oh, at the low level that's not—" I knew, actually, like that's not going to go very fast.

But in terms of pushing up at a higher level above even the RTL level, I didn't think it would ever replace what I always thought of as the middle level RTL, what Synopsys Design Compiler and XL did. In the push to get higher than that, I thought using C++ was a great idea. I always thought that the interface between C++ and Verilog, that’s where there should have been language people working on that way back. That's the thing where the company saw no value and didn't put any effort into it. But to be able to go easily from, say a Verilog behavioral code into C++, what I always thought as the test for it is to be able to create an object, say a C++ object, pass it into Verilog so the Verilog side knew it as an object, and was able in the Verilog code to be able to call the method, call the method over on the C++ side and vice versa, because objects were the center focus of that extension. It was a hard problem to be able to do that, but it didn't go anywhere. People just didn't think it was a problem. It was more of, "Okay, let's create this whole new language, say take SystemC, and then try and figure out how does it play with the other tools. Well, I think the main reason that became a problem is that SystemC was initially put out for free. There was a free simulator to be had. So the whole user base—you know that that really disrupted the pricing model of EDA, I mean seriously. It took the longest time to get over that issue. The biggest problem is how do the tools interplay with each other. Do users have to completely jump over into the other way of doing modeling? Well, that wasn't going to happen. It causes quite a disruption.

Golson: Gateway was acquired by Cadence, and you stayed at Cadence and worked there, and then you decided to leave Cadence and completely leave EDA. So how did that come about?

Moorby: Not intentionally. I actually left Cadence and didn't know what I was going to do. I actually decided, "I'm going to leave first, and then work out what I was going to do." I fairly quickly got into the field of multimedia and a little bit of sort of animation kind of work. It was really very interesting work. It was programming and solving hard mathematical problems. It was a very good diversion.

Golson: So you mentioned mathematics. So back to your original studies, and so you were able to apply some of your mathematics background to the graphics and video work you were doing. So that was at Avid, is that right?

Moorby: Not quite. It was actually a bit more complex than you are making out, because there were multiple companies involved. I did join Avid for a while. I had a project for about four months in the field of video editing a little bit, but that didn't take off. It was short lived, and then went somewhere else, a company called Symplify where I tried to—we didn't take off as a startup. There was no money raised, but the idea was a new language. I guess I was being way naïve to think—it was at least about five years too early. The industry basically didn't want to know.

Golson: Speaking of Symplify was that your idea to start with, or did you join a project that was underway?

Moorby: It was just a small group of us, who put the ideas together, and we tried to raise money, but we never achieved raising any money. I think that was where the EDA was—it was getting more and more difficult to raise money for EDA projects. Actually, there were two parts. There was a language play and I said, "Oh, I've got an idea for a great new language," and everyone said, "Well, there's no money."

Golson: A language for HDL, an HDL?

Moorby: For HDL, yes. The idea wasn't just the language, obviously, it was the ideas of tools behind it, but it actually morphed pretty quickly into their building an emulator, a very high-speed simulation, and probably didn't have the right set of people in the startup. We got close to raising some money and getting some traction, but it didn't happen. So very quickly I actually went back to Avid for a very short time, but then Avid went through a big turnover, an upheaval. I left there and joined a startup, which was sort of, well, a lot of the founders were from Avid, but it was centered on special effects in animation and video. That's where I really got to use some mathematics to solve a 3D problem.

The general idea was to have the video images, the images from the video where the camera moves in 3D space taking a picture of a scene, a 3D scene, and through all the images to reconstruct the 3D world. So you end up with knowing exactly the camera path, and in every image the depth of every pixel so you can actually recreate that 3D world, and actually introduce a special effect to a 3D constructed object and actually move it around the scenery and have it interact with the objects. So I worked on all the mathematics to be able to reconstruct that, and so that was a great project. That was another startup that didn't eventually make it. They raised some money...

Golson: What was the name?

Moorby: That's embarrassing.

Golson: Well, it didn't make it after all.

Moorby: It didn't make it.

Golson: But you were about to say they raised money?

