10/00

CARVER MEAD INTERVIEW
By: Gene Youngblood
(Document 1 of 4 of Mead Interview)

GENE: The progression toward a speed/power inversion has been happening continuously since 1959 when the integrated circuit was invented and yet today we still can't build a supercomputer that doesn't require acres of freon tubes for cooling.

CARVER: Oh yes you can. It's just they're not doing it. You see the problem has been twofold. One, as the technology developed the big computer people -- the Amdhahls and the Crays -- stayed with the older bipolar gate array technology. They haven't participated in the revolution. Now the semiconductor guys don't know much about computers so they've copied a bunch of ancient architectures. So what we have in the personal computers is creaky old architectures with this wonderful technology. So the people driving the technology don't know anything about systems and the people who have built traditional large computer systems haven't participated in the technology revolution. Supercomputers are an extremely inefficient use of power and space. They just brute-forced it. They said we're not going to use any cleverness at all, we're just going to pour the coal to it and see how fast we can make it run. So they just turn up the knob and go for it. And you end up with these giant steam engines that blow off all this heat. It doesn't make any sense at all from any point of view. But, you see, the situation is actually much better than that. And that's the part nobody counts. I mean if you take today's techno- logy and use it to do a really novel architecture you can get a factor of 10,000 right now. Today. But you don't do it by running the clock that fast. You do it through parallelism. There's a lot more to be gained through architecture than there is in clock speed. You can get 10 GHz out of a parallel array of 10 MHz clocks.

GENE: But aren't there serious problems with software for parallel architectures? There seems to be controversy whether schemes like dataflow and functional programming can actually overcome the communication problems.

CARVER: As long as you're talking software you're missing the point. Because you're thinking of something programmable. And those schemes will never work out in terms of enormous parallelism. That's why I think this super-computer thing will turn out to be a net loss to our country. Because they're still clinging to the belief that we're going to make programmable highly parallel machines. I worked on that for a long time and finally came to the conclusion that it just wasn't going to make it. You were going to get factors of ten or a hundred and that was the end of it. Well that's not enough. We need eight or nine orders of magnitude to do the kinds of things we want to do with computers today. There's factors of a million there if you do it right. But for that you can't separate the architecture from the algorithm. You have to build that algorithm in silicon, not program it somehow. I think they'd better be facing straight into the fact that there are dedicated architectures for these enormous tasks.

GENE: What do you call the kind of architecture you're talking about?

CARVER: I call them silicon algorithms. You just map the application right into silicon. And the technology for doing that we understand pretty well. I don't mean we've work- ed it all out. But we've developed the technology for taking high-level descriptions and putting them on silicon. It's called silicon compilation, which is the obvious next step. You need to compile things right down into the chip instead of compiling them into some code that you feed to this giant mass of chips that grinds away on it trying to interpret it.

GENE: So the silicon compiler solves not only the complexity problem but also the problem of massively parallel architectures by allowing us to experiment with the dedicated hardware that alone can realize the full potential of that architecture.

CARVER: You bet. Otherwise there's no hope. Because then you're stuck. If it takes five years to design a chip nobody's going to experiment with algorithms in silicon, right? Because it takes years to get anyplace. And that's in series with your learning curve. So you have to have a technology for experimenting with designs in silicon more rapidly than that.

GENE: Then microelectronics and computer science have become synonymous.

CARVER: They have to. You see, the thing that allowed people to separate the machinery of computing from the practice of computing -- hardware from software, if you like -- was the idea that what computers did was a sequential process. The old Turing or Von Neumann idea. Once you start to do things in a highly concurrent way you can't separate the algorithm from the architecture any more. You're confronted with the necessity of mapping classes of applications directly into silicon. There's no such thing any more as the general purpose programmable machine when it comes to really high bandwidth computing -- which is the only reason you need concurrent processing in the first place. So that doesn't mean Von Neumann machines will go away; they'll be used as general purpose controllers for these special purpose instruments. The user has to talk to something that's coherent, that has a language and an operating system and so forth. But instead of that poor little thing trying to execute all those instructions, you're going to have special-purpose engines it calls upon to do these enormous tasks. So you can think of the dedicated chips as extremely capable instructions that you call from your personal computer.

GENE: Would a personal computer based on one of the new 32-bit microprocessors be powerful enough to control a such a high speed engine?

CARVER: Oh sure. The very first one of those on the market is the Silicon Graphics Iris machine with the Geometry Engine in it that Jim Clark designed. It's got a 68000 that controls an enormous graphics pipeline with at least 100 times more computation capability, which nevertheless is a slave to the microprocessor. The reason, of course, is that one small instruction can cause an enormous amount of work to get done. So the microprocessor is used to update the picture.

GENE: Even with dedicated hardware there's a communication problem in any massively parallel architecture. So people start talking about wafer-scale integration. Is wafer scale technology necessary for highly concurrent machines?

CARVER: It's not essential but it would help a lot. There are many dimensions to the problem. But essentially you're right that every time you come off the chip and have to drive a big wire instead of the kind of wires you have on the chip, you pay for that in speed. So to the extent that you can put the wiring on a wafer instead of on a circuit board, the thing will run that much faster and be that much more cost effective. The problem is nobody has yet come up with a viable scheme for tolerating the faults that always occur. None of the redundancy schemes are viable yet in terms of efficient use of the technology. It has not attracted the magnitude of research effort that it's worth. I think it's a very exciting area. But as an alternative to wafer-scale integration you can arrange the chips as islands of computation where the links between them aren't too huge. It depends entirely on the nature of the algorithm. How tightly connected are the clumps of computation? You could, for example, arrange the islands to be synchronous and where they're not connected too tightly use an asynchronous protocol so you don't have to lock all the clocks together. You'd still have a fairly small box. So even if you have to go off chip you arrange it so everything that's on the chip is tightly connected. That's the whole game of making concurrent architectures whether you're on a wafer or not. It has to be based on locality. Otherwise the wiring mess just gets completely out of hand. Incidentally, the brain has the same problem. Your cortex is basically two-dimensional. It's only a millimeter thick. That's a lot thicker than a chip but it's by no means a fully connected three-dimensional volume. People think that because the brain is encased in this little round thing it's three-dimensional. It's not. It's two-dimensional, and it's wired in a two-dimensional way. In fact there are two layers. There's the gray matter on top of the cortex which is where the processing happens, and there's the white matter on the bottom half of the cortex which is where the wires are -- a solid mat of wires about half or one-third as thick as the cortex. That's exactly what we have in silicon except we don't have quite as many layers of thickness. But it doesn't qualitatively change the nature of the problem. And we're getting more layers faster than the brain is. So the whole idea of putting priority on locality is as true in the brain as in a chip.

