10/00

CARVER MEAD INTERVIEW PART TWO
Document 2 of 4 of Mead Interview
By: Gene Youngblood

CARVER: Full custom handcrafted chips typically take three years. What that means is that people hand design certain pieces and then they use some kind of computer aid to help them put the stuff together. Nobody sits down and draws the whole chip on one giant sheet of mylar any more like they used to. That's impossible. At least ten years ago people began using simple CAD systems to help place the pieces they'd drafted by hand. That's just plain common sense. Even the old kind of design aids helped. And there's been this continual evolution of things that made that easier to do. So any chip done in the last ten years, nobody sat down and drew the whole thing on one piece of paper. They drew pieces of it and then they plotted it and then they drew some more, and so on. You can get there that way but it's an enormous amount of effort. If you have a good methodology and good people that understand the design you can do some pretty ambitious chips using not very advanced tools, if you've thought the thing through.

GENE: By what factor does a silicon compiler reduce design time?

CARVER: Maybe 75% or an order of magnitude. From several weeks to several months. It depends on what you count; a lot of design goes into just conceptualizing what you need. One thing a silicon compiler allows you to do which we never could do before is exploratory architecture, where you can actually try a design and see how it comes out; if it doesn't work you try something else. We could never that before because the amount of energy to implement a design is so high that once you get on an approach you force it to finish, even if it's a horrible chip when you're done. And there's a lot of examples on the market. They got finished because economically it's not feasible to scrap them and start over.

GENE: Will design time ever be independent of chip complexity?

CARVER: It depends on what you mean by complexity. It's certainly not a direct function of the number of transistors; it's a function of how much the designer has to think about the design. That's quite a different matter. That has not been a dimension along which people have measured things very much. So design time will indeed become independent of raw transistor count, but people will figure out conceptually more complex things and then it'll take them longer to get those figured out. What a silicon compiler does is make the implementation trivial. It doesn't mean getting the idea is any easier.

GENE: Will something equivalent to supercomputer power be necessary for silicon compilation as we approach the multimillion-transistor level of complexity? Even now Cray is already promoting its machines for VLSI design.

CARVER: What people think of when they say supercomputer is going to change. I already showed you a chip that'll do about 600 VAXes worth of computation. The actual numbers of adds and multiplies are only maybe five or ten million per second; that sounds like maybe ten VAXes. But by the time you get done shuffling all the data around and getting everything in the right place and all that, it ends up being a few hundreds times real time instead of a few tens times real time. most of what goes on in a standard computer is just data shuffling. And that doesn't change with a supercomputer. It's the same problem. So people are going to reconceptualize what they think about supercomputers. It's going to be parallel architectures and a lot of special-purpose architecture. Sure, there'll be some standard “general-purpose” architectures but the big breakthroughs will be in special purpose architectures. The whole notion of supercomputing is going to change. The people in that business don't want it to change, because they're in that

business.

GENE: Well, let us say then, will power in the supercomputer range, no matter how it's achieved, be necessary for VLSI design?

CARVER: Sure. But what's really going to happen is that instead of doing a supercomputer people will build special little accelerators to do pieces of the problem. There'll be a little simulation engine that runs simulations like crazy. These various things that do the particular computations, and you'll plug those boards into your workstation.

GENE: Why is the silicon compiler die size larger than hand-crafted chips?

CARVER: There are a lot of approaches to silicon compilation today. The one I worked on, the chip size for a given function is much smaller than gate arrays or standard cells but still larger than hand design of course, because that's what hand design does -- if you see some space you pack some stuff into it. In a very funny way it does get to the point where that's compensated. What really happens is people redo the architecture to make the chip more efficient. So actually for a given function they can get things that are actually smaller than hand designs by exploring the architectural approaches. But if you took any one of those once you're finished and redesigned it by hand it would be smaller. But you can't afford to do that in VLSI. So the whole thing is really deceptive because what a silicon compiler allows you to do is experiment with the architecture until you find the most efficient architecture, and you can never do that in hand design. People will look at a given chip and say "Oh, I could make that smaller by hand," but they never would have gotten there by hand. So you're working a different end of the problem; you're letting people experiment on the architectural end. Then of course if they wanted to take that particular architecture and repack it by hand you can always get it physically smaller. It's always true that starting with something that exists you can always make it better by working on it. What the silicon compiler does is let you start from scratch and get to something that exists that's really quite efficient.