Moorby: They raised quite a bit of money, actually. The problem was way harder. It was way ahead of its time, and the problem was very, very severe, and it was hard to get the solutions that we came up with to solve the technical problems, and the marketing to get those two just right so that you could actually start selling products. It wasn't an easy space to work in.

Golson: So in the EDA world...

Moorby: Then I went back to Co-Design, back to EDA.

Golson: That's right, back to EDA. Back when you were first at Cadence at that point you had this rock star status in EDA, I think. Everyone knew your name. You were Mr. Verilog, and you head off and go into a totally different industry.

Moorby: I chased hard problems. I just liked picking up on hard problems and going away and spending years to solve them.

Golson: Back to solving puzzles back when you were a boy growing up.

Moorby: That's right.

Golson: One thing you had argued with for a while was you thought that languages were limited because they were stuck in the ASCII character set, and if you expanded beyond that you might get a more expressive language, or easier, or… So I would love to hear you expand on that. It sounds very much like APL.

Moorby: I think you're referring to the Symplify ideas where I thought that looking at the Verilog descriptions that it was generally way too verbose. There was a lot of extra typing to do, and attempting to do graphical entry systems, not structural entry but graphical entry that did the same kind of thing, but in a much more user-friendly—so you could enter the code much faster. I explored that kind of ideas in several different ways and nothing was working.

So I had this idea that if you could just do a bit more than straightforward ASCII text—behind the scenes you'd represent the constructs in some sort of compact form, but a bit more than an ASCII text. Have an editor to go with it that would give you all the feel of graphical entry, so you could actually code it really quite fast, and also it would look much more—well, the actual structure of the code would be much more obvious. So instead of having—in fact, our measure was how much could we compact the standard, and of course it was all about synthesizable code, of course. How you could take a normally written Verilog text code and our measure was that we wanted to get that to be at least 3X more compact if not a great deal more. So I was experimenting with some ability with graphical characters. So instead of typing always all the time, or @(posedge) and all that, you'd actually have very, very simple characters to actually represent those statements. I certainly didn't want to go as far as an APL keyboard, but if you could do things in a very, very simple way that you'd get the speed of entry and the visual look you would get a great deal more code on the screen. I believe very much the more code you could actually see at once the less bugs you would end up with. You could actually see the structure of what you're designing much clearer. So it was all about that.

And, in fact, part of that compacting the code was—remember what we talked about where the discussions of configurations started—so what I wanted to also expand in is using a procedural-like language to express structure. There are a lot of pretty good neat ideas, actually, in that. I didn't take it far enough to know how general it could become, all design styles, you could sort of write it so much more in a compact form, and write it much faster. But there were some examples where the code really did compact, so instead of writing you'd define a module and then instantiate it a couple of dozen times with some of them having slight differences. Then you'd replicate it out text-wise. Then you'd have a very compact succinct way to write it procedurally to generate that stuff very much more effectively. So you'd actually write it much faster. So I always believed then that you'd actually have a lot less bugs, and you'd be able to read it so much better. But, as I say, nobody wanted a new language. That came much later.

Golson: Well, let's talk about that a little bit. When did you first hear about Co-Design?

Moorby: Other than the times when Peter and I were sort of considered to be in competition with each other, so we kept clear and not talking about technicalities, but as soon as we'd get away from that situation we said, "Well, come over. Let's talk. I've got some ideas." So when I was out of the industry, and Peter and Simon had started the initial ideas that became Co-Design, I was consulting for them. We had just a couple of meetings together, and I gave them some ideas about where I thought the language should go, and Peter, of course, had many ideas. So they were working for a good number of years on what led up to the—they called it the SUPERLOG. That's what that became. It didn't become all of what SystemVerilog is today, but the design part of the extensions of Verilog did go in there.

Golson: So how did they eventually persuade you to come onboard full time?

Moorby: I wanted to leave the company I was in. It was a failing startup. I could see that way, way too many mistakes had been made, and it wasn't going to make it. It was another one of those obvious decisions.

Golson: So when you came onboard with Co-Design, what were your tasks? What were you working on specifically?