GENE: Are silicon compilers based on artificial intelligence? Are they expert systems?

CARVER: You could think of them as expert systems, if you like, in the sense that they capture the expertise of the designers, but they don't do it by being artificial intelligensia; they do it because some really smart people worked the problem. Silicon compilers are based on ordinary, old-fashioned, good systems design. That's a whole different

game. In my opinion, artificial intelligence has done absolutely nothing that ever helped anyone to do a chip design. They may help in the future but so far they haven't.

GENE: Will the silicon compiler become the universal method of chip design?

CARVER: Yes, for highly concurrent architectures.

GENE: Regardless of the complexity of the chip?

CARVER: Well you simply can't implement most of these algorithms without the complexity. There has to be a certain scale of integration before it makes sense. But yes, you're going to need a silicon compiler to design even a Von Neumann-style chip at the complexity level of VLSI But once you have VLSI what's to stop you from doing something a whole lot more interesting than a Von Neumann computer? That's really the point. People talk about the complexity as though it were a problem. It's not; it's an opportunity. We have to think about it as an opportunity for new ideas, not as a problem that has to be overcome so we can make one more Von Neumann computer. That's a terrible waste of a very beautiful technology. And that's what we're seeing right now. The situation is analogous to what happened in nineteenth century England after Farraday demonstrated that you could make electric generators and motors. The factories of that day were long sheds with huge steam engines that drove a rotating shaft that ran along under the ridge pole of the roof with big pulleys on it and belts down to all the machines. If you wanted to stop a lathe you slid the belt from the drive pulley onto an idler. Well, when they built the first electric motors they built enormous motors and used them in place of the steam engine. You still had the shaft and pulleys and belts. They could not conceptualize that they really ought to have fractional horsepower motors distributed around -- what we now call the parallel processing approach. But the other thing was that of course they couldn't afford to change everything. We look at that today and say well that was really dumb. But if you were living back then you'd probably have a thousand arguments for why it was the only sensible thing to do. That's where we are today in computer science. We have this wonderful technology and we're building one monolithic computer. And if that turns out to be inefficient we'll make a big one on a chip. So they talk about micromainframes. It reminds me of a cartoon of the board of directors in Detroit and one of the directors is saying "Well, if we have to make a compact, we'll make the biggest compact in the industry!"

CARVER: The reason I got into the whole sordid affair fifteen years ago is that we as a culture were stalled. The really smart innovators, who are always the little guys, were unable to get at the technology. And that's criminal. And of course the big companies liked it that way. Because it's fat lazy comfortable, right? Which is why they don't like me. I'm rocking the boat. But the important thing that has happened is that the little guy like Jim Clark, one guy, can get at it and turn the whole market upside down. That Geometry Engine that Jim Clark built is a different architecture but the same idea that I proposed to Dave Evans [of Evans and Sutherland] in 1975 as precisely the right thing to do with the technology, namely to do the transformation and clipping stuff. Here was the company that was supposed to be leading the way in graphics unwilling to do anything about it. They could sell these megabuck boxes, why did they need a cheap one that did the same thing? Finally Jim Clark did it, six years later.

GENE: As early as 1972 you were saying we'd soon reach the limits of Von Neumann architecture. And yet even today several more orders of magnitude in performance can still be gotten out of that architecture through submicron scaling and increased compaction.

CARVER: But you see the Von Neumann machine doesn't take very good advantage of that. For one thing it treats all memory as if it were equidistant. It makes everything as hard to get to as everything else. They try to compensate with things like caches, where they put some memory closer to the processor, but that's after the fact. It's not built into the architecture. So as a result Von Neumann architecture actually works against increased compaction at the chip level. It doesn't scale well at all. It's not a way you'd ever design a computer if you had VLSI technology in mind. If you want to get the full benefit of this marvelous new technology -- this new medium, if you will -- you need a whole new way of thinking about computer architecture. Everybody viewed this technology as a cost reduction mechanism for traditional designs instead of as a new medium with which to realize new classes of architecture. I've spent the last fifteen years of my life trying to get people to think in a fresh new way. And they haven't been doing that. The solid fundamental work that's necessary even to understand what's sensible to do isn't being pursued in very many places. I mean the fundamental limitations work from an architectural and algorithmic standpoint, not from a technology point of view. We're in pretty good shape technologically. There's a lot of depth

there across the board. But it's not paralleled by similar depth in the leading-edge systems area. We don't have good people in every major department at every major university doing excellent work on the cutting edge of architecture as we do in device physics and materials research. Basically that's just starting to happen now. There have been some signs of life recently, but that's after fifteen years of beating on it.

GENE: You said in 1977 that there was an eight-order-of-magnitude possibility in compaction of integrated circuits -- potentially 100 million transistors on a chip. How do those early projections look to you today?

CARVER: The original analysis we did in 1972 of how small you could make devices -- one quarter of a micron -- is holding up remarkably well. The chip sizes have also been going up, although slowly. In that 1977 report we were thinking of a chip about a square centimeter, approximately what they are today. So if you scale the devices to a quarter-micron and enlarge the chip area a little more you can get a hundred-million transistors on a chip. That's with the technology that's known today. There's not a single step in there that hasn't been proven feasible. You'd use ion-beam dry etching and X-ray lithography. That will be done in the next decade. There should be people doing it now.

GENE: How much more computing power is available beyond today's technology just by scaling everything to the limits?

CARVER: It depends on what you count. You can count computation per unit power or unit area. I think the fairest thing is computation per dollar. What are you going to get for your money in terms of cycles per second of real computation? From now on that number is going to go roughly like the square of the scaling. Ideally, when you scale everything the speed/power product goes like the cube: switching speed increases linearly as you scale down and density goes up as the square. If it's two to one in dimension it's four to one in density -- if you reduce a transistor by a factor of two on each side, four little transistors now fit where the big one used to be. But in addition to that they're running twice as fast. So every time you go down a factor of two in size you get a factor of eight -- a factor of four in the number of gates and a factor of two in speed. That says gate cycles per unit area goes like the cube also. But as you approach physical limits you don't quite get that full cube law anymore, which is why I say that from now on it's going to go like the square. So if we scale from, say, 1.25 down to .25 micron, that's five to one, which is 125 times in gate cycles per unit area and also 125 times in how much computation you get per unit power; it's probably not quite that good due to approaching physical limits, but we'll certainly get a factor of 100. And remember we're not really at 1.25 micron yet. It's really a 1.5 process today, and that's only the leading edge; three microns is the standard production process in today's commercial world. So that means we're talking about chips a thousand times faster than the ones in today's personal computer once we reach all those limits. And that's not counting concurrent operation. A computer with a thousand of those chips running concurrently would be a million times faster than today's machine.