GENE: Is "full custom" synonymous with silicon algorithms?

CARVER: It means hand crafted. It's not synonymous with a dedicated architecture. Historically, "full custom" has meant that you give the thing to somebody in a little design shop and they do a hand-crafted design for that purpose.

GENE: In the literature they only say that silicon compilers are for full custom chips; they don't specifically talk about dedicated architectures. That's incredible, given the fact that everyone would acknowledge that dedicated architectures are inherently faster than general ones, and the only historical barrier to doing that has been the design time, and now you have a technology that can do it and they don't even remark that achievement.

CARVER: It's always true when you have a new technology that the only terms in which people can discuss it are the old terms. So it's really difficult to explain to people how things have to work when you have a new technology. The only thing people can do is compare with what they know. The only thing they know is that historically people have hand-crafted some designs.

GENE: There's the issue of who specifies the architecture, the designers or the compiler. Today we're not yet at the point where the computer simply doesn't have the intelligence to do that. So the designer specifies the architecture and then experiments with it through the silicon compiler. But apparently there are people pursuing the AI approach or expert systems approach where the compiler would have sufficient intelligence to actually make architectural decisions.

CARVER: Well the AI people haven't done anything yet so it's hard to make comments about it. That doesn't mean they won't. There's certainly room for heuristics in helping with the process, to

the extent that what we mean by AI is a heuristic approach to optimization -- of course there's room for that and everybody's going to be doing that. But there's another thing going on that you need to know about. What a lot of people mean by automatic architecture is fixed architecture. They have picked the architecture ahead of time and they compile a chip in that class. And all the early silicon compilers were like that. I wrote one myself in 1971 that was like that. it only did one little architecture. And it did all the silicon compiling things -- it took in the thing and generated the simulation from the same source that generated the artwork, but it only did this one little tiny architecture. And then when Dave Johannsen did his thing he did a wider class of architecture. And a lot of people now are doing an even wider class of architectures, but they're still within limits. There's nothing wrong with that if that architecture fits what you want to do, but it doesn't allow you to do architectural exploration, which is one of the dimensions of silicon compilation that's really highly important. The approach that the Silicon Compiler Inc. people tool -- which is Dave Johannsen and his people -- is to build a tool for people to do general architecture work. So they built a tool for architects; it does implementations of systems for people who want to do architecture -- to build algorithms in silicon, if you like. that isn't the only thing that needs to be done. A couple of companies back east have built silicon compilers for building microcode engines. That's going to have an impact on people who want to build microcode engines -- that's a data path with a set of finite-state machines that sequences it. Our own early silicon compilers were basically aimed at that thing too. since then they've become generalized. And of course once you have a general tool you can always put a layer on top of it that has a higher level representation and allows you then to generate the input for the general chip compiler. So the way I look at the actual evolution of the industry is there will be a few really general chip compilers, a bunch of front-ends for those that allow people to take a special thinking process or heuristic programs or whatever and generate input for that, then there will be some honest-to-god special-purpose silicon compilers.

GENE: Can we achieve VLSI density without sacrificing clock speed? For example, bipolar and CMOS are converging today and people say CMOS will outperform bipolar. So will be we able to carry a 100 MHz clock into VLSI?

CARVER: Oh sure. The problem with that is not that you can't make dense things that run at high clock rates, it's that you're not going to have the entire chip completely lockstep at those clock rates. Because just distributing a clock at those frequencies means that you're going to lose the synchrony over the chip. You should ask Chuck Seitz about the fact that different pieces of the chip are going to have to be able to operate autonomously. And you couple them in such a way that their clocks don't have to be absolutely in phase. You run pieces of the chip on very fast clocks and you communicate between them with some other frequency.

GENE: If you think of any machine as being a clock, in the normal human-scaled world there seems to be a constant rule which says the fastest clock has to use the most energy, and the one uses the most energy usually has some size constraints on it in order to dissipate that heat. Therefore, in the normal physical human world the fastest clock that would also use the least amount of energy is an impossible clock. But when you get down to the submicron domain you have an impossible clock, because in order to achieve both high speed and high density compaction, it has to use the least amount of energy comparatively speaking.