Moorby: To develop the complied code for the simulator, actually.

Golson: So another simulator?

Moorby: Oh yes, another simulator. They put together, actually, a very, very capable interpreter, interpreted simulator to go with SUPERLOG. We thought it was pretty optimal. I think we had some ways to measure against Chronologic. I can't remember how. I think something told us the speed differences, and we knew we were about two or three times slower than the Chronologic, so we had to do something about that. So that was my task to do something about that. But that's a long story where that went. Part of the problem was the approach to compile code. They had some false starts with it. We can get into a story about where that went, but I kind of suspect it's going to be quite an offshoot.

Golson: I'd love to hear it.

Moorby: Well, I think it's to do with the big lesson, which actually goes right back, actually, to—it was all to do with the issue and problems with cache memories when you don't fit into cache. So you know that most people...

Golson: It's bad when you don't fit into cache, yes.

Moorby: But software engineers learn the lesson one day in their career, that if they don't fit into cache memory, things start to slow up. And this effect, actually, I could not work out going from—let me get this straight. I think it was maybe the Sun machine to a VAX machine or some other—I think it was the high-end VAX machine, and then I saw the same effect on an IBM mainframe. When you get slowed up by not fitting into cache memory it's fairly subtle, and things are not obvious as to what's going on because it's so far down in the hardware that you really have no idea what it is that's slowing it. So in porting the XL algorithm onto some machines I had noticed—I think the VAX was the first where I noticed the effect, because I very carefully measured how many clock cycles per event the machine took. Some machines would be—I can't remember the numbers, like 10,000 clock cycles or something. No, it wouldn't be 10,000. I can't remember the metric I used, but I think I had a certain benchmark where how many gates I could actually get through in so many seconds. The gate description was very compact. I could notice that the events per cycle were very, very high, but going on to some machines where the design—you could simply take a design and instantiate it out. That's obviously going to take more memory. Our assumption was clearly that some machines were going two, three times slower than it should. We were thinking, "Well, why on earth is this happening?" And it took something like a year or more to keep on trying to analyze these new machines I was working on, that it was a cache memory problem. The speed of the machine was relying on the cache memories more and more at that point in time that if you didn't fit into cache memory you were going to go slower, and this effect grew. Over the years the use of cache memory became more and more, so it's quite a lesson to learn how do you fit into cache memory and not go typically three times slower if you don't.

So when it came to this complier at Co-Design I was taking the output of the interpreted code that had blown all the interpreted code out to its full output state for the simulator to work on. So the equivalent is that everything was instantiated out. So it was a big set of opcodes that the simulator worked on. I took the output of that and translate it into C, so low-level C, and compiled it. We knew that the number of instructions it was creating was about three times more optimal, but what I was simulating wasn't going any faster, and it was the same old story. Because of what generally people call code bloat, which means the code was three times bigger, so to get that through the CPU it put a lot of pressure on the cache memory, so it was stalling because of cache misses. So the code should have gone three times faster, but it actually was going three times slower because of the cache memory. I think people are still learning how to avoid such issues even today.

Golson: Frankly, many people, I think, don't bother. They just, "Well, next year I'll have a faster computer and...”

Moorby: Or more cache memory.

Golson: Or more cache memory, that's right. That's right. Why was Verilog-XL so fast? So it was because you spent the time to make it faster.

Moorby: Well, and the data structure was extremely compact, so most of the time it would fit into cache, but of course when you go into especially 50,000, 100,000 up to a million gates you're really wringing out the simulator having to scan most of that memory every cycle. It's no longer going to fit in cache.

Golson: So let's back up a minute, and go back to when you're at Gateway and Gateway's acquired by Cadence. So can you talk about the business side of that, or even more interesting is how that affected you as an engineer, this small company getting acquired by this big company.