GENE: What about three-dimensional chips?

CARVER: These things are built up in layers and in principle you could keep going. Everybody knows how to do that. For example, SOI -- silicon on insulator -- is probably the most promising way to start stacking things. But there are other ways. The problem is that the yield -- the overall probability that the thing works -- goes down exponentially with the number of layers in the process, even if it's just an insulation layer or a layer of wiring. And exponentials are deadly things. After a while they really get you. And it's not just the yield problem. As you make things smaller they also become much more susceptible to things like cosmic rays and all kinds of other mechanisms of failure.

GENE: Compound materials like gallium arsenide are supposed to be less vulnerable to soft errors.

CARVER: Every technology has its own horrible problems and that includes gallium arsenide. There's no magic. The number of electrons created by ionizing particles isn't tremendously different between materials, not by huge orders of magnitude. It's really a size question, a straight-forward scaling issue -- how many electrons represent your signal versus how many electrons are created when you put an ionizing particle through the material? As you scale down there are fewer electrons per device, so it becomes easier for a random particle to switch the transistor for the same reason it's easier for you to switch it.

CARVER: I believe that the vision of the original founders of AI -- the Minskys and McCarthys -- was extremely correct. But when they got into it they discovered that what they were really headed for took eight or nine orders of magnitude more computation than you could get out of a regular computer. And so there came to be two groups: one went looking for that eight or nine orders of magnitude; the other just punted and faked it -- and that's the vast majority of the AI community today. I'm one of the people who's off trying to find that eight or nine orders of magnitude.

GENE: I've heard cellular automata used as a general term for massively parallel architectures.

CARVER: Cellular automata are an invention of Von Neumann, strangely enough. He described them back in the forties. It was the first ultra-concurrent architecture, a really good first step in what's now become a large class of things. For example, you can view Kung's systolic algorithms as an extension of cellular automata. And if you look at it that way, it's a very natural evolution. But in fact a cellular automaton is an extremely precise thing -- an interconnected set of finite automata. Traditional cellular automata have only been connected to neighbors, and as such they represent only very local computations. Any such local computation can be expressed as a cellular automaton but it's often not relevant to do so. Very few algorithms map well into that domain. So they've kind of boxed themselves out of a lot of the more interesting views of computation.

GENE: What about the demarcations between levels of integration, like SSI, MSI, and so forth?

CARVER: To me the differences aren't so much in transistor count as in the level of functionality, that is, how you have to think about it in terms of what it's doing. The first integrated circuit in 1959 had one gate or one flip-flop, so you could think of them as elementary logic elements. Medium-scale chips had things like counters and registers. Now people could start thinking at a higher level than just gates. That was in the middle sixties. The next real step happened when memories and microprocessors were integrated on the chip around 1971. That's when LSI begins for me, not because of the number of transistors but because now it was a complete system-level function. Those early micros only had about three thousand transistors but they were a processor nevertheless and they changed the way people thought. Now actually if you look at the Intel 286 today, it isn't really more capable -- or not very much -- than those early chips. It's got a lot more transistors but the architecture's pretty much the same; the instruction set's pretty much the same. So I don't put it in a different class from the 8008. Also, according to this view the gate-array business represents a throwback: gate arrays are the SSI level and standard cells are the MSI level.

GENE: If 32-bit microprocessors aren't qualitatively different from 8-bit versions, then when does VLSI begin?

CARVER: I'm looking forward to the machine that'll do ray-tracing in real time. It's not beyond our capability right now. Provided you compiled the algorithms into silicon in a massive array. I don't believe that polygonal representation for shapes is the right one. I think stuff like superquadrics is a whole lot more sensible -- maybe not exactly that representation, but something that has some nice mathematical properties when it comes to ray-tracing. So if you're going to use .CP10 a ray-tracing algorithm you'd better use something other than polygons. In fact, you have to invent the representation and the algorithm together. That's why nobody's going to do it unless they get their head around the whole problem.

CARVER: My own work is concerned with music. As you know, if you try to simulate even one musical instrument you're in for at least 10 MIPs worth of computation for a single voice -- one string on a guitar or whatever. In some cases you get a whole instrument like a flute, but if you try to do a piano you've got to have a huge number of processors. So you get into the gigaops very quickly to simulate any reasonable ensemble at all. A whole orchestra is a lot of gigaops. And it turns out you can do that. We're right now looking at about 10 MIPS per chip for doing music. A specific architecture for doing music, not good for anything else, and you can make some of the most beautiful voices you've ever heard in your life. We'll be able to simulate a very convincing orchestra. Now I'm not saying we're ever going to replace a real orchestra, but there's a set of things you can do if you have the ability to create your own orchestra. This is what the synthesizer guys ought to be doing and can't; and the reason they can't is that you need four or five more orders of computation than is available today. That's what prevents them from true orchestral simulation. A standard synthesizer sounds like hell and it doesn't have to, but you've got to find that four or four or five orders of magnitude -- in this case it isn't eight or nine, only four or five, but it's still a lot. And we can put it in a personal workstation for use by composers. I believe that's the way composition will be done. Composers

simply cannot afford to experiment. You can't afford to pay a hundred union musicians to fool around. So composers are really stuck, just be- cause they need that four or five orders of magnitude. We're still in the very early experimental stages but we can already mix amazing voices. Most people say it sounds like a real instrument, not like a computer. Except you can transform it and move it into different spaces. We can simulate a marimba bar that would have to be 27 feet long.

End of document 1

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

10/00

Carver Mead Interview Part III
(With Dick Lyon, Schlumberger AI researcher)
By: Gene Youngblood
Document 3 of 4 of Mead Interview

CARVER: When I say that one of our music chips is equal to 600 VAXes, well, "equal" is a funny word. What I mean is that if you do the same calculations on a VAX it takes about ten minutes to do one second of sound. So one of those chips does as much computation of that particular sort as 600 VAXes could do. Now VAXes are very poorly set up for doing computations of that sort, but so is every other general purpose computer. Now there are special-purpose engines that can do comparable things; so mine isn't the only way to do it.

GENE: What did you mean by "by the time you get done shuffling all the data around, you wind up with something that's a few hundred times real time instead of a few tens times real time?" What do you mean by real time?