CARVER: Well, remember that what you said in the beginning is really true: if you try to take a whole VLSI chip and clock it at 100 MHz it will indeed dissipate an impossible amount of energy. But you don't have to run the whole chip at the same clock rate and you don't have to clock every element every time -- there are a lot of ways you can get the effect of that speed without having this global thing that's just pumping all that charge all the time. That's just physics. If you have every element in there switching every time, then there's just the amount of stored energy times the frequency, and that doesn't change depending upon how many elements. You make the elements smaller, there are more of them, so in that sense you can run more things faster with the same power; but the basic physics doesn't change. Pumping things up and down at a fast rate takes a huge amount of energy and the faster the rate and the more things, the more energy. That part

doesn't change. It's really the area: how much area do you pump? Think of the chip as a big capacitor and you're just pumping charge in and out of it, and every time you pump it it's one-half CV squared that you lose in energy; so the power is one-half CV squared times the frequency. Period. Capacitance times V-squared hasn't been changing a lot with chip evolution; capacitance has been getting a little bigger, the voltage a little lower; it's getting a little better but basically that's one of those things that doesn't change much. What's really going on is that we can make individual pieces of the circuit extremely fast and then you do something to not have to run the whole circuit that fast. For example, communications controllers have to decode stuff coming in off of, say, a local-area network at 10 MHz; that means you've got to have some multiple of that in your resolution of things, so that's got to be fast on the front-end and for error-correcting, but then you get it in and you go into some parallel thing that can run a lot slower, so you do most of your processing at a lot slower rate, but in parallel. So your high clock rates are primarily for interfacing the chip with the outside world. If you're interfacing with an optical fiber you've got to be running in the gigahertz range. But somehow there's a way of not having to switch every element every time. (Chuck can tell you some nice things about the whole self-timing thing where you only do switching if you need it. Things don't switch unless they change).

GENE: Geoffrey Fox claims the speedup through concurrency is a linear function of the number of PE's, period.

CARVER: I have gone on record as saying that the special-purpose architectures are going to be the way to approach highly concurrent architectures. The reason for that isn't that isn't that -- yeah, for Geoffrey's particular problem, he can arrange things in such a way that you only have nearest neighbor communications and then you can get a linear increase in computation with the number of elements. And that'll work until he wants to do something a little more sophisticated. The computations they're doing isn't very different from the one you have to do in music, and they're getting about one VAX per circuit board worth of computation and we're getting about 600 VAXes per chip. That should calibrate you a little bit.

GENE: What's the problem with doing floating-point in silicon?

CARVER: We chose to do 64-bit fixed point instead of 16-bit floating point in our music chips because it's much cleaner and for a lot of applications if you go to a really long word you can use a fixed point number every bit as effectively as you can use a shorter floating point number, and the computation does get enormously simplified. I mean, doing floating point is a bitch. There's no question about that. Because you have all the interaction between the exponent and the mantissa's. In particular, adds are horrible.

GENE: There's a lot of floating point in graphics.

CARVER: Well nobody's ever thought through if you really need to do that or not. It's just that, they look at the dynamic range they have, which is huge, and they say gee we've got to do floating point. On the other hand they've only got a 1000 by 1000 matrix, so it's certainly a finite resolution of things. So I believe nobody's really thought through whether you can do that with a long-word fixed point arithmetic. Nobody has a long-word, fixed point engine, it's all 16-bit or 32-bit. Algorithms in silicon would facilitate that. But I'm really not sure whether it would be more effective to buy a bunch of floating point engines to do it or whether it would be more effective to just have some long-word fixed-point engines -- of which you could have a lot more on the same silicon. And then you ask yourself what the tradeoff is. And it's just an engineering tradeoff, it's not a religious issue.

GENE: Geoffrey Fox said that for his particular scientific problems, the larger the problem the less viable it is to build it into hardware. The more parts you have with greater degrees of freedom, the more general the problem and you need a general purpose computer.