Moorby: I don't know how well known it was, but there was Joe Costello of Cadence. He wanted to acquire both Synopsys and Gateway at the same time, so he wanted it all. It was quite naturally that the

Design Compiler and Verilog-XL went together, and he didn't get his way with Synopsys. I think a lot of people know that, but he did acquire Gateway. When was that? Right at the end of '89, so I think it was quite a smooth transition. I was somewhat left alone to pursue and continue with the momentum with the Verilog market. We did our best to integrate it with the rest of the Cadence tools with different degrees of success. So that went on for something of the order of about a year. I don't know largely what happened, but Prabhu Goel left. I believe he fell out of—had a difference of opinions with Joe Costello. I think part of the problems, of course, is that Gateway was on the east coast centered in Lowell just before I left, but there was a big office that was created in Chelmsford, which still actually exists. There was the merger also with Valid that had an office across the road, so it all combined into one big office in Chelmsford. Prabhu left and there was a typical big company upheaval of management, people leaving; all sorts of new people coming in, and largely quite a disaster, I think. Politically, it was unclear where anything was going. A lot of ill feeling probably was created. I was largely well out of any of that, but I continued to give it my best, and that was the time when I started to lead the VHDL engineering team. There were lots of mergers and the merging of teams from Valid, from the rest of Cadence, and the old Gateway, was—some of that was messy, some of it was okay. But it was an upsetting time, obviously, because you didn't know where it was going. Your tools were being played around politically, and there were all sorts of ideas of where it was all going that you didn't always agree with. But I—I think I've always been a type to always explore new projects. When I've been on a project for many years, and got to a point, I usually like to, okay, what's the next big problem to go to. And whether that's leaving a company or not, there's always been—sometimes it's with the same company, usually it's with another company, usually also a startup company to jump into a nice great big hard problem to solve and so on. But I did give a fairly good shot—I think we went into some details earlier about the VHDL work. Where that needed to go for Cadence was fairly straightforward, from my point of view, there was no hard technical problems to solve, the language was there, the tools were around and it all just needed to work together better. So I left in '92, and I—on the east coast—also remember that, during this period of about a year leading up from the time when Prabhu left, to other management coming in and—some of the managers really messed up, others were okay, and on the east coast, there was a couple of different managers come in to lead the east coast. One was a disaster, the next guy was good. I think that, for me, the challenge went away. So that was a point when I decided just leave the company. And made a decision to leave the company first, and then decide what to do after that. I knew something would open up. I'd go and find a new challenge, and I did pretty quickly.

Golson: Let's jump ahead to Co-Design, and you're working on SUPERLOG, working on a simulator, on the language—the idea of SUPERLOG starts to take hold among designers, that "Gee, we want this lots more than we want SystemC." So walk us through how that, as the interest in that SUPERLOG was ramping up.

Moorby: We had a marketing strategy that we had thought about. Well it was a combined marketing strategy and technical. It's to bring into the fold, so to speak, all the main characters on the Verilog standardization committee. We knew that if we got them on our side to believe in where SUPERLOG was going, they would, in addition to what we were doing in Co-Design, would see it through and tell the world that this is where it's all going. And at that point in time, of course it—and I can't remember the different years in which the VHDL versus Verilog played out. I think, for the longest time, we all believed that VHDL would win out. But then it didn't, and everybody sort of analyzed, “Well it's not going away, why?” you know, so—“Well, Verilog is much better!” kind of statements. And it was always a faster simulator, that probably was one of the lessons that took—I think the synthesizer, but you could say more about the synthesizer, I think on the Verilog side, remained a much better way of doing hardware design than the VHDL side. Whether that's just a language preference, I couldn't say. I think, from the practical tool point of view, my sense of saying, well it's the tool that drives the industry, not the language. I think that would eventually show itself in that. The Verilog was more appealing to the tool set, than being the best of the languages, so I think that that is what probably allowed it to win out.

But then of course, the language was, it was clearly showing its age. When I looked at it, I think, wow, you know, this is really old-fashioned. I remember the thoughts you had of FORTRAN when you got into the C programming thing. Wow, so people really programmed in that? It was that same flavor. Verilog is really showing its age. So we—something should—more effort, I think, should have happened in it earlier, but it didn't from the point of view there was no money. You know, the managers in the various, especially the big EDA companies, never put any value, ever, on the language. It was like well, other people do that, or the academics do that, or let somebody else do that. We don't want to pay anybody time and money to think about the language. Well, we'll just follow somebody else. So nothing ever could get started. I think the people that—what was being done on the committees, you know, what they went through, what were the versions they—but 2001 was the big version.