CARVER: You ought to be able to generate the sound as fast as you need the samples. To compute the samples as fast as they're needed for you to hear the sound. You can't do that on a regular computer. If you look at how many multiplies and adds it takes per second to do these kinds of computations, it's only five or ten million per second. That sounds like five or ten times as many as a VAX can do. But it turns out that's not all the VAX is doing. It ends up shuffling all the data around to be in the right place -- you fetch it and you get it in the machine and you multiplying it and then you get it back out again. And all those how many multiplies you can do per second don't count if you're only getting the data to multiply, you see. We were very surprised. People had told me this for years, and I always thought well yeah, it's true for general programs, but I never thought that for a program like this it would be true that data shuffling took most of the time. But it turns out that there's enough pieces of data that you have to grind on and put into other places that that ends up taking more time than all the multiplies. Every time you want a piece of data you've got to compute it's address, then do a partial calculation, then figure out where to put the answer, and then put it back, and so on. All those things eat you alive. So only about one-tenth of the instructions actually end up multiplying anything. So you could do the actual multiplies in zero time and it doesn't speed things up very much. Whereas with special-purpose architecture you arrange it so the data automatically flows to the right place at the right time. In fact in our chip and others that was the key to the whole design -- to make sure that that's what happened.

GENE: Is there a rule of thumb for the amount of speedup you get from a dedicated architecture. All else being equal, if you're going to put the same algorithm in silicon as opposed to running it on a VAX. What kind of speedup in principle, just by doing that?

CARVER: For a lot of the applications we've looked at it's more than two orders of magnitude. One way to look at it is that you get one factor of ten from what was just being discussed -- from the fact that it's specialized -- and then as many more factors of ten as you like from the fact that you're putting more of the specialized hard- ware in that -- you get a factor of ten or 100 because you used 10 or 100 multipliers and another factor of ten because it's not general purpose. So that's a hundred or a thousand right there. So that's why we're at a factor of the order of 1000 -- because you've got things working in parallel and then you're not doing any data-shuffling. So that's another factor of ten.

DICK: My speech recognition machine's getting the same factor with respect to our general-purpose computer; I get somewhere between a factor of 1000 and 5000 speedup. I'm doing speech recognition by implementing a model of how the ear processes sounds. Sounds of all sorts including music.

CARVER: So my machine does this particular class of computations about 600 times faster than a VAX. But it won't run a PASCAL program and it doesn't run UNIX and it doesn't maintain a file system. The way I think of general purpose computers is they're the little environment that does all those stupid things -- hooks up to Ethernets, hooks up to telephone lines, hooks up to keyboards, hooks up to humans, compiles languages, accesses file systems -- does all those things AND issues commands to the special-purpose engines that do the really hard

grunt-work a thousand or more times faster than it could.

CARVER: Six hundred times a VAX would be about ten times as fast as a Cray. I haven't programmed this on a Cray so I wouldn't say that; in fact a Cray is set up to do this kind of calculation reasonably well also, so that for this particular task the Cray might be more than 60 times a VAX. I want to be very careful not to mislead people about this. One way of thinking about it is that you're not just taking the VAX and making a special purpose thing; you're taking the whole program that does a special thing on the VAX and you're saying that I'm going to take that very special application and instead of a general purpose programming language on a general purpose computer, I'm going to put the whole thing -- application and all -- right down into silicon.

GENE: Who invented the silicon compiler?

CARVER: It evolved gradually over a period of about eight years. But the first one that was recognizable as a compiler instead of something less general was done by Dave Johannsen in 1977. I did a very special purpose one in 1971 for finite-state machines, but it wasn't a silicon compiler in the sense that we talk about them today.

GENE: The appearance of silicon compilers is a significant historical event; does what makes them possible come out of your structured approach to VLSI design?

CARVER: It had been possible for a long time. It isn't a matter of making it possible; it's that people had been thinking of hand-crafting everything because in the early days you never had enough silicon to waste any at all. And any structured approach to design is in some sense slightly inefficient at the bottom level. You can always find little places that aren't all filled up with transistors. Exactly the same history happened in code. Back when machines only had a 4K memory, people wrote every machine instruction because you couldn't afford any waste at all. But when you have a megabyte of memory you can let a compiler allocate variables because if you waste ten percent it doesn't matter any more. The same thing is true of silicon compilation: people did not foresee the fact that we would have so much real estate available; that the real problem was the design cost, not the cost of the real estate. When I tried to tell people that in the early days they laughed at me. That was within five years of the time they were standing up and saying it's the most important problem for the semiconductor industry. They were laughing when you would say that the design cost was going to be larger than the production cost for complicated parts. Now that it's true it's them that invented it. But at the time I remember trying to get the Intel people interested -- this was in 1973 -- and Gordon Moore said it was an academic problem, and by the late 1970s he was publishing his curve of the design costs going up to the moon. The compiler viewpoint comes from saying we're going to have enough real estate and we're not going to have enough engineering man-hours, so what you ought to be working on is the design problem, not how dense is the silicon.

DICK: But it's still fair to say that this approach of doing things in a more structured way is what made silicon compilers possible, by setting up that way of thinking. To go from hand-crafting to structural building blocks sets the stage for the next thing, which is compilation. So it's fair to say that the Mead-Conway approach made silicon compilers possible.

CARVER: And Dave Johannsen was the one who first really saw how to pull the whole thing together in a much more general way than people had done before.

GENE: So it's a kind of synergy between VLSI, which makes possible a more simplified structured approach, plus that making possible the silicon compiler -- all of that together results in the ability to put algorithms in silicon with a fast turnaround time. You can't separate the compiler from the structured approach and you can't separate the structured approach from VLSI.

CARVER: There are a lot of things that can't be separated. Having people in business who are willing to fab chips that you design. In fact we had to start a whole new set of companies to do that because the traditional semiconductor people didn't want to do it. They still don't want to do it. They'll do it for their large customers who want to make a million parts. So this thing couldn't happen until we had access to fabrication. And there was a major amount of energy went into getting people to understand that there was a good business in that; and right now that's the only part of the semiconductor business that's making money. It's happening in a big way now. DARPA provides fab services at a very low cost to research projects in a few hundred universities. But there are

also a number of commercial places that will do that kind of thing now for low-volume engineering prototypes.

GENE: How do you talk about what is actually on a dedicated chip? What makes the adders and multipliers in your chip dedicated?