CARVER: That's the old lore of the programmable machine people. They know how to program;

they don't know how to design chips. So they assume its easier to write a program. They can't conceive that with a chip compiler you could compile a special-purpose architecture for a problem easier than you can figure out how to use one of those stupid programmable highly concurrent machines. Right now if you look at the amount of programming time it takes use an array processor, you could easily have a special-purpose architecture up and running before you could have the same algorithms running on an array machine. I'm not saying that's always true; there is a lot of merit to programmable things you can change in some way; but it's a big mistake to think that sequential programming languages are going to work on concurrent machines for anything except the most simple problems. It's real easy when everything's doing the same thing, to write down what it does and you have a bunch of Von Neumann machines working on it and its straightforward. But when it's more complicated than that it's not so trivial any more. I'll once again refer to the music problem, where the instruments are separable in the sense that a voice is separate from other voices, but then they have to get choreograph- ed together with some kind of conductor kind of thing, and each one's a little different, and now it isn't quite so trivial anymore.

GENE: It's probably the same thing in graphics if you want to do photoreal simulations of natural phenomena. That's very interesting. The physicists will say they're doing the real serious work, modeling the universe, and if you want to play with the big boys and model the universe, then you need a general purpose supercomputer. But it seems to me that modeling natural phenomena for visual simulation is at least as complex if not more so in terms of the dynamic complexity of the problem.

CARVER: It has one other advantage: you can tell if you've done it. A whole lot of physics these days has the problem of how many angels are dancing on the head of a pin.

GENE: So could it be that modeling the universe is a simpler problem than modeling an orchestra or a small ensemble?

CARVER: Yes, it's more real. Physics has gotten itself off in left field. I don't want to get my physics friends mad at me. I do a lot of physics myself. But the physics community is in a bit of trouble nowadays and the reason is it has lost a lot of contact with reality. I mean the kinds of experiments where they look for these particles at some enormous energies and they have some symmetry groups which they think explain the particles, which are really just a way of cataloging. A symmetry group doesn't explain anything. It's like Mendelev before we understood what made the periodic table; you could see there was periodicity in it and you could make some predictions from that. Well that's nice but it doesn't say why it's there. My perception is that they've got a very peculiar view going in physics and they cover that up with a lot of ego. It's very, very far from anything direct and experiencable by real human beings or even measurable by real human beings. If you look at the experiments that are done by casts of thousands on billion-dollar facilities, the whole thing has taken on an air of unreality that's just monumental.

GENE: If the 286 isn't significantly different from the 8008 in the sense that it's a microprocessor, then at what level of thinking does VLSI begin for you?

CARVER: It's certainly true that the microprocessor started an era where computation and silicon weren't separate any more. In other words the real significance of the micro- processor was that people could no longer think of computation as going separate from its technology base. In fact it never had been separate. It's just that people thought of it that way. There was the computer industry and there was some other place where they bought their parts, and those were just parts and they didn't really matter. Well the complexity of things continued to grow because the basic fab process allowed you to put more and more transistors down; but building a big memory is no different than building a little memory. So in terms of the way the fab guys have to think, it's VLSI all right because they've got to make the thing yield. So from their point of view it's VLSI the moment it gets to 100,000 transistors. But from the point of view of the designer those memories are no different than they ever were. They're just little wizened circuits. And the memory designer's job is no harder, involves no more intellectual content, than it did before VLSI. Have