Golson: Yes, the biggest one.

Moorby: They did a great job. But it was, you know, half of that were fixing things that should have been fixed in the first place, and you know, the other half, you know, they pushed some of the constructs up a little, you know, and tidied up a bit. But in terms of what one would consider a new language, that's—2001 was certainly not the flavor of a new language. But then you can say, well, if you have a new language, what do you mean, you start all over again? And that would never happen. So SUPERLOG, you know, it's largely to the wisdom of Peter Flake, and Simon Davidmann was there driving largely the business side of it. And I, until I joined, I did a tiny little bit of consulting work with them, but not very much. And when I joined them, they had largely worked out the new constructs for the language, but it was all on the design side. So what became SystemVerilog was much, much more, but SUPERLOG was just focused on the design extension. So I'll let you answer the next question with that.

Golson: So, SUPERLOG, you got successful enough to attract yet another big company, in this case, Synopsys.

Moorby: Intel.

Golson: Intel? So okay, tell me about Intel.

Moorby: The simulator was very capable, but chasing Chronologic, as I explained, was, you know, proving to be a bit of a problem. But on the language side—the company was making some good sales, there were enough people in the world that really wanted to move on, and get themselves a better language, and I think the simulator proved itself to be good enough so that, you know, oh yeah, this is much better, we'd rather have some of the constructs that were at SUPERLOG, and I wasn't too much involved how Intel started to get hold of it, but they—I think it was a combination where there was a certain set of constructs in the language, they said, okay, we really want those. Intel was then going through a transition phase of how to—politically, that, before then, the company was all about their internal tools, a very, very capable set of tools. But high level management, I mean, I can't remember. It's somebody obviously, you know, word of mouth came down, that the high level management in Intel had made this decision that they're going to go through so many years of a transition, to having all of their tools being from the public domain. In other words, from, supported from the EDA industry, and they can't afford to support this any longer. So I think that they have sort of equivalent EDA people in the company, obviously didn't necessarily like that, but I think they had to deal with it. So they wanted to basically come out and I think they had the objective of taking over total influence of the EDA market, because they, you know, the EDA market had better listen to them and do what they want.

So their first stop, I think largely, was probably Synopsys, and said, okay, we're going to get Synopsys to do what we want. And at the same time, they were looking at SUPERLOG. Said, okay, this is getting closer to their—because they had an internal simulator and language, very, very capable. I said, okay, it's got features that, are similar to our own, and we want to influence where this is all going. And they could see us starting to influence the whole standardization effort of the next, you know, where the committee wanted to go, because we were doing a very good job of getting them on our side and seeing our way of doing things. So I think Intel started to tell Synopsys, they said, well, I think you need to acquire this. You can't let these guys, if they get into big-time buying Synopsys tools, they don't want somebody else to go off and freewheel in developing a new language, because they've got to have control over it all. I think that is the essence, I believe, of the driving force of Synopsys actually acquiring Co-Design.

Golson: How did that affect you as an engineer, after you got acquired, what happened with your products that you were working on, and your work on the language.

Moorby: Well, the ideas in SUPERLOG was very quickly merged with all the ideas within Synopsys, where they wanted to push the assertion language. And mainly the assertion language and the Vera, the verification language. Sort of merge it all into one big umbrella, push it into the committee, the standardization committees, and essentially dominate that—where all the thinking in that language—all that language effort went, and make sure that their toolset was the best. And they did a—I mean, if you look at each of those areas, I mean, they had very good tools.

Golson: Synopsys did.