CARVER: Well, on one of our chips we're apt to have fifty multipliers all working at the same time; on a microprocessor you have to have one -- or zero, more likely, because they use adders instead and make up the multiply out of additions. Almost all microprocessors including the MC68000 have zero multipliers. Both in Dick's speech chip and the chip we use for music, the key is there's a way to set it up so that where the data comes from and where it goes to is something that can be set up and just run and not take a bunch of instructions to figure out everything.

GENE: Because the instructions are the chip?

CARVER: The chip has a specialized way of setting up where the data should come from and where it should go, so that the facilities for moving data efficiently from every place it might want to come from to every place it might want to go to are a big part of the chip. And it's not hardwired to a particular algorithm; it's got enough flexibility to be able to run a reasonable number of algorithms within a class. Like different music instrument models can be put on our chip. Different kinds of digital filtering structures can be put on Dick's speech chip.

GENE: So it's not accurate to say that this chip is a cello?

CARVER: It can be configured to be any kind of instrument, but it's dedicated in the sense that it performs that limited class of calculations which are relevant to those physics formulas. The big challenge in this specialized hardware game is to make a piece of hardware that's not so specialized that it's only good for one thing. You want to leave a certain amount of programmability of some sort. This results in a good chunk of your silicon area -- maybe as much as half -- being dedicated to the job of getting data from where it's coming from to where it's going. In a general purpose computer that takes ninety percent of your resources; in our chips it's probably less than 20 percent, in Dick's speech chip it's fifty percent. You get different amounts of generality for the different amounts of overhead.

GENE: How many transistors is on your chip?

CARVER: It has 64 multipliers, each giving a 32 by 32 multiply with a 64-bit product. There are 32 stages of the multiply. It has about 200,000 transistors. That's a lot less than the Hewlett-Packard chip that has 450,000. But for this particular application our chip is about 1000 times faster. The time-honored way of getting a high transistor count is to put a lot of memory on the chip. Our chip has a lot of memory which is mixed in with the processors, and the very thing that makes it efficient is that you mix the processing in with the memory so you don't have to move things from the memory to a processing element and back. You mix them together so processing is going on all the time. Let me say while we're at it that the 600 VAXes estimate is extremely conservative because the clock for our chip is slowed down so that it matches the sample rate which is the standard that all D to A converters use. There'd be nothing to prevent us from running the chip five times faster if you wanted to do graphics instead of sound. But in a sense that's not fair because we're computing these samples; and the fact that you could compute them faster than real time doesn't help. On the other hand, because a VAX computes them slower than real time we ought to be able to at least count that somehow. But essentially once you're at real time at a given sampling resolution you don't need to go any faster. Your ear couldn't hear it anyway. Now if we were working on bats instead of humans, well bats can hear at 200 KHz instead of 20 -- they're 10 times faster than people -- so then we'd have to run the clock 10 times as fast. We could do that, too, in which case it'd be the equivalent of 6000 VAXes. Now the other thing is that you may have the need to generate 10 musical instruments at once, and with our chip if you run it 10 times faster it still can't quite do that -- it can generate one very fast instrument but not ten at once. Dick's speech chip has an extra degree of flexibility that lets you be ten things at once instead of one thing very fast. In fact his chip is not very good at doing one thing very fast, so he couldn't make very good music for bats but he could maybe analyze sounds from several microphones at once. That costs him a lot; he has half his chip dedicated to getting data from where it's coming from to where it's going and by the way storing it temporarily in between. That's the extra cost of that extra generality. And that's the tradeoff you make depending on what you want to do. The discussion among people who do these specialized architectures is another level up from all this,

namely, now that you've got things that run 1000 times as fast as a general purpose computer would on the same problem, can you make it 10,000 times or 100,000 times faster -- can you get an even better match to the problem which uses the silicon in even a better way? It's just amazing when people start playing this game. It's not just more concurrency, it's algorithms that are more thought-through, tighter. In fact both Dick and I have traded off for generality the ultimate in performance. There would have been ways for each of us if we'd wanted to be really specialized to make the things another order of magnitude faster. We chose to have some generality instead. For example we both use a lot more bits of accuracy than are needed for most of the problems we're attacking, because occasionally you suddenly need more bits, so you spend the bits always and pay a big cost, but this means you don't often run into a problem where you don't have enough bits. That's one of the dimensions of variability, how many bits you choose to put in. We use 64-bit accumulators. Most people who do music synthesis would say that's outrageous, we're throwing away resources.

GENE: Today to get an updated version of a software package I buy a floppy disk. How would I update my computer if everything's in hardware? Buy a new board?

CARVER: There's a little company in the bay area called Silicon Solutions, Inc. You should talk to them because they make an algorithm in silicon. They founded this company on the premise that algorithms in silicon are going to be the future. Their first product checks design rules on integrated circuits. So it's an integrated circuit that checks for errors in the next integrated circuit. The next thing they're building is a chip simulator. What they did is make an add-on for a standard workstation. So you can plug the card into your workstation's multibus slot. And there's a little software package that goes with your standard UNIX operating system -- workstations are all UNIX based -- and now you can be up and running with their design-rule-checker in your workstation. So instead of a disk that you plug into your disk slot there's a board you plug into the board slot. There's a chip that implemented the Ethernet protocol. It got built on a little board that plugs into the PC. So that's an example of an already existing approach. When you buy software on a disk you pay five dollars for the floppy and three-hundred dollars for the ideas in it. A board costs about the same, maybe ten or twenty dollars for a cheap board instead of five dollars for a floppy disk. Or even if it was $100 for the board you're getting instead of something new but just as slow as ever, it does something new but 1000 times faster. So that factor of 1000 costs you a hundred bucks over what it would be if it was a software package. And you get a factor of 1000. That's a pretty good price. That's a factor of ten per dollar.

End of Carver 3

10/00

Carver Mead Interview Part IV
(With Dick Lyon, Schlumberger AI researcher)
By: Gene Youngblood
(Document 4 of 4 of Mead Interview)

DICK: You asked about what it's like to design algorithms when it's not just Pascal programs or something. It's a different game, but people who design algorithms always do it with some target computing technology in mind, whether it's a general purpose computer or full custom silicon or maybe you're designing the interconnect pattern for Carver's chip to make a new instrument model. Whatever your target technology is, that defines the game you're playing when you design algorithms. The neat thing about custom silicon is that you've got a lot of new kinds of games other than just trying new programs. So designing musical instruments is a new kind of game. You come up with algorithms that you could have written in a general purpose programming language but you probably wouldn't have, because if you had a general purpose computer to play with, you would have done something different -- more complicated, probably; more efficient maybe. If you're designing from bare silicon and you have to make a new algorithm, the game is wide open. All you know is you're making it out of submicroscopic transistors. If you're designing within the context of somebody's silicon compiler, you're a little more restricted than that -- you know you can make it out of predefined module types and register types and bus structures and so on. so that defines the rules of the game. A programming language puts a lot of preconceptions in your head about what you should do with a computer.