you heard the term "algorithmic complexity?" The idea is, how would you characterize the complexity of a machine? Well, if you look at a crystal, for example, it can have ten to the twenty-fourth atoms in it. But you can specify it by the shape of a unit cell and the way the unit cells are stacked together. So there's really only two pieces of information. So in a few tens of bits you can specify a crystal. So it's not anywhere the ten to the twenty-fourth kind of complexity because of the very regular nature of it. Programs are the same way. You write a loop and the loop can go and create a god-awful amount of stuff, but if it's one simple little thing, no matter how much it's repeated it's not complex somehow. So if you look at it from that point of view and say well how can we talk about the complexity of silicon algorithms, it's really the sort of unique function and the amount of unique interconnection that have to be specified, that aren't just repeated regularly, that gives you the nature of the complexity of the thing. So let's look at Geoffrey Fox's machine. It's a crystal. He's already told you that. No matter how big it is it's a crystal. It has one kind of unit cell that's hooked up in some way to its neighbors. so you have to specify exactly what you have to specify in a crystal: it's no more complex than the description of its unit cell and of its connections. And the fact that it might have a longer-word ALU doesn't make it more complicated. But if it has a more complicated ALU -- if it does floating-point, that's a lot more complicated than a fixed-point ALU. But if you go from a 16-bit ALU to a 64-bit ALU that doesn't make it more complicated, algorithmically. It's harder for the fab guy to get yield out of it, so from his point of view it may be qualitatively different. So the way I look at it is that the fab guys have created a technology that's independent of what people use it for: a systems designer can sue it for either a sophisticated purpose or a dead simple purpose. If people decide to just make crystals out of it -- make a memory -- which doesn't have any more conceptual content than memories ever did, then it's just a cost-reduction mechanism. That's all it is. It hasn't added any real value except that you get more memory in a box of the same size for the same price. And that's nice. But the real value of it added by what you put into the memory. And that's why memories are cheap: there's no conceptual content in them. But now you take something like a 32-bit microprocessor -- and I was a little facetious: the 286 is really quite a bit more sophisticated than the 8080 -- so in fact they've added more value there than you would if it had been a memory, because it does a lot of things better; on the other hand, if you take that another level and ask if there's any new conceptual content in it, well no, it's doing what minicomputers always did. So it's new to silicon in some way maybe but it adds nothing new to the theory of computing. Incidentally, this is why there's always what people call the hardware/software tradeoff. It means where is the value? Is the value in the patterns on the silicon or is the value in the little bit-patterns in the memory after you've made the silicon? And you can always trade that off. The information's always going to be on the silicon, but it may be in little electrical charges on the memory locations; in which case it was the value added by the guy that wrote the software. Or it may be that someone actually created a structure that embodied that algorithm, in which case he put it directly on the silicon -- in the wiring, in the content of those pieces of silicon. So for me, VLSI begins when you can start thinking about algorithmic complexity. That's when it gets interesting. And I've always thought of it that way, ever since it was just LSI. Because the potential was there to build really complex things.

GENE: The silicon algorithm is a photograph that does what it represents.

CARVER: Furthermore they're beautiful. As forms, they're beautiful. If you've lived with them for awhile you can appreciate them as an art form. You can tell a beautiful design from an ugly design that does the same thing very easily.

CARVER: There are two transistors per memory location in a DRAM and six in a SRAM. And three or four transistors per gate, if you're building gates. But the part they don't tell you -- these people that like to count gates -- is that if you take a good well-designed silicon chip that's done by experts and you make a gate diagram for what it does -- see, the thing itself can't be represented by gates, because they don't represent a lot of the functionality you can get with transistors. Bit if you make a gate diagram of the thing it turns out that we get about one equivalent gate for every one and a half transistors. Because you use transistors not for making gates but for making things much more clever than gates. Like you use a single-pass transistor as a memory location. If you

had to build that with gates you'd have to make two cross- coupled AND gates and some other stuff and it'd take at least four gates to make what one single-pass transistor does. So gates aren't an appropriate description for what happens in VLSI. and there's this enormous argument about well, that's 30,000 transistors that's only 8000 gates. No 30,000 transistors is probably like 20,000 gates -- if you did it with gates, like in a gate-array. That part doesn't get told. But it means you're using the silicon in a much more effective way than making gates out of it. So to summarize, if it's done well the number of transistors per gate-equivalent is between one and two. That's because you don't use them to make gates; if you used them to make gates it'd be three or four. So you're already using them twice as effectively as you would if you made gates, just by being the slightest bit intelligent.

GENE: What is meant by "devices per chip" and how much of the chip area is devoted to "gates" and how much to wires?

CARVER: Devices usually refers to transistors, and that's misleading because wires take up about 95 percent of the chip area. Let me say it another way: if you put the wires down that have to be there you can usually slip the transistors underneath and not notice. It's down on the bottom levels and the wires go over the top, and usually the wires you can work in with pieces of the transistor and it turns out that if they were shrunk to zero area for the actual transistors themselves you wouldn't save more than five or ten percent of chip area.

GENE: So when you say 100 million transistors per chip you're only talking about ten percent of the chip area?