Moorby: Synopsys did. I think the assertion, what they had done with assertions, and a lot of that work was now being in cooperation with Intel, that they had an assertion, a simulation of the assertions, that was bolted into the simulator, and that was quite capable. And there was—you could see a lot of good work where that was going, in terms of assertion checking and on into more of the formal side of things. The Vera system was very capable, it had very good traction, customers wanted that to go, so they wanted Vera to be standardized, or something else happen to it. They didn't want it to go away, obviously, so I think Synopsys made a—probably a good decision to use that change and opportunity with SystemVerilog to actually bring that under that umbrella and still leverage the Vera customer base that they would be transitioned into whatever would be done, developed. And so the customers would eventually be happy, because it would all be standardized. So all of these pretty substantial tools were being put under the SystemVerilog umbrella and put into the standardization committees, and I saw, I don't remember how many committees were formed. There were lots of them. There were all sorts of committees popping up.

Golson: And how were you—were you involved at all in that, or did you just—

Moorby: I largely focused—joining Synopsys, I started to focus on the inner time wheel issues, I'm trying to help define or standardize, and bring out more of the technical details of how the event loop worked, because in order for other simulators to know—well, what had already gone into the standard was the normal events, were the NBA events, that there was a time wheel that defined when the one set of events occurred, and then followed by the NBA, followed by—I think it was called, at the end of the time wheel, where you do things like display and send values out to a display system, or capture system.

So that had already been defined in the committee work, but when you throw in all the extra constructs like the assertions and the—especially the program, the time of which the test program executed, all needed to be worked out in a way that could be standardized, and of course we worked with some of the people in Mentor and Cadence, too—because they were all up in arms. So you have to define this stuff, you can't just leave it very arbitrary, because we're all going to be different. So I was working on that mainly. I did some debug work on assertions. I hadn't actually worked on assertions, up until that time. So that was some very interesting—it's a different way of looking at logic, so I found that interesting, and found the ability to debug complex assertions, to be abysmal. So okay, I'll see what I can do, and came up with a debug mechanism for them, and that was—I was fairly happy with that. But they kept on extending the language, and especially in the assertion area and I—hard to keep up. I mean, it was all great stuff and it was all going more and more into feeding to the formal side of the analysis, but it was hard to—I mean, each one of these areas, you know, you take assertions, what they were doing there, in the verification language, what they were doing there. You could only keep up—each one was a full-time job, just to keep up and to keep understanding and do any work.

So to—I mean this is where I formed the opinion that no one person can understand all of the SystemVerilog. We're talking about normal hardware engineers trying to work with it, or verification engineers working with it, you know, they'll only know one piece of it. And there's no—there's just not enough time to be able to be an expert in any of the other areas. So I formed the opinion that—that's where I wrote a paper to say that the thing wasn't a single language. No one person can think in this language—to me, the definition of a language is that the normal set of people, or let's say at least a majority of people, working with the language, can actually think in the language, and understand most of it.

Golson: You're fluent. You're fluent in the language.

Moorby: And fluent and you can think in it, you know, like when you learn a foreign language, a point may come when you actually dream in it, or when you're always thinking it. That, I call a language, a coherent language that you can actually express yourself with. And SystemVerilog is not that. But it's a very capable set of individual tools, very powerful tools, put under this umbrella kind of language, and that was useful, but I mean, but it's very easy to criticize the interplay between the parts, when it doesn't always work out how you think it should.

Golson: Right. Well just looking at the hardware description aspect of it, I mean, it does have this backward compatibility, at least that it strives for, so I'm imagining some 1986 code.

Moorby: You're talking about the synthesizable parts?

Golson: Yes, yes.

Moorby: There were some useful extensions that went into it. Largely, I think the useful ones had been worked out with SUPERLOG, I don't think there were many constructs that went into—well, there were not many new constructs that made its way into Design Compiler. And I really can't tell you today, how many new constructs are in Design Compiler, from a language point of view, compared to how the language was defined, let's say in the—let's say, well when was the first version?

Golson: The first version of SystemVerilog, or—

Moorby: What was the first version of Verilog that was standardized?

Golson: Oh, 1995, wasn't it?

Moorby: Okay, the '95 standard. So if you look at the synthesizable constructs that was in Design Compiler then, versus what they put in with the 2001 standardization effort. And then the SystemVerilog standardization—I don't believe there are that many extra constructs that went in. I mean, there were some obvious good ones. I know, and I worked on, for the longest time, the interface construct that went in, they tried to put it in, and then I started to hear rumors that they tried and they pulled it, and—to tell you, I don't know how it ended up. Putting these sorts of constructs in is a real—it's very difficult, and to—I mean, there are some of the minor extensions of various data types and things that obviously are very useful. But some of the larger constructs are really hard to get in there, make them stable, make them useful, to a point where hardware designers can't do without them. So I think—I mean, all in all, I think SystemVerilog was a very useful, very good thing to happen. I just don't have the view that it's one language. It's a group of languages.

Golson: How did you decide to leave Synopsys? I mean, yet again, you're acquired by a big company, and you stayed for a while, and then you're moving on.

Moorby: Yeah, I enjoyed myself, I stayed there for six years, and no, it was good work, working on parallel programming, various projects, but the last—main one was parallelizing the simulator.

Golson: At this point, that simulator is VCS?

Moorby: VCS, yes. I learned a lot of things to do and don't do with trying to parallelize code. <*pauses*> You're going to ask me eventually, why leave?

Golson: Yes, yes—where did you go afterwards?

Moorby: Yes, to Sigmatix. Coming largely, I'd say, clearly defined as being outside the EDA industry. Had some links, meaning that it's in the next generation of wireless implementations. But how to develop high-speed software for the new wireless networks, another very big problem area to work on. Yes.

Golson: In 2005, EDAC honored you with the Kaufman Award. What was that experience like? Was that a big surprise to you, or had—

Moorby: That was pretty surreal.

Golson: Was it?

Moorby: Very surreal. I—for a number of years, actually, probably going on three, five years, you know, sometimes at the DAC conference, I bump into somebody like Wally Rhines and they’d say, “You've got to have the Kaufman Award,” and said, “You'll get it one year, but not just yet.” <*laughs*> So from that, I always wondered, well, will it ever happen? But yes, eventually it did. Aart [Aart de Geus] called me up one day, and—

Golson: Is that right? Yes. And you were gone from Synopsys at that point.

Moorby: Well—

Golson: Were you still—you were still there.

Moorby: No, I was in Synopsys, yes, and Aart was the chairman of the committee to do the award. And he took great pleasure in being able to call me up and tell me. That was pretty surreal. It was like, wow, okay, that's good.

Golson: That was obviously a fun experience, getting the award and having all these people show up, to sing your graces.

Moorby: Yes. Not used to the exposure, that's for sure, but it was—yes, I think I've got myself turned around to actually—I just have to enjoy this and not be too worried about the exposure and—that's not quite like me to be exposed to that sort of thing.

Golson: A common refrain that I heard from people is that you're a very quiet, behind-the-scenes person, but I think it was Richard Newton who said, "Still waters run deep," and that there's a lot going on beneath the surface with you.

Just a few questions wrapping up here. Can you pick out what accomplishments are you most proud of? I mean, you can just say, well, Verilog, but you might have something much more specific, that—

Moorby: No, not really. Just that, taking on really big challenges, and just staying with them. I don't know whether the right words to say is that you're proud of staying with them, or what, but I think that's given me the most enjoyment in a career, that is to pick a really, really hard problem, whether it turns into a successful startup or not, <inaudible> you hope it does, but it hasn't always. But that you actually get the enjoyment of solving those problems, learning a great deal from it, and actually being able to not be in the same space, or anywhere near the same problem, but being able to use your experience to get better at solving problems.

Golson: I have a quote from you, "For everything you add, try to take two away. Keep it simpler."

Moorby: That's Occam's Razor.

Golson: Yes, yes.

Moorby: Yes, it's—especially in a startup that's probably one of the best words of wisdom anybody could give a startup, to keep it simple. Most startups fail because they try to do too much. But it's certainly harder these days to do something—keep things simple and still make money. High tech has become more and more complicated, and you have to do more and more to get anywhere to make money. So it's more of a challenge these days. But the enjoyment has never gone away, of taking on really tough problems and just staying with it for as long as you have to.