CARVER: In fact you do things very differently in different programming languages. Not because you couldn't do the same thing in two different languages but because the way you think in one language leads to a way of doing it which is natural in that language. Even though you could have thought in another way it's not as natural. So the language of silicon has certain things that are extremely natural to it. The idea of regularity matches very well the idea that communication is very expensive. If you think of things locally and have them communicate not too globally with too many other things, a): it helps you to think about it and make regular structures, and b): it also makes things that are very high performance. That was the part that the old silicon hackers missed. They didn't understand that a regular design can produce better performance than some irregular way. Because you force your thought processes to encompass the overall way that the thing works.

DICK: You still get questions from people who don't know enough about it who are steeped in the old lore but don't understand very much. They ask things like “If you design a chip in the Mead-Conway structured style, how much performance do you have to give up: 20 percent, 30 percent?” I say that's a silly question. The question is how much performance can you gain. They think of the structure as being an artificial limitation, which it is, but it allows you to do things you couldn't or wouldn't do otherwise. And you get to invent the structure, which means you make your problem easier not only at the bottom level of hacking transistors but at the next level up -- you can do creative design and algorithm planning at a higher level and make it easy to build the whole thing and make it efficient. Instead of doing it the old way where you draw many pages of schematics and then have somebody try to force it into the chip.

GENE: Why can't we get both VLSI and high speed at the same time?

CARVER: In 1956 when I was a student it was thought that if you make a bigger transistor it always got slower. That was the common experience of the day. So this one company said if I make a tiny transistor I can make it faster, and if I make a bunch of these little transistors and put them all in parallel, why isn't that faster? Well it is faster. Because all the parasitics that loaded down the big transistors aren't a property of the individual transistor; the real problem was that people were scaling things wrong to make the big transistors. They were thinking about it wrong. And these guys had figured it out. They asked me to prove scientifically that you don't have to get something that goes slower and takes more power if you make it bigger. So I analyzed what was actually going on there and that's actually what got me started thinking on this whole train of scale -- the very first consulting job I ever had. It turns out that it's the same kind of misconception when people assume something is constant that isn't when you scale.

GENE: If that's the case, then, can we put a number on the speed limit of the Von Neumann computer?

CARVER: We can do that very well. Let's do a one-chip machine. Things are scaling in such a way that the power/delay product is getting better by something between the square and the cube of the scaling factor. As you get close to the limits you don't get the whole cube law any more. So if you take the minimum-size transistors -- quarter-micron -- you have gate delay times in the 10 picosecond range. People have already made circuits with 30 picoseconds and the latest I heard was 18 picoseconds in submicron CMOS. And it's going to get better. So the speed limit of one isolated gate not driving anything will be in the 10 picosecond range; see, the problem with the Von Neumann architecture is they have these damn long wires you have to drive, which is inherent in the large RAM configuration -- so you need either long wires or many stages of delays. So you can say, well, what do I have to do to drive that thing? Well, the wires in terms of the technology underneath, are getting longer because you still want the wire to go clear across the chip because you're building a bigger machine; on the other hand the transistor itself is getting better able to drive, because the channel lengths are shorter and the submicron transistors are stronger for a given length. So when you add it all up, the actual speed gets better about linearly with the scaling. So from where we are now in device size -- say 1.25 micron -- that's a factor of five from the .25-micron technology, and it runs at about 20 MHz. So you can look forward -- if you're going to build a microcomputer that fills a chip at a quarter-micron and you have good processing and you've done things sensibly, you can look forward to in excess of 100 MHz clock rates for those things. That's already respectable by the standards of people like Cray. And you can look forward to the same kind of complexities you have on those kinds of machines. Right now you have complexities like you have on VAXes on one chip, there's no reason you can't have a complexity on the order of a Cray on one chip by that time. And the power per unit area would be about the same as it is today.

GENE: So the ultimate speed of the Von Neumann machine is about ten times what it is today, that is, ten times a Cray?

CARVER: No. You'd be just about getting in the range on one chip where Cray is today by the time you do all that. The speed limit on a single chip is about a Cray. That's a good number to keep in your head: you'll get about one Cray on a chip, including the clock rate and everything, by the time you push ordinary MOS devices as far as they're going to go. That is, if you choose to make a Cray. Which is an elaborated or extended version of the Von Neumann architecture with vector processing. So we're talking between 50 and 100 megaflops depending on how well its done.

GENE: And how much speedup will we get from new materials like gallium arsenide? A factor of ten more?

CARVER: I doubt it. First, fundamentally you can't make it as small as silicon. The reason gallium arsenide is fast is because the electron mobility is high. The reason the mobility is high is because the effective mass is low. This means the tunneling distances are longer. There's a direct relationship between effective mass and how far you can tunnel. Since tunneling is one of the things that restricts how small you can make devices, to the extent that that's the limiting factor -- and it does enter in submicron devices -- you'll bump against it sooner with gallium arsenide than you will with silicon. Exactly how all that comes out is not known right now. Nobody has yet done it. Meanwhile people are working on some very fancy things called superlattice structures which give extremely high mobilities. But exactly how the device structures are going to scale is something we need to know. These are compound layers whose lattices match and you can make the bandgap go up and down by a sort of stairstep thing. That makes a quantum effect on the electrons. It basically shapes the bandgap in a nice way and already McGill has worked out a theory for that and the measured shift in the photo-emission has turned out to agree exactly with his theory. That would be the HEMT device. That may get you the factor of ten or even more in terms of raw speed. But you can't take the numbers for individual devices and apply them to large-scale circuits because. For instance what's wrong with gallium arsenide technology right now is not the individual devices. They work better than you would have thought. They're better than my first predictions, because there's a fluke, there's a violation of Murphy's Law that happens. When an electron goes in under the gate you'd think it would come up to saturated velocity -- which is only a little better in gallium arsenide than in silicon -- but it turns out they do a ballistic overshoot, so they actually, for short channelings, go faster than the saturated velocity because they haven't started to have enough collisions yet. That was a violation of Murphy's Law of the first order. It was wonderful. The real problem with that kind of technology is that there's only one kind of device -- a depletion mode device. So you have to have the gate voltage of opposite polarity from the drain voltage, which means you have to level shift between every stage. It's horrible, just like the old vacuum tube things. And it turns out that ends up taking more space than the circuit by a lot. So it

isn't at all clear that the densities will ever be competitive with what you can do in silicon.