CARVER: Yeah, but you see, if it were just the transistors there they wouldn't do anything. What they do is determined by how they're interconnected. So actually it's not a bad measure, because when people have a working chip they tell you how many transistors they were able to interconnect constructively. And that's an important number. That number is one dimension of this complexity issue: it doesn't tell you if they're extremely regular or if they have more information than that in them. And just because a thing appears regular doesn't mean there isn't information in it that makes things not equivalent to other things. Like a ROM looks extremely regular but it may have a lot of information in it because of the patterns in ROMs. Like Geoffrey Fox's computer: if you were to put a different program in every machine, it would be a much more complex thing than if you have one program in all the machines. That's what I meant by algorithmic complexity. Of course, figuring out how to use it would also be that much more complex.

GENE: Is there a rule of thumb by which you can relate the minimum feature size or design rule to the total area of a transistor in terms of square microns?

CARVER: A typical transistor in today's world is of the order of one design rule by one design rule. Three by three is the most common transistor.

GENE: I read something that said 480 square microns.

CARVER: That's the amount of chip per device. If you take the total number of devices and divide by chip area that's what you get. So then immediately you can take those two numbers and figure out what the fraction of utilization is.

GENE: What is the computer revolution?

CARVER: There are a number of levels. The most obvious level is that since anybody can own an ordinary Von Neumann-style computer, people are now discovering that they can do a lot of things, so everybody's got a PC at home. They write programs to do all kinds of things. That's become a thing you can talk about which you could never talk about before. It's like the car. Enough people own cars that you can talk about them as a metaphor for explaining other things. To me that's been the value of the microcomputer phenomenon: not so much what they do but the

fact that they're obvious enough and useful enough and ubiquitous enough that everybody knows at least something about them. They're not great big things that gobble you up anymore. So that's been a big help. But much more important than that in terms of microcomputers are the microcomputers that get built into things -- telephones and typewriters and VCRs and all the ordinary everyday life appliances and tools. Electric drills. A lot of people don't understand that the reason electric drills don't get stuck as much as they used to is that they have a microprocessor in them that looks at how fast the chuck's going around and where your finger is on the trigger and figures out what they ought to do. Those are really much more powerful in the revolution they create because they solve real problems instead of being something that you have to program to solve a real problem. They actually just do it. Somebody worked the whole thing out instead of just saying well here, now it's your turn. All of this is in the context of what people think of as computers. The fact that there has occurred a viable third-party software industry where it's now considered OK to write software, you can actually make money doing that. That's not something that was perceived a few years ago. That's where most of the actual value is added in terms of algorithmic complexity. The value is added in the programs, not the hardware itself. The hardware is trivial. Most computers don't have much in them. They're really just a receptacle for software. So of course the really important thing that's coming up -- and we haven't really begun to see it -- is the revolution of algorithms in silicon, where all the things you could never do with computers -- which were the things you really wanted to do -- like make pictures and music and all of the things that relate to people and to what people really want to do -- ordinary computers can't touch that. They're off by at least five orders of magnitude in computational power. That to me is the really exciting thing and it's the thing that hasn't been touched yet by what people are talking about as the computer revolution. But I think if you dig underneath all that, people's notion of value is changing. It used to be that people thought of material as value, so they'd think about steel or aluminum or gold or diamonds. It was things that were valuable. After a while it became clear that it was also energy that was valuable: energy allowed you to transform things into other things. Up until very recently that's been people's image of value: things held value. And the fact that it's actually information that's the thing that's valuable rather than the substance, is something that's a new idea. That's why nobody would give you a dime for software. Because what the hell it's just a tape you can copy so why is there any value in it. You could have said the same for motion pictures except that was in a different domain so it had different laws and it grew up understanding that the value wasn't in the celluloid. But everyplace else, especially in technology, people had this very weird view of where the value really was. To me the most fundamental revolution that's going on is people are now beginning to perceive that not just in entertainment is the information the important thing. And that's always been true in entertainment, in music and motion pictures and paintings. It wasn't those little tubes of paint and a piece of canvas, it was the information that was there. So it's always been true in the arts but it has taken a long time to dawn in other areas. And now it's happening in machines and businesses. People are beginning to see that the information is the valuable thing. It brings these closer to the fine arts and the human pursuits, not farther away. That's not generally perceived but it's very true.

GENE: Is it fair to say that highly concurrent silicon algorithms put many orders of magnitude more computing power in the hands of average people? Is that part of the revolution?

CARVER: Oh yeah. Things that the ordinary computer could never do. Vision. Speech understanding. Music. Visual simulation.

End of Carver