Golson: So this is a tough question. If you could change or redo an event or a decision in your career, what would you go back and change?

Moorby: There's two answers to that. One is “Nothing at all”.

Golson: All right, that's a good answer.

Moorby: The other is, “Go back and redo everything”. <*laughter*> Yes, you know, because you've got to do better.

Golson: That's right, that's right.

Moorby: But would it be so exciting and interesting? Probably not.

Golson: Probably not. All right. Do you have advice for someone starting out in the industry today, a young engineer, or—specifically in EDA, perhaps?

Moorby: No real good words of wisdom—I don't—how do you give that to young engineers? They don't listen to you anyway. <*smiles*> My only word of advice is, they've got to get to know themselves, to get to know what kind of problems they like to solve, in the sense that—I can deal with long-term, what I think of as long-term problem solving, and the important thing is, the positive feedback you get from your work, and some people need that positive feedback, that stimulus, coming back for—whether it's other people, or solving a problem, or writing a code, a piece of code that works. Some people need that positive feedback on a daily basis, perhaps, or maybe even an hourly basis. I prefer the bigger problem, where you may not get a positive feedback for a much longer period of time than that. You might be working on some code or whatever, for months before you get that—oh, okay, I've got that, it's working, and somebody can see the effect, and you get that positive feedback. I think, for anybody starting out in engineering, you need to know yourself enough to know what you need. If you need that short-term positive feedback, then maybe you need to move to marketing and sales. <*laughs*> But if you like the big problems, it can be rewarding, but that time scale of that positive feedback can be pretty frustrating at times, because if it doesn't come after months of working away, and there's no guarantee it ever comes.

Golson: Yes, yes.

Moorby: Yes, you have to know what you want.

Golson: Did you ever imagine Verilog would have this long a life?

Moorby: Never thought about it. Never really—never thought in terms of how long it would last, looking at the future, that's the part I never looked at. I mean, we're always looking back, I've thought of that many times, but from now on, I was thinking, well how long—I guess it's all now a question for SystemVerilog, how long will that last, but I guess what they've done is never going to go away. We're stuck with it forever.

Golson: So my last question is, how do you see the future of design, hardware design, and perhaps that's part of your answer, is "Well, it's SystemVerilog," but do you have a vision you see?

Moorby: Isn't that what makes it all exciting? I mean we don't know, do we? I mean, I think there's a strong tendency to believe that the industry certainly gets more and more saturated, more and more, you know, getting to the asymptote of the limit of where it could go, and I'm talking about hardware, okay? So the work I've done at Sigmatix and so on, is that of, well, why—is the boundary between hardware and software, what is this? So if I put software embedded within a chip, well is that still hardware or not? If that's a very capable processor that's inside that chip, and I've got all the software inside it, you tell me, is that hardware design, or system design or what?

But I think, like at Sigmatix, that implementing the 4G wireless standards in software, there's this long term question that you always answer, and of course it's always been there, but on different scales. You can talk about a flip-flop, it's got software in it, it's got a zero or a one in it. So a zero and a one is software.

Well you can go on to higher levels than that, and chips can, if it's got software in it, is that still, you know, what is that? I guess you'd call it a system on a chip, now. But is that where it's all going, and will hardware itself, what we traditionally know as hardware, will we be designing a billion gate designs at some point? Are we already? I'm not sure, are we already? But how much of that is software, and… Working at Sigmatix, it's not considered to be part of EDA. So is EDA going to expand into these other areas or not? They haven't—I thought, like four or five years ago, that it would, but it doesn't seem like it is expanding out, it just seems to still be, doing better than it used to, but it's still largely, it's traditionally staying with the hardware. I guess hardware problems are still getting harder and harder to deal with, so there's always a problem to solve.

But these days, I'm more interested in the software influence over hardware, and especially where all the parallel programming is going. Parallel programming has got a long, long way to go.

Golson: Is there anything else you need to add?

Moorby: I don't think so.

Golson: All right, well thank you very much, Phil, it's been a pleasure, thank you.

Moorby: Thank you.

END OF PART TWO

END OF INTERVIEW