DICK: But for that same reason it would probably make possible very high density, very fast memories. And you might still be able to get a Cray-size Von Neumann machine on a chip because the memory is where most of it goes. In fact as you everything up and the amount of global wiring gets more and more dominant, the amount of memory gets bigger and bigger, the power of the arithmetic processor goes up a little bit and that's it. So the biggest thing goes into more wire, the next biggest thing goes into more memory, and the least gain is in the processor. This is what people have been doing. When you grow a Von Neumann machine in any technology that's what happens whether it's on a chip or not. Cray is a departure from that because they figured out how to put a lot more in the processing; but it's less Von Neumann-like than other machines.

CARVER: Cray is one of the people who from day one realized that, well, he said two things. He said computer design has two problems. One is how you get the heat out. The second is how you keep the thickness of the wiring mat within bounds. He thought the key to his design was getting the heat out. Well we can take care of making more complex things with less heat by just scaling down the silicon. So that part isn't the key; but the other part is, the wiring management. And he was the only major computer designer that I know of who specifically twenty years ago identified the thickness of the wiring mat as the dominant thing about computer design. Everybody else was saying well you have to have the x-y-z instruction and there's this and that, and he was saying no, it's the thickness of that wiring mat.

GENE: Will there ever be a Von Neumann machine with a one nanosecond clock.

CARVER: There might be. We already got you to under ten nano-seconds with our little chip at a quarter-micron. You could do it today with a small enough Von Neumann machine. A real simple machine. You'd come close to that today using submicron CMOS. They've already demonstrated .8-micron CMOS with 35-picosecond gates. You could no doubt make a credible one-nanosecond small Von Neumann machine. If it was only eight bits wide and had only three registers. Or if you built it like Cray did and pipeline the hell out of it. And we could certainly make these bit-serial things to run that fast. That's our music chips and Dick's speech chip, where you do one bit at a time. Because the signals only have to go to the next stage instead of going clear across the chip.

GENE: Is RISC architecture relevant only to Von Neumann architecture?

CARVER: Yes, and what it really means is something they did early on and forgot about later. People used to build all machines as reduced instruction sets because they couldn't afford anything else. The best example of that was the old Data General Nova, which had a real simple instruction set. Ever since then they built in ever more elaborate instruction sets, believing that it was going to make their computers run faster. And it turns out that what limits your computer is the load instruction and the store instruction and the index calculations. If you do those fast it doesn't much matter what else you do. And it doesn't at all matter how fast you do arithmetic. So people finally went back and rediscovered the fact that had been known for a long time by good computer designers: that it doesn't really much matter what you do with instructions if you get the basic ones real fast. So they said why put in all the others, why not just make them up out of these primitives? You have to have a compiler that's smart enough to do a decent job of that.

DICK: The other argument is that the machines with complicated instruction sets don't really get any advantage from them unless you have a compiler that's smart enough to figure out when to use complicated instructions. With the RISC you may get better performance even with a very simple compiler.

CARVER: But in fact nobody has yet done the real experiment of actually making a complete system with a complete operating system and everything on one of these RISC things and actually honest-to-god putting it side by side with some other computer and given it the smoke test. That'll no doubt happen but it's incredibly late in coming, because the figures people quote are concocted examples rather than real honest-to-god things happening.

GENE: Surely there will always be a supercomputer, because there'll always be the ten-million-dollar machine.

CARVER: Oh but you know what it's going to be? It's going to be one of these chips that has the Cray on it and a

bunch of memory and discs and all that, which will eat up a whole lot of money, and then it's going to be a box that has one kind of special purpose chip on it which does the signal processing things, and another card that has another kind of chip which does the geometry calculations, and another card with chips that do whatever else. When you talk about extending the instruction set...it's like right now when you buy a machine you get an arithmetic accelerator and you plug it in and it runs ten times as fast doing floating-point. So you'll have something that runs ten-thousand times as fast doing signal processing. And something that runs a million times as fast doing graphics rendering. And something that runs a hundred-thousand times as fast doing whatever else you're doing. There'll be these cards which are these accelerators for special classes of problems. And they'll plug into this thing. There will always be card cages and power supplies and fans (or cryogenic cooling systems) but the thing you plug in there is going to have a god-awful amount of capability compared with the little poor wizened Cray that's sitting on that one chip.

DICK: The interesting problem is to figure out how there will be ten-million-dollar machines. I mean, what part of that are going to spend your money on. I believe there will still be such machines but they'll probably be dominated by head-per-track discs or something like that. You'll spend nine out of the ten million dollars getting enough secondary storage and i/o bandwidth that's fast enough to keep up with the thing. Any one of our little chips can generate data a god-awful amount faster than any disc can soak it up. We solve the problem by just putting it out on a loudspeaker. We never store samples at all, ever; and it may be that for some applications you never would need to store all that data.

DICK: In the speech analysis problem, you come in with some sampled speech, which may be at a reasonably low rate like you get off a telephone system, maybe better quality. In the intermediate calculations you end up with places in your block diagram between the first block and the second block where the data rate may be 100 times higher than what you started out with. There's no way you're going to feed that back into your general purpose computer or put it on disc or anything else. But if you put specialized stuff out there to keep dealing with it until you've gotten close to an answer, until you've narrowed it down to what word was said, then you can throw away all that intermediate data. it's a tremendous amount of data but you don't need to keep it. You just look at it to extract the next stuff from it and you throw it away. Put it in the bit-bucket.

GENE: What you guys are doing is amazing.

DICK: It's happening so fast that even new engineering graduates aren't aware of the half of it.

GENE: Any given application always has a limiting case. In imaging, you don't need to compute more than what your display device can show.

CARVER: There's no point in making a picture faster than real time, any more than there is in me making samples faster than real time.

GENE: Alvy Ray Smith thinks you need a complexity of 80 million polygons per frame for photoreal natural phenomena.

CARVER: That's a Jim Kajiya measure: a complex scene has more than one polygon per pixel. Actually it turns out there are some very interesting problems associated with scenes that are much more complex than one polygon per pixel. Which is the kind our guys deal with.

GENE: So let's say we reach some arbitrary point beyond which no more computation power is needed, then you're saying that it's probably going to be possible to put that kind of power on a board and that's that.

CARVER: You just use it until you get a better idea of how to do it, better algorithms. The real interesting problem now is the origin of life problem, right? It's the thing I talked about with the "Newmath" company. They're working the origin of life problem. They want to design silicon algorithms so they're building things that'll help them do that -- and by the way they can sell that too, because other people will want to make silicon algorithms too -- people who want to make better graphics algorithms, who's going to help them? Because you've got to figure out if your ideas are right before you go and make the better graphics algorithm. So there's always going to be room for a thing that's more general than the specific thing because somebody has to invent the next one. So you'll need a development system that's an engineering work- station for people doing computer graphics, or speech or music -- which is not

really a general purpose computer but it more general than the final silicon algorithms it will produce.

DICK: This is exactly the idea that motivated my architecture: make a machine much more general than what I wanted to do. So that I could experiment with algorithms on that machine. So that the next time around I'll know what I want to do and I can put it in silicon.

GENE: So say we've got the limits of what we need to do on one card of specialized chips, and I've got my general purpose computer controlling it. So you're saying that at that point a supercomputer would be a system like that but one which would do all these things.

CARVER: And a lot of them. So that your instructions, instead of being a load instruction or a store instruction -- you still have those -- but what you really do is you say go Fourrier-transform this, go render this display list on the screen with the ray-tracing board, go analyze this speech with this other board.

GENE: But in fact a network in which those functions are distributed, each node with a particular specialty, would be equivalent to the supercomputer. You could synthesize a supercomputer by having small processors on a network.

CARVER: I think that thing you just said is going to replace the supercomputer as a concept: you have servers on a network. You have a server that does the Fourier analysis and a server that does the graphics rendering and so on, instead of thinking of it as being in one box. Because then it's just a much more convenient setup. So in fact I don't see any great big blue boxes with giant power supplies within a few years. I just don't see them at all. Because you can buy something that's got a thousand times the capability in one of these little boxes you hook on a network. That makes so much more sense.

GENE: So bandwidth is emerging as the major bottleneck.

CARVER: It always has been. It's a major opportunity that bigger bandwidths are available now. Fibers not only carry more, they go longer distances. The networks you're likely to find in a medium-sized company or research lab is the network that runs down the hall to the room where you've got all your specialized machines. So you can use this powerful machine from your own office. That kind of net- work is providing a lot of power to the people and you don't have to go out over the phone company to do any of that. That's a decision that can be made by a small company, not a big company. So what we're finding is that all of the definitions of these things are being done by the people who are building the stuff. And the government agencies and AT&T come along later and say, "Oh, gee! We didn't say it was OK." Well you didn't have to say it was OK, thank god; somebody got a chance to do it anyway. If we were waiting around for a network standard to emerge nobody would have any networks right now. In fact we've had networks since the mid-70s because people just went off and did them. When you leave the building, that's when the politics start. It's horrible.

GENE: Where are we now in speech recognition in relation to something that could be put into most user interfaces? Is this technology we've just been talking about leading to a solution of the user-independent recognition device within ten years.

DICK: Certainly within ten years there's going to be a lot happening. It's a dangerous field to make predictions about. If you look ten or twenty years ago people were making very optimistic predictions and not doing the right things. What's happened during that time is that, in many cases people have pretended that the problem is technology limited. So every time there's an advance in IC technology they make the next better thing, but it's still no good.

CARVER: It's just like microcomputers. The idea stream has not gotten better as fast as the technology has. So they make a much faster thing that's still lousy because it doesn't have any new ideas in it. That has happened a lot. Just like the microprocessor is an image of the ugly old computers of yesteryear.

DICK: When you record with just one microphone you lose a lot of information. So while your two ears don't have any trouble hearing me, this one microphone may have more difficulty. This is one area where there's a clear direction that's needed in the speech recognition business -- things that listen to people talking need to have two

microphones at least. They don't. None of them do. Because people haven't figured out how to take advantage of that. That's one of the things I'm working on.

CARVER: That's the stupidest most zero-order thing you can imagine, but it still hasn't been done to this day.

DICK: And one excuse they make is that it would cost twice as much silicon to analyze signals from two microphones. That's a totally lame excuse, because if you can make the problem work by throwing silicon at it you ought to. Because nothing else is working. But it's not that simple. You've got to have the ideas of what to do with the silicon other than put more memory in it and recognize more words poorly -- which is what people are doing.

The key thing that's motivating my own research is to figure out how people do it and model the human hearing system. That approach is not far enough along yet to show demonstrated good results, but it is far enough along to be extremely optimistic about it.

GENE: Not far enough on the biology side?

DICK: No, in my actual implementation of ideas. I can't yet show you a system that achieves spectacular performance. But I can show preliminary results that look encouraging.

GENE: Do we know how people do it?

DICK: We know a lot about how people do it, but there are still some open questions. There's an area that's not receiving enough research attention. There's a lot of research in the physiology of the hearing system and psychophysics, that kind of thing, but not enough on the computational approach -- trying to figure out what are the algorithms that the human is doing.

CARVER: In fact, work like the people here are doing is very much in the very near future, it will stimulate a lot of biology work because it will raise questions that have not been addressed by the biology community. Because they're not trying to build a system and unless you try to build it you simply don't get faced with those questions. So I think there's going to be a synergy between people like Dick and the biologists and the psycho-acoustic and psycho-vision people that is going to really...in fact both of us are spending a lot of our time doing that now because it's just a very gorgeous area that's almost untapped.

DICK: The biologists are starting to understand how important it is to look at the computational aspects, so we've formed some beginnings of some important collaborations with a couple of biologists.

CARVER: After all, there's this working computer that does this wonderful thing, that people haven't analyzed as a computer.

DICK: How do you compare how a microprocessor is put together to the way Carver's chip is put together. The best thing that works really well is a chip photo. Just show a picture of the chips. The difference between a regular design and a non-regular design doesn't need to be explained. It's just there, and strikingly obvious. Carver's chip is simple enough, that in addition to that you could actually write down the entirety of information that it contains. You could write the CIF-code (Caltech Intermediate Form) for the layout in five pages. It's a representation of geometry. You could have basically a printout of the computer text file from which the chip is created. it would be interesting to show the chip, the code that created the chip, and contrast that with the quantities of information that the chip produces. The leverage is the fact that you put a little bit of information into it to make the chip, and the chip is able to produce back huge quantities of information every second, which sound like music. It's an information amplifier. So the pictures and the listing, even though it's entirely unintelligible, would be interesting to see that there aren't too many bits there.

End of Carver